ReadTheDocs-Breathe Documentation

Release 1.0.0

Thomas Edvalson

Jun 03, 2020
## 1 The AMD ROCm Programming-Language Run-Time

## 2 Solid Compilation Foundation and Language Support

2.1 AMD Radeon Open Compute platform (ROCm) Release Notes v3.5.0

2.1.1 AMD ROCm Release Notes v3.5.0

2.1.2 Supported Operating Systems and Documentation Updates

2.1.2.1 Supported Operating Systems

2.1.2.2 Documentation Updates

2.1.2.2.1 HIP-Clang Compiler

2.1.2.2.2 AMD ROCDebugger (ROCgdb)

2.1.2.2.3 AMD ROCm Systems Management Interface

2.1.2.2.4 AMD ROCm Deep Learning

2.1.2.2.5 AMD ROCm Glossary of Terms

2.1.2.2.6 General AMD ROCm Documentation Links

2.1.3 What’s New in This Release

2.1.3.1 Upgrading to This Release

2.1.3.2 rocProf Command Line Tool Python Requirement

2.1.3.3 Heterogeneous-Compute Interface for Portability

2.1.3.4 Radeon Open Compute Common Language Runtime

2.1.3.5 OpenCL Runtime

2.1.3.6 AMD ROCm GNU Debugger - ROCgdb

2.1.3.7 AMD ROCm Debugger API Library

2.1.3.8 rocProfiler Dispatch Callbacks Start Stop API

2.1.3.9 ROCm Communications Collective Library

2.1.3.10 MIOpen - Optional Kernel Package Installation

2.1.3.11 New SMI Event Interface and Library

2.1.3.12 API for CPU Affinity

2.1.3.13 Radeon Performance Primitives Library

2.1.4 Fixed Issues

2.1.4.1 Device printf Support for HIP-Clang

2.1.4.2 Assertions in HIP Device Code

2.1.5 Known Issues

2.1.5.1 HIPify-Clang Installation Failure on CentOS/RHEL

2.1.5.2 Failure to Process Breakpoint before Queue Destroy Results in ROCm Debugger Error

2.1.5.3 Failure to Process Breakpoint before Queue Destroy Results in ROCm Debugger API Error

2.1.5.4 rocThrust and hipCUB Unit Test Failures

2.1.5.5 Multiple GPU Configuration Freezes with Imagenet Training and tf_cnn_benchmark on TensorFlow
2.1.5.6 Issue with Running AMD ROCm v3.3 User Mode with AMD ROCm v3.5 DKMS Kernel Module ......................................................... 13
2.1.6 Deprecations ........................................................................ 13
  2.1.6.1 Heterogeneous Compute Compiler .................................... 13
2.1.7 Deploying ROCm .................................................................. 13
2.1.8 DISCLAIMER ........................................................................ 13
2.2 AMD ROCm Installation Guide v3.5.0 ........................................ 14
  2.2.1 Deploying ROCm ................................................................. 14
    2.2.1.1 Ubuntu ......................................................................... 14
      2.2.1.1.1 Installing a ROCm Package from a Debian Repository .... 14
      2.2.1.1.2 Uninstalling ROCm Packages from Ubuntu ................ 15
      2.2.1.1.3 Installing Development Packages for Cross Compilation 15
      2.2.1.1.4 Using Debian-based ROCm with Upstream Kernel Drivers 16
    2.2.1.2 CentOS v7.7/RHEL v7.8 and CentOS/RHEL v8.1 ................ 16
      2.2.1.2.1 Preparing RHEL for Installation ............................ 16
    2.2.1.3 Installing CentOS v7.7/v8.1 for DKMS ......................... 17
      2.2.1.3.1 Installing ROCm ................................................... 17
      2.2.1.3.2 Testing the ROCm Installation .............................. 18
      2.2.1.3.3 Compiling Applications Using HCC, HIP, and Other ROCm Software 18
      2.2.1.3.4 Uninstalling ROCm from CentOS/RHEL .................... 18
      2.2.1.3.5 Installing Development Packages for Cross Compilation 18
      2.2.1.3.6 Using ROCm with Upstream Kernel Drivers ................ 18
    2.2.1.4 SLES 15 Service Pack 1 ................................................ 19
      2.2.1.4.1 Performing an OpenCL-only Installation of ROCm ....... 19
  2.2.2 AMD ROCm MultiVersion Installation ................................... 20
  2.2.3 ROCm Installation Known Issues and Workarounds ................. 22
  2.2.4 Getting the ROCm Source Code ........................................... 23
    2.2.4.1 Installing the Repo .................................................... 23
  2.2.5 Downloading the ROCm Source Code .................................... 23
  2.2.6 Building the ROCm Source Code ......................................... 23
2.3 Machine Learning and High Performance Computing Software Stack for AMD GPU v3.5.0 ................. 23
  2.3.1 Machine Learning and High Performance Computing Software Stack for AMD GPU v3.5.0 23
    2.3.1.1 ROCm Binary Package Structure .................................. 24
    2.3.1.2 ROCm Platform Packages ........................................... 26
2.4 AMD ROCm Version History .................................................. 27
  2.4.1 New features and enhancements in ROCm v3.3 ........................ 27
  2.4.2 New features and enhancements in ROCm v3.2 ........................ 28
  2.4.3 New features and enhancements in ROCm v3.1 ........................ 28
  2.4.4 New features and enhancements in ROCm v3.0 ........................ 28
  2.4.5 New features and enhancements in ROCm v2.10 ....................... 29
  2.4.6 New features and enhancements in ROCm 2.9 .......................... 29
  2.4.7 New features and enhancements in ROCm 2.8 .......................... 30
  2.4.8 New features and enhancements in ROCm 2.7.2 ........................ 30
  2.4.9 Issues fixed in ROCm 2.7.2 .............................................. 30
  2.4.10 Upgrading from ROCm 2.7 to 2.7.2 .................................... 30
  2.4.11 New features and enhancements in ROCm 2.6 .......................... 31
  2.4.12 New features and enhancements in ROCm 2.5 .......................... 32
  2.4.13 New features and enhancements in ROCm 2.4 .......................... 33
  2.4.14 New features and enhancements in ROCm 2.3 .......................... 33
  2.4.15 New features and enhancements in ROCm 2.2 .......................... 34
  2.4.16 New features and enhancements in ROCm 2.1 .......................... 34
  2.4.17 New features and enhancements in ROCm 2.0 .......................... 34
  2.4.18 New features and enhancements in ROCm 1.9.2 ......................... 35
  2.4.19 New features and enhancements in ROCm 1.9.1 ......................... 35
2.12.1.1 Prerequisites ................................................................. 78
2.12.1.2 Installing pre-built packages ........................................ 78
2.12.1.3 Quickstart rocBLAS build ............................................. 78
2.12.1.4 Manual build (all supported platforms) .............................. 78
2.12.1.5 rocBLAS interface examples .......................................... 79
2.12.1.6 GEMV API ................................................................. 79
2.12.1.7 Batched and strided GEMM API ...................................... 79
2.12.1.8 Asynchronous API ....................................................... 79
2.12.1.9 API ........................................................................... 80
  2.12.1.9.1 Types ..................................................................... 80
    2.12.1.9.1.1 Definitions ...................................................... 80
    2.12.1.9.1.2 rocblas_int ..................................................... 80
    2.12.1.9.1.3 rocblas_stride ................................................. 80
    2.12.1.9.1.4 rocblas_half .................................................. 80
    2.12.1.9.1.5 rocblas_handle ............................................... 80
    2.12.1.9.1.6 Enums ............................................................ 80
    2.12.1.9.1.7 rocblas_operation ........................................... 80
    2.12.1.9.1.8 rocblas_fill ................................................... 81
    2.12.1.9.1.9 rocblas_diagonal ............................................. 81
    2.12.1.9.1.10 rocblas_side .................................................. 81
    2.12.1.9.1.11 rocblas_status ............................................... 81
    2.12.1.9.1.12 rocblas_datatype .......................................... 80
    2.12.1.9.1.13 rocblas_pointer_mode .................................... 82
    2.12.1.9.1.14 rocblas_layer_mode ....................................... 83
    2.12.1.9.1.15 rocblas_gemm_algo ....................................... 83
  2.12.1.9.2 Functions .............................................................. 84
    2.12.1.9.2.1 Level 1 BLAS ................................................... 84
    2.12.1.9.2.2 rocblas_<type>scal() ....................................... 84
    2.12.1.9.2.3 rocblas_<type>scal_batched() .............................. 85
    2.12.1.9.2.4 rocblas_<type>scal_strided_batched() .................. 85
    2.12.1.9.2.5 rocblas_<type>copy() ........................................ 86
    2.12.1.9.2.6 rocblas_<type>copy_batched() ............................. 87
    2.12.1.9.2.7 rocblas_<type>copy_strided_batched() .................. 88
    2.12.1.9.2.8 rocblas_<type>dot() .......................................... 89
    2.12.1.9.2.9 rocblas_<type>dot_batched() .............................. 90
    2.12.1.9.2.10 rocblas_<type>dot_strided_batched() ................... 91
    2.12.1.9.2.11 rocblas_<type>swap() ...................................... 92
    2.12.1.9.2.12 rocblas_<type>swap_batched() ........................... 92
    2.12.1.9.2.13 rocblas_<type>swap_strided_batched() ................ 93
    2.12.1.9.2.14 rocblas_<type>axpy() ...................................... 94
    2.12.1.9.2.15 rocblas_<type>asum() ...................................... 95
    2.12.1.9.2.16 rocblas_<type>asum_batched() ........................... 95
    2.12.1.9.2.17 rocblas_<type>asum_strided_batched() ................ 96
    2.12.1.9.2.18 rocblas_<type>nrm2() ..................................... 96
    2.12.1.9.2.19 rocblas_<type>nrm2_batched() ........................... 97
    2.12.1.9.2.20 rocblas_<type>nrm2_strided_batched() ................ 98
    2.12.1.9.2.21 rocblas_<type>amax() ...................................... 99
    2.12.1.9.2.22 rocblas_<type>amax_batched() ........................... 100
    2.12.1.9.2.23 rocblas_<type>amax_strided_batched() ................ 100
    2.12.1.9.2.24 rocblas_<type>amin() ...................................... 101
    2.12.1.9.2.25 rocblas_<type>amin_batched() ........................... 101
    2.12.1.9.2.26 rocblas_<type>amin_strided_batched() ................ 102
    2.12.1.9.2.27 rocblas_<type>rot() ....................................... 103
    2.12.1.9.2.28 rocblas_<type>rot_batched() ............................ 104
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rocblas_&lt;type&gt;rot_strided_batched()</td>
<td>105</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotg()</td>
<td>106</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotg_batched()</td>
<td>106</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotg_strided_batched()</td>
<td>107</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotm()</td>
<td>108</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotm_batched()</td>
<td>109</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotm_strided_batched()</td>
<td>109</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotmg()</td>
<td>110</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotmg_batched()</td>
<td>110</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;rotmg_strided_batched()</td>
<td>111</td>
</tr>
<tr>
<td>Level 2 BLAS</td>
<td>112</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;gemv()</td>
<td>112</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;hemv()</td>
<td>113</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;hemv_batched()</td>
<td>113</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;hemv_strided_batched()</td>
<td>114</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;gemv_batched()</td>
<td>114</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;gemv_strided_batched()</td>
<td>115</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trmv()</td>
<td>116</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trmv_batched()</td>
<td>116</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trmv_strided_batched()</td>
<td>117</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;ger()</td>
<td>117</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;ger_batched()</td>
<td>118</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;ger_strided_batched()</td>
<td>118</td>
</tr>
<tr>
<td>Level 3 BLAS</td>
<td>119</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;syr()</td>
<td>119</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;syr_batched()</td>
<td>120</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;syr_strided_batched()</td>
<td>120</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trtri()</td>
<td>121</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trtri_batched()</td>
<td>121</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trtri_strided_batched()</td>
<td>121</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trsm()</td>
<td>122</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trsm_batched()</td>
<td>122</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trsm_strided_batched()</td>
<td>123</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;trmm()</td>
<td>124</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;gemm()</td>
<td>124</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;gemm_batched()</td>
<td>125</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;gemm_strided_batched()</td>
<td>126</td>
</tr>
<tr>
<td>rocblas_&lt;type&gt;geam()</td>
<td>126</td>
</tr>
<tr>
<td>BLAS Extensions</td>
<td>127</td>
</tr>
<tr>
<td>rocblas_gemm_ex()</td>
<td>128</td>
</tr>
<tr>
<td>rocblas_trsm_ex()</td>
<td>128</td>
</tr>
<tr>
<td>rocblas_trsm_strided_batched_ex()</td>
<td>129</td>
</tr>
<tr>
<td>Build Information</td>
<td>130</td>
</tr>
<tr>
<td>rocblas_get_version_string()</td>
<td>130</td>
</tr>
<tr>
<td>Auxiliary</td>
<td>130</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td><code>rocspatial_destroy_mat_descr()</code></td>
<td>272</td>
</tr>
<tr>
<td><code>rocspatial_set_mat_type()</code></td>
<td>273</td>
</tr>
<tr>
<td><code>rocspatial_set_mat_fill_mode()</code></td>
<td>273</td>
</tr>
<tr>
<td><code>rocspatial_set_mat_index_base()</code></td>
<td>273</td>
</tr>
<tr>
<td><code>rocspatial_get_mat_index_base()</code></td>
<td>273</td>
</tr>
<tr>
<td><code>rocspatial_set_mat_type()</code></td>
<td>274</td>
</tr>
<tr>
<td><code>rocspatial_get_mat_type()</code></td>
<td>274</td>
</tr>
<tr>
<td><code>rocspatial_set_mat_fill_mode()</code></td>
<td>274</td>
</tr>
<tr>
<td><code>rocspatial_get_mat_fill_mode()</code></td>
<td>274</td>
</tr>
<tr>
<td><code>rocspatial_set_mat_diag_type()</code></td>
<td>275</td>
</tr>
<tr>
<td><code>rocspatial_get_mat_diag_type()</code></td>
<td>275</td>
</tr>
<tr>
<td><code>rocspatial_create_hyb_mat()</code></td>
<td>276</td>
</tr>
<tr>
<td><code>rocspatial_destroy_hyb_mat()</code></td>
<td>276</td>
</tr>
<tr>
<td><code>rocspatial_create_mat_info()</code></td>
<td>276</td>
</tr>
<tr>
<td><code>rocspatial_destroy_mat_info()</code></td>
<td>277</td>
</tr>
<tr>
<td><code>rocspatial_axpyi()</code></td>
<td>277</td>
</tr>
<tr>
<td><code>rocspatial_doti()</code></td>
<td>279</td>
</tr>
<tr>
<td><code>rocspatial_dotci()</code></td>
<td>282</td>
</tr>
<tr>
<td><code>rocspatial_gthr()</code></td>
<td>283</td>
</tr>
<tr>
<td><code>rocspatial_gthrz()</code></td>
<td>285</td>
</tr>
<tr>
<td><code>rocspatial_roti()</code></td>
<td>287</td>
</tr>
<tr>
<td><code>rocspatial_sctr()</code></td>
<td>288</td>
</tr>
<tr>
<td><code>rocspatial_coomv()</code></td>
<td>290</td>
</tr>
<tr>
<td><code>rocspatial_csrmv_analysis()</code></td>
<td>294</td>
</tr>
<tr>
<td><code>rocspatial_csrmv()</code></td>
<td>297</td>
</tr>
<tr>
<td><code>rocspatial_csrmv_analysis_clear()</code></td>
<td>304</td>
</tr>
<tr>
<td><code>rocspatial_ellmv()</code></td>
<td>304</td>
</tr>
<tr>
<td><code>rocspatial_hybmv()</code></td>
<td>309</td>
</tr>
<tr>
<td><code>rocspatial_cross_zero_pivot()</code></td>
<td>312</td>
</tr>
<tr>
<td><code>rocspatial_crossv_buffer_size()</code></td>
<td>312</td>
</tr>
<tr>
<td><code>rocspatial_crossv_analysis()</code></td>
<td>316</td>
</tr>
<tr>
<td><code>rocspatial_crossv_solve()</code></td>
<td>319</td>
</tr>
<tr>
<td><code>rocspatial_crossv_clear()</code></td>
<td>327</td>
</tr>
<tr>
<td><code>rocspatial_csr2coo()</code></td>
<td>336</td>
</tr>
<tr>
<td><code>rocspatial_coo2csr()</code></td>
<td>336</td>
</tr>
<tr>
<td><code>rocspatial_csr2csc_buffer_size()</code></td>
<td>342</td>
</tr>
<tr>
<td><code>rocspatial_csr2csc_buffer_size()</code></td>
<td>344</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell_width()</code></td>
<td>358</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell()</code></td>
<td>358</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell_buffer_size()</code></td>
<td>358</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell_analysis()</code></td>
<td>361</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell0()</code></td>
<td>365</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell_clear()</code></td>
<td>377</td>
</tr>
<tr>
<td><code>rocspatial_csr2coo()</code></td>
<td>378</td>
</tr>
<tr>
<td><code>rocspatial_coo2csr()</code></td>
<td>379</td>
</tr>
<tr>
<td><code>rocspatial_csr2csc_buffer_size()</code></td>
<td>381</td>
</tr>
<tr>
<td><code>rocspatial_csr2csc()</code></td>
<td>382</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell_width()</code></td>
<td>388</td>
</tr>
<tr>
<td><code>rocspatial_csr2ell()</code></td>
<td>389</td>
</tr>
<tr>
<td><code>rocspatial_ell2csr_nnz()</code></td>
<td>394</td>
</tr>
</tbody>
</table>
2.12.5.13.8 rocsparse_ell2csr() ........................................ 395
2.12.5.13.9 rocsparse_csr2hyb() .................................. 401
2.12.5.13.10 rocsparse_create_identity_permutation() ............ 405
2.12.5.13.11 rocsparse_csrsort_buffer_size() ....................... 406
2.12.5.13.12 rocsparse_csrsort() .................................. 406
2.12.5.13.13 rocsparse_coorosort_buffer_size() ................... 408
2.12.5.13.14 rocsparse_coorosort_by_row() ......................... 409
2.12.5.13.15 rocsparse_coorosort_by_column() ..................... 411
2.12.6 rocSOLVER .......................................................... 412
2.12.6.1 Introduction ..................................................... 412
2.12.6.2 Build and install ................................................. 413
2.12.6.3 Brief description and functionality .......................... 413
2.12.6.4 Benchmarking and Testing ..................................... 414
2.12.6.5 rocSOLVER API ................................................... 415
2.12.6.5.1 Definitions .................................................. 415
2.12.6.5.2 Enums ......................................................... 415
2.12.6.5.2.1 rocsolver_int ........................................... 415
2.12.6.5.2.2 rocsolver_operation .................................... 415
2.12.6.5.2.3 rocsolver_fill ........................................... 415
2.12.6.5.2.4 rocsolver_diagonal ..................................... 415
2.12.6.5.2.5 rocsolver_side ........................................... 415
2.12.6.5.2.6 rocsolver_direct ......................................... 416
2.12.6.5.2.7 rocsolver_storev ......................................... 416
2.12.6.5.2.8 rocsolver_status .......................................... 416
2.12.6.5.3 Matrix permutations and manipulations .................. 416
2.12.6.5.3.1 rocsolver_<type>laswp() ................................ 416
2.12.6.5.4 Householder reflexions .................................... 417
2.12.6.5.4.1 rocsolver_<type>larfg() ................................ 417
2.12.6.5.4.2 rocsolver_<type>larft() ................................ 418
2.12.6.5.4.3 rocsolver_<type>larf() ................................ 419
2.12.6.5.4.4 rocsolver_<type>larfb() ................................ 420
2.12.6.5.5 Orthonormal matrices ..................................... 421
2.12.6.5.5.1 rocsolver_<type>org2r() ................................ 421
2.12.6.5.5.2 rocsolver_<type>orgqr() ................................ 422
2.12.6.5.5.3 rocsolver_<type>orgl2() ................................ 422
2.12.6.5.5.4 rocsolver_<type>orgql() ................................ 423
2.12.6.5.5.5 rocsolver_<type>orgbr() ................................ 424
2.12.6.5.5.6 rocsolver_<type>orm2r() ................................ 425
2.12.6.5.5.7 rocsolver_<type>ormqr() ................................ 426
2.12.6.5.6 Special Matrix Factorizations ............................. 427
2.12.6.5.6.1 rocsolver_<type>potf2() ................................ 427
2.12.6.5.6.2 rocsolver_<type>potf2_batched() ....................... 427
2.12.6.5.6.3 rocsolver_<type>potf2_strided_batched() .............. 428
2.12.6.5.6.4 rocsolver_<type>potrf() ................................ 429
2.12.6.5.6.5 rocsolver_<type>potrf_batched() ....................... 429
2.12.6.5.6.6 rocsolver_<type>potrf_strided_batched() ............... 430
2.12.6.5.7 General Matrix Factorizations ............................ 431
2.12.6.5.7.1 rocsolver_<type>getf2() ................................ 431
2.12.6.5.7.2 rocsolver_<type>getf2_batched() ....................... 432
2.12.6.5.7.3 rocsolver_<type>getf2_strided_batched() .............. 433
2.12.6.5.7.4 rocsolver_<type>getrf() ................................ 434
2.12.6.5.7.5 rocsolver_<type>getrf_batched() ....................... 435
2.12.6.5.7.6 rocsolver_<type>getrf_strided_batched() ............... 436
2.12.6.5.7.7 rocsolver_<type>geqr2() ........................................ 437
2.12.6.5.7.8 rocsolver_<type>geqr2_batched() ............................ 438
2.12.6.5.7.9 rocsolver_<type>geqr2_strided_batched() ................ 439
2.12.6.5.7.10 rocsolver_<type>geqrf() .................................... 440
2.12.6.5.7.11 rocsolver_<type>geqrf_batched() ........................... 441
2.12.6.5.7.12 rocsolver_<type>geqrf_strided_batched() ................ 442
2.12.6.5.7.13 rocsolver_<type>gelq2() .................................... 443
2.12.6.5.7.14 rocsolver_<type>gelq2_batched() ........................... 443
2.12.6.5.7.15 rocsolver_<type>gelq2_strided_batched() ................ 444
2.12.6.5.7.16 rocsolver_<type>gelqf() .................................... 445
2.12.6.5.7.17 rocsolver_<type>gelqf_batched() ........................... 446
2.12.6.5.7.18 rocsolver_<type>gelqf_strided_batched() ................ 447

2.12.6.5.8 General systems solvers ........................................ 448
2.12.6.5.8.1 rocsolver_<type>getrs() ..................................... 448
2.12.6.5.8.2 rocsolver_<type>getrs_batched() ............................ 449
2.12.6.5.8.3 rocsolver_<type>getrs_strided_batched() ................. 450

2.12.6.5.9 rocSOLVER handle auxiliaries .................................. 452
2.12.6.5.9.1 rocsolver_create_handle() .................................. 452
2.12.6.5.9.2 rocsolver_destroy_handle() ................................. 452
2.12.6.5.9.3 rocsolver_add_stream() ..................................... 452
2.12.6.5.9.4 rocsolver_set_stream() ..................................... 452
2.12.6.5.9.5 rocsolver_get_stream() ..................................... 452

2.12.6.5.10 Other auxiliaries .............................................. 452
2.12.6.5.10.1 rocsolver_set_vector() ..................................... 452
2.12.6.5.10.2 rocsolver_get_vector() ..................................... 452
2.12.6.5.10.3 rocsolver_set_matrix() ..................................... 453
2.12.6.5.10.4 rocsolver_get_matrix() ..................................... 453

2.12.7 hipSPARSE .............................................................. 453
2.12.7.1 Installing pre-built packages .................................... 453
2.12.7.2 Quickstart hipSPARSE build ...................................... 453
2.12.7.3 Manual build (all supported platforms) ......................... 453
2.12.7.4 Functions supported ............................................... 454
2.12.7.5 hipSPARSE interface examples ..................................... 454
2.12.7.6 CSR MV API .......................................................... 454

2.12.8 rocALUTION ............................................................. 454
2.12.8.1 Introduction .......................................................... 454
2.12.8.2 Overview ............................................................. 454
2.12.8.3 Building and Installing ............................................ 455
2.12.8.4 Installing from AMD ROCm repositories ......................... 455
2.12.8.5 Building rocALUTION from Github repository .................. 456
2.12.8.6 Download rocALUTION .............................................. 456
2.12.8.7 Using install.sh to build dependencies + library ............. 456
2.12.8.8 Using install.sh to build dependencies + library + client .... 457
2.12.8.9 Using individual commands to build rocALUTION ............. 458
2.12.8.10 Common build problems .......................................... 459
2.12.8.11 Simple Test ........................................................ 459
2.12.8.12 API ................................................................. 459
2.12.8.12.1 Host Utility Functions ........................................ 460
2.12.8.12.2 Backend Manager .............................................. 461
2.12.8.12.3 Base Roculation ............................................... 463
2.12.8.12.4 Operator ........................................................ 464
2.12.8.12.5 Vector .......................................................... 465
2.12.8.12.6 Local Matrix .................................................... 473
2.12.8.12.7 Local Stencil .................................................... 492
Index

2.22.4.1 Introduction ................................................. 617
2.22.4.1.1 IPC API .................................................. 617
2.23 Tutorial .......................................................... 620
2.24 ROCm Glossary ................................................. 620

Index 623
AMD ROCm is the first open-source software development platform for HPC/Hyperscale-class GPU computing. AMD ROCm brings the UNIX philosophy of choice, minimalism and modular software development to GPU computing.

AMD ROCm is built for scale; it supports multi-GPU computing in and out of server-node communication through RDMA. AMD ROCm also simplifies the stack when the driver directly incorporates RDMA peer-sync support.
THE AMD ROCM PROGRAMMING-LANGUAGE RUN-TIME

The AMD ROCr System Runtime is language independent and makes heavy use of the Heterogeneous System Architecture (HSA) Runtime API. This approach provides a rich foundation to execute programming languages such as HCC C++ and HIP.

Important features include the following:

- Multi-GPU coarse-grain shared virtual memory
- Process concurrency and preemption
- Large memory allocations
- HSA signals and atomics
- User-mode queues and DMA
- Standardized loader and code-object format
- Dynamic and offline-compilation support
- Peer-to-peer multi-GPU operation with RDMA support
- Profiler trace and event-collection API
- Systems-management API and tools
Support on Metal

- Compiler
- BLAS
- FFT
- SPARSE
- RAND
- Deep Learning Solver
- Collective Communication Lib
- RDMA
- UCX
- MPI

Language RT API

System RT API

Core Linux Driver

- RHEL/CentOS
- SuSE
- Ubuntu
CHAPTER TWO

SOLID COMPILATION FOUNDATION AND LANGUAGE SUPPORT

• LLVM compiler foundation
• HCC C++ and HIP for application portability
• GCN assembler and disassembler

AMD ROCm gives developers the flexibility of choice for hardware and aids in the development of compute-intensive applications.

2.1 AMD Radeon Open Compute platform (ROCM) Release Notes v3.5.0

June, 2020

2.1.1 AMD ROCm Release Notes v3.5.0

This page describes the features, fixed issues, and information about downloading and installing the ROCm software. It also covers known issues and deprecated features in the ROCm v3.5.0 release.

2.1.2 Supported Operating Systems and Documentation Updates

2.1.2.1 Supported Operating Systems

The AMD ROCm v3.5.x platform is designed to support the following operating systems:
• Ubuntu 16.04.6(Kernel 4.15) and 18.04.4(Kernel 5.3)
• CentOS 7.7 (Kernel 3.10-1062) and RHEL 7.8(Kernel 3.10.0-1127)(Using devtoolset-7 runtime support)
• SLES 15 SP1
• CentOS and RHEL 8.1(Kernel 4.18.0-147)
2.1.2.2 Documentation Updates

2.1.2.2.1 HIP-Clang Compiler

- HIP FAQ - Transition from HCC to HIP-Clang
- HIP-Clang Porting Guide
- HIP - Glossary of Terms

2.1.2.2.2 AMD ROCDebugger (ROCgdb)

- ROCgdb User Guide
- ROCgdbapi Guide

2.1.2.2.3 AMD ROCm Systems Management Interface

- System Management Interface Event API Guide

2.1.2.2.4 AMD ROCm Deep Learning

- MIOpen API

2.1.2.2.5 AMD ROCm Glossary of Terms

- Updated Glossary of Terms and Definitions

2.1.2.2.6 General AMD ROCm Documentatin Links

Access the following links for more information on:

- For AMD ROCm documentation, see https://rocmdocs.amd.com/en/latest/
2.1.3 What’s New in This Release

2.1.3.1 Upgrading to This Release

You must perform a fresh and a clean AMD ROCm install to successfully upgrade from v3.3 to v3.5. The following changes apply in this release:

- HCC is deprecated and replaced with the HIP-Clang compiler
- HIP-HCC runtime is changed to Radeon Open Compute Common Language Runtime (HIP-ROCClr)
- In the v3.5 release, the firmware is separated from the kernel package. The difference is as follows:
  - v3.5 release has two separate rock-dkms and rock-dkms-firmware packages
  - v3.3 release had the firmware as part of the rock-dkms package

2.1.3.2 rocProf Command Line Tool Python Requirement

SQLite3 is a required Python module for the rocprof command-line tool. You can install the SQLite3 Python module using the pip utility and set env var ROCP_PYTHON_VERSION to the Python version, which includes the SQLite3 module.

2.1.3.3 Heterogeneous-Compute Interface for Portability

In this release, the Heterogeneous Compute Compiler (HCC) compiler is deprecated and the HIP-Clang compiler is introduced for compiling Heterogeneous-Compute Interface for Portability (HIP) programs.

NOTE: The HCC environment variables will be gradually deprecated in subsequent releases.

The majority of the codebase for the HIP-Clang compiler has been upstreamed to the Clang trunk. The HIP-Clang implementation has undergone a strict code review by the LLVM/Clang community and comprehensive tests consisting of LLVM/Clang build bots. These reviews and tests resulted in higher productivity, code quality, and lower cost of maintenance.
For most HIP applications, the transition from HCC to HIP-Clang is transparent and efficient as the HIPCC and HIP cmake files automatically choose compilation options for HIP-Clang and hide the difference between the HCC and HIP-Clang code. However, minor changes may be required as HIP-Clang has a stricter syntax and semantic checks compared to HCC.

NOTE: Native HCC language features are no longer supported.

### 2.1.3.4 Radeon Open Compute Common Language Runtime

In this release, the HIP runtime API is implemented on top of Radeon Open Compute Common Language Runtime (ROCclr). ROCclr is an abstraction layer that provides the ability to interact with different runtime backends such as ROCr.

### 2.1.3.5 OpenCL Runtime

The following OpenCL runtime changes are made in this release:

- AMD ROCm OpenCL Runtime extends support to OpenCL2.2
- The developer branch is changed from master to master-next

### 2.1.3.6 AMD ROCm GNU Debugger - ROCgdb

The AMD ROCm GNU Debugger (ROCgdb) is the AMD ROCm source-level debugger for Linux based on the GNU Debugger (GDB). It enables heterogeneous debugging on the AMD ROCm platform of an x86-based host architecture along with AMD GPU architectures and supported by the AMD Debugger API Library (ROCdbgapi).

The AMD ROCm Debugger is installed by the rocm-gdb package. The rocm-gdb package is part of the rocm-dev meta-package, which is in the rocm-dkms package.

The current AMD ROCm Debugger (ROCgdb) is an initial prototype that focuses on source line debugging. Note, symbolic variable debugging capabilities are not currently supported.

You can use the standard GDB commands for both CPU and GPU code debugging. For more information about ROCgdb, refer to the ROCgdb User Guide, which is installed at:

- /opt/rocm/share/info/gdb.info as a texinfo file
- /opt/rocm/share/doc/gdb/gdb.pdf as a PDF file

The AMD ROCm Debugger User Guide is available as a PDF at:

https://github.com/RadeonOpenCompute/ROCm/blob/master/gdb.pdf

For more information about GNU Debugger (GDB), refer to the GNU Debugger (GDB) web site at: http://www.gnu.org/software/gdb
2.1.3.7 AMD ROCm Debugger API Library

The AMD ROCm Debugger API Library (ROCdbgapi) implements an AMD GPU debugger application programming interface (API) that provides the support necessary for a client of the library to control the execution and inspect the state of AMD GPU devices.

The following AMD GPU architectures are supported: * Vega 10 * Vega 7nm

The AMD ROCm Debugger API Library is installed by the rocm-dbgapi package. The rocm-gdb package is part of the rocm-dev meta-package, which is in the rocm-dkms package. The AMD ROCm Debugger API Specification is available as a PDF at:


2.1.3.8 rocProfiler Dispatch Callbacks Start Stop API

In this release, a new rocprofiler start/stop API is added to enable/disable GPU kernel HSA dispatch callbacks. The callback can be registered with the â€”rocprofiler_set_hsa_callbacksâ€™ API. The API helps you eliminate some profiling performance impact by invoking the profiler only for kernel dispatches of interest. This optimization will result in significant performance gains.

The API provides the following functions: * hsa_status_t rocprofiler_start_queue_callbacks(); is used to start profiling * hsa_status_t rocprofiler_stop_queue_callbacks(); is used to stop profiling.

For more information on kernel dispatches, see the HSA Platform System Architecture Specification guide at http://www.hsafoundation.com/standards/.

2.1.3.9 ROCm Communications Collective Library

The ROCm Communications Collective Library (RCCL) consists of the following enhancements: * Re-enable target 0x803 * Build time improvements for the HIP-Clang compiler

NVIDIA Communications Collective Library Version Compatibility

AMD RCCL is now compatible with NVIDIA Communications Collective Library (NCCL) v2.6.4 and provides the following features: * Network interface improvements with API v3 * Network topology detection * Improved CPU type detection * Infiniband adaptive routing support

2.1.3.10 MIOpen - Optional Kernel Package Installation

MIOpen provides an optional pre-compiled kernel package to reduce startup latency.

NOTE: The installation of this package is optional. MIOpen will continue to function as expected even if you choose to not install the pre-compiled kernel package. This is because MIOpen compiles the kernels on the target machine once the kernel is run. However, the compilation step may significantly increase the startup time for different operations.

To install the kernel package for your GPU architecture, use the following command:

`apt-get install miopen-kernels--`

- is the GPU architecture. For example, gfx900, gfx906
- is the number of CUs available in the GPU. For example, 56 or 64
2.1.3.11 New SMI Event Interface and Library

An SMI event interface is added to the kernel and ROCm SMI lib for system administrators to get notified when specific events occur. On the kernel side, AMDKFD_IOC_SMI_EVENTS input/output control is enhanced to allow notifications propagation to user mode through the event channel.

On the ROCm SMI lib side, APIs are added to set an event mask and receive event notifications with a timeout option. Further, ROCm SMI API details can be found in the PDF generated by Doxygen from source or by referring to the rocm_smi.h header file (see the rsmi_event_notification_* functions).

For the more details about ROCm SMI API, see


2.1.3.12 API for CPU Affinity

A new API is introduced for aiding applications to select the appropriate memory node for a given accelerator (GPU). The API for CPU affinity has the following signature:

```
rsmi_status_t rsmtopono_maffinity_get(uint32_t dv_ind, uint32_t numa_node);
```

This API takes as input, device index (dv_ind), and returns the NUMA node (CPU affinity), stored at the location pointed by numa_node pointer, associated with the device.

Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.

2.1.3.13 Radeon Performance Primitives Library

The new Radeon Performance Primitives (RPP) library is a comprehensive high-performance computer vision library for AMD (CPU and GPU) with the HIP and OpenCL backend. The target operating system is Linux.

For more information about prerequisites and library functions, see

https://github.com/GPUOpen-ProfessionalCompute-Libraries/MIVisionX/tree/master/docs
2.1.4 Fixed Issues

2.1.4.1 Device printf Support for HIP-Clang

HIP now supports the use of printf in the device code. The parameters and return value for the device-side printf follow the POSIX.1 standard, with the exception that the `\%n` specifier is not supported. A call to printf blocks the calling wavefront until the operation is completely processed by the host.

No host-side runtime calls by the application are needed to cause the output to appear. There is also no limit on the number of device-side calls to printf or the amount of data that is printed.


2.1.4.2 Assertions in HIP Device Code

Previously, a failing assertion caused early termination of kernels and the application to exit with a line number, file, and failing condition printed to the screen. This issue is now fixed and the assert() and abort() functions are implemented for HIP device code. NOTE: There may be a performance impact in the use of device assertions in its current form.

You may choose to disable the assertion in the production code. For example, to disable an assertion of:

```c
assert(foo != 0);
```

you may comment it out as:

```c
//assert(foo != 0);
```

NOTE: Assertions are currently enabled by default.

2.1.5 Known Issues

The following are the known issues in the v3.5 release.

2.1.5.1 HIPify-Clang Installation Failure on CentOS/RHEL

HIPify-Clang fails to install on CentOS/RHEL with the following error:

```plaintext
file from install of hipify-clang conflicts with file from package hip-base
```

**Workaround:** This is a known issue and the following workaround is recommended for a successful HIPify-Clang installation on CentOS/RHEL:

- Download HIPify-Clang RPM. For example, `hipify-clang-11.0.0.x86_64.rpm`
- Perform a force install using the following command:

```
sudo rpm -ivh --force hipify-clang-11.0.0.x86_64.rpm
```
2.1.5.2 Failure to Process Breakpoint before Queue Destroy Results in ROCm Debugger Error

When ROCgdb is in non-stop mode with an application that rapidly creates and destroys queues, a breakpoint may be reported that is not processed by the debugger before the queue is deleted. In some cases, this can result in the following error that prevents further debugging:

```
[amd-dbgapi]: fatal error: kfd_queue_id 2 should have been reported as a NEW_QUEUE before next_pending_event failed (rc=-2)
```

There are no known workarounds at this time.

2.1.5.3 Failure to Process Breakpoint before Queue Destroy Results in ROCm Debugger API Error

When the ROCdbgapi library is used with an application that rapidly creates and destroys queues, a breakpoint may be reported that is not processed by the client before the queue is deleted. In some cases, this can result in a fatal error and the following error log message is produced:

```
[amd-dbgapi]: fatal error: kfd_queue_id 2 should have been reported as a NEW_QUEUE before next_pending_event failed (rc=-2)
```

There are no known workarounds at this time.

2.1.5.4 rocThrust and hipCUB Unit Test Failures

The following unit test failures have been observed due to known issues in the ROCclr runtime.

rocThrust
- sort
- sort_by_key

hipCUB
- BlockDiscontinuity
- BlockExchange
- BlockHistogram
- BlockRadixSort
- BlockReduce
- BlockScan

There are no known workarounds in the current release.

2.1.5.5 Multiple GPU Configuration Freezes with Imagenet Training and tf_cnn_benchmark on TensorFlow

A random freeze has been observed with Imagenet training and tf_cnn_benchmark on TensorFlow when multiple GPU configurations are involved.

Note: There is no freeze observed with single GPUs.

There are no known workarounds at this time.
2.1.5.6 Issue with Running AMD ROCm v3.3 User Mode with AMD ROCm v3.5 DKMS Kernel Module

Running AMD ROCm v3.3 in the user mode with the AMD ROCm v3.5 DKMS kernel module will cause the following features to be broken:

- IPC import/export, cross memory copy (used by UCX and MPI)
- Experimental GDB support

**Resolution:** Install ROCm v3.5 Thunk (*hsakmt*) when using ROCm 3.5 Kernel Fusion Driver (KFD).

2.1.6 Deprecations

2.1.6.1 Heterogeneous Compute Compiler

In this release, the Heterogeneous Compute Compiler (HCC) compiler is deprecated and the HIP-Clang compiler is introduced for compiling Heterogeneous-Compute Interface for Portability (HIP) programs.

For more information, see HIP documentation at: https://rocmdocs.amd.com/en/latest/Programming_Guides/Programming-Guides.html

2.1.7 Deploying ROCm

AMD hosts both Debian and RPM repositories for the ROCm v3.5.x packages.

For more information on ROCM installation on all platforms, see


2.1.8 DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale. S AMD, the AMD Arrow logo, Radeon, Ryzen, Epyc, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Google® is a registered trademark of Google LLC. PCIe® is a registered trademark of PCI-SIG Corporation. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. Ubuntu and the Ubuntu logo are registered trademarks of Canonical Ltd. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
2.2 AMD ROCm Installation Guide v3.5.0

- Deploying ROCm
  - Ubuntu
  - `CentOS v7.7/RHEL v7.8 and CentOS/RHEL 8.1`^n
  - SLES 15 Service Pack 1
- AMD ROCm MultiVersion Installation
- ROCm Installation Known Issues and Workarounds
- Getting the ROCm Source Code

2.2.1 Deploying ROCm

AMD hosts both Debian and RPM repositories for the ROCm v3.x packages.

The following directions show how to install ROCm on supported Debian-based systems such as Ubuntu 18.04.x

**Note:** These directions may not work as written on unsupported Debian-based distributions. For example, newer versions of Ubuntu may not be compatible with the rock-dkms kernel driver. In this case, you can exclude the rocm-dkms and rock-dkms packages.

2.2.1.1 Ubuntu

2.2.1.1.1 Installing a ROCm Package from a Debian Repository

To install from a Debian Repository:

1. Run the following code to ensure that your system is up to date:

   ```bash
   sudo apt update
   sudo apt dist-upgrade
   sudo apt install libnuma-dev
   sudo reboot
   ```

2. Add the ROCm apt repository.

   For Debian-based systems like Ubuntu, configure the Debian ROCm repository as follows:

   ```bash
   wget -q -O - http://repo.radeon.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
   echo 'deb [arch=amd64] http://repo.radeon.com/rocm/apt/debian/ xenial main' | sudo tee /etc/apt/sources.list.d/rocm.list
   ```

   The gpg key may change; ensure it is updated when installing a new release. If the key signature verification fails while updating, re-add the key from the ROCm apt repository.

   The current rocm.gpg.key is not available in a standard key ring distribution, but has the following sha1sum hash:

   ```plaintext
   e85a40d1a43453fe37d63aa6899bc96e08f2817a rocm.gpg.key
   ```

3. Install the ROCm meta-package. Update the appropriate repository list and install the rocm-dkms meta-package:
sudo apt update
sudo apt install rocm-dkms

4. Set permissions. To access the GPU, you must be a user in the video group. Ensure your user account is a member of the video group prior to using ROCm. To identify the groups you are a member of, use the following command:

```
groups
```

5. To add your user to the video group, use the following command with the sudo password:

```
sudo usermod -a -G video $LOGNAME
```

6. By default, you must add any future users to the video group. To add future users to the video group, run the following command:

```
echo 'ADD_EXTRA_GROUPS=1' | sudo tee -a /etc/adduser.conf
echo 'EXTRA_GROUPS=video' | sudo tee -a /etc/adduser.conf
```

7. Restart the system.

8. After restarting the system, run the following commands to verify that the ROCm installation is successful. If you see your GPUs listed by both commands, the installation is considered successful.

```
/opt/rocm/bin/rocminfo
/opt/rocm/opencl/bin/x86_64/clinfo
```

Note: To run the ROCm programs, add the ROCm binaries in your PATH.

```
echo 'export PATH=$PATH:/opt/rocm/bin:/opt/rocm/profiler/bin:/opt/rocm/opencl/bin/x86_64' | sudo tee -a /etc/profile.d/rocm.sh
```

### 2.2.1.1.2 Uninstalling ROCm Packages from Ubuntu

To uninstall the ROCm packages from Ubuntu 16.04.6 or Ubuntu 18.04.4, run the following command:

```
sudo apt autoremove rocm-opencl rocm-dkms rocm-dev rocm-utils
```

### 2.2.1.1.3 Installing Development Packages for Cross Compilation

It is recommended that you develop and test development packages on different systems. For example, some development or build systems may not have an AMD GPU installed. In this scenario, you must avoid installing the ROCk kernel driver on the development system.

Instead, install the following development subset of packages:

```
sudo apt update
sudo apt install rocm-dev
```

Note: To execute ROCm enabled applications, you must install the full ROCm driver stack on your system.
2.2.1.1.4 Using Debian-based ROCm with Upstream Kernel Drivers

You can install the ROCm user-level software without installing the AMD’s custom ROck kernel driver. To use the upstream kernels, run the following commands instead of installing rocm-dkms:

```bash
sudo apt update
sudo apt install rocm-dev
echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"'
sudo tee /etc/udev/rules.d/70-kfd.rules
```

2.2.1.2 CentOS v7.7/RHEL v7.8 and CentOS/RHEL v8.1

This section describes how to install ROCm on supported RPM-based systems such as CentOS v7.7/RHEL v7.8 and CentOS/RHEL v8.1.

2.2.1.2.1 Preparing RHEL for Installation

RHEL is a subscription-based operating system. You must enable the external repositories to install on the devtoolset-7 environment and the dkms support files.

Note: The following steps do not apply to the CentOS installation.

1. The subscription for RHEL must be enabled and attached to a pool ID. See the Obtaining an RHEL image and license page for instructions on registering your system with the RHEL subscription server and attaching to a pool id.

2. Enable the following repositories for RHEL v7.x:

   ```bash
   sudo subscription-manager repos --enable rhel-server-rhscl-7-rpms
   sudo subscription-manager repos --enable rhel-7-server-optional-rpms
   sudo subscription-manager repos --enable rhel-7-server-extras-rpms
   ```

3. Enable additional repositories by downloading and installing the epel-release-latest-7/epel-release-latest-8 repository RPM:

   ```bash
   sudo rpm -ivh <repo>
   ```

   For more details, see https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm for RHEL v7.x
   see https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm for RHEL v8.x

4. Install and set up Devtoolset-7.

   **Note:** Devtoolset is not required for CentOS/RHEL v8.x

To setup the Devtoolset-7 environment, follow the instructions on this page: https://www.softwarecollections.org/en/scls/rhscl/devtoolset-7/

Note: devtoolset-7 is a software collections package and is not supported by AMD.
2.2.1.3 Installing CentOS v7.7/v8.1 for DKMS

Use the dkms tool to install the kernel drivers on CentOS/RHEL:

```
sudo yum install -y epel-release
sudo yum install -y dkms kernel-headers-`uname -r` kernel-devel-`uname -r`
```

2.2.1.3.1 Installing ROCm

To install ROCm on your system, follow the instructions below:

1. Delete the previous versions of ROCm before installing the latest version.

2. Create a `/etc/yum.repos.d/rocm.repo` file with the following contents:

   ```
   [ROCm]
   name=ROCm
   baseurl=http://repo.radeon.com/rocm/yum/rpm
   enabled=1
   gpgcheck=0
   
   CentOS/RHEL 7.x : baseurl=http://repo.radeon.com/rocm/yum/rpm
   CentOS/RHEL 8.x : baseurl=http://repo.radeon.com/rocm/centos8/rpm
   
   Note: The URL of the repository must point to the location of the repositories’ repodata database.
   
3. Install ROCm components using the following command:

   ```
   sudo yum install rocm-dkms
   
   Note: This step is applicable only for CentOS/RHEL v8.1 and is not required for v7.8.
   ```

4. Restart the system. The rock-dkms component is installed and the /dev/kfd device is now available.

5. Set permissions. To access the GPU, you must be a user in the video group. Ensure your user account is a member of the video group prior to using ROCm. To identify the groups you are a member of, use the following command:

   ```
   groups
   ```

6. To add your user to the video group, use the following command with the sudo password:

   ```
   sudo usermod -a -G video $LOGNAME
   ```

7. By default, add any future users to the video group. Run the following command to add users to the video group:

   ```
   echo 'ADD_EXTRA_GROUPS=1' | sudo tee -a /etc/adduser.conf
   echo 'EXTRA_GROUPS=video' | sudo tee -a /etc/adduser.conf
   ```

   Note: Before updating to the latest version of the operating system, delete the ROCm packages to avoid DKMS-related issues.

8. Restart the system.

9. Test the ROCm installation.
### 2.2.1.3.2 Testing the ROCm Installation

After restarting the system, run the following commands to verify that the ROCm installation is successful. If you see your GPUs listed, you are good to go!

```
/opt/rocm/bin/rocminfo
/opt/rocm/opencl/bin/x86_64/clinfo
```

**Note:** Add the ROCm binaries in your PATH for easy implementation of the ROCm programs.

```
echo 'export PATH=$PATH:/opt/rocm/bin:/opt/rocm/profiler/bin:/opt/rocm/opencl/bin/x86_64' |
sudo tee -a /etc/profile.d/rocm.sh
```

### 2.2.1.3.3 Compiling Applications Using HCC, HIP, and Other ROCm Software

To compile applications or samples, run the following command to use gcc-7.2 provided by the devtoolset-7 environment:

```
scl enable devtoolset-7 bash
```

### 2.2.1.3.4 Uninstalling ROCm from CentOS/RHEL

To uninstall the ROCm packages, run the following command:

```
sudo yum autoremove rocm-opencl rocm-dkms rock-dkms
```

### 2.2.1.3.5 Installing Development Packages for Cross Compilation

You can develop and test ROCm packages on different systems. For example, some development or build systems may not have an AMD GPU installed. In this scenario, you can avoid installing the ROCm kernel driver on your development system. Instead, install the following development subset of packages:

```
sudo yum install rocm-dev
```

**Note:** To execute ROCm-enabled applications, you will require a system installed with the full ROCm driver stack.

### 2.2.1.3.6 Using ROCm with Upstream Kernel Drivers

You can install ROCm user-level software without installing AMD’s custom ROCk kernel driver. To use the upstream kernel drivers, run the following commands

```
sudo yum install rocm-dev
echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"'
sudo tee /etc/udev/rules.d/70-kfd.rules
```

**Note:** You can use this command instead of installing rocm-dkms.
2.2.1.4 SLES 15 Service Pack 1

The following section tells you how to perform an install and uninstall ROCm on SLES 15 SP 1.

**Installation**

1. Install the “dkms” package.
   
   ```
   sudo SUSEConnect --product PackageHub/15.1/x86_64
   sudo zypper install dkms
   ```

2. Add the ROCm repo.
   
   ```
   sudo zypper clean -all
   sudo zypper addrepo --no-gpgcheck http://repo.radeon.com/rocm/zyp/zypper/ rocm
   sudo zypper ref
   zypper install rocm-dkms
   sudo zypper install rocm-dkms
   sudo reboot
   ```

3. Run the following command once
   
   ```
   cat <<EOF | sudo tee /etc/modprobe.d/10-unsupported-modules.conf
   allow_unsupported_modules 1
   EOF
   sudo modprobe amdgpu
   ```

4. Verify the ROCm installation.

5. Run `/opt/rocm/bin/rocminfo` and `/opt/rocm/opencl/bin/x86_64/clinfo` commands to list the GPUs and verify that the ROCm installation is successful.

6. Set permissions.

   To access the GPU, you must be a user in the video group. Ensure your user account is a member of the video group prior to using ROCm. To identify the groups you are a member of, use the following command:

   ```
   groups
   ```

7. To add your user to the video group, use the following command with the sudo password:
   
   ```
   sudo usermod -a -G video $LOGNAME
   ```

8. By default, add any future users to the video group. Run the following command to add users to the video group:
   
   ```
   echo 'ADD_EXTRA_GROUPS=1' | sudo tee -a /etc/adduser.conf
   echo 'EXTRA_GROUPS=video' | sudo tee -a /etc/adduser.conf
   ```

9. Restart the system.

10. Test the basic ROCm installation.

11. After restarting the system, run the following commands to verify that the ROCm installation is successful. If you see your GPUs listed by both commands, the installation is considered successful.
   
   ```
   /opt/rocm/bin/rocminfo
   /opt/rocm/opencl/bin/x86_64/clinfo
   ```

Note: To run the ROCm programs more efficiently, add the ROCm binaries in your PATH.
echo 'export PATH=$PATH:/opt/rocm/bin:/opt/rocm/profiler/bin:/opt/rocm/opencl/bin/x86_64' | sudo tee -a /etc/profile.d/rocm.sh

Uninstallation

To uninstall, use the following command:

```
sudo zypper remove rocm-opencl rocm-dkms rock-dkms
```

Note: Ensure all other installed packages/components are removed. Note: Ensure all the content in the /opt/rocm directory is completely removed. If the command does not remove all the ROCm components/packages, ensure you remove them individually.

### 2.2.1.4.1 Performing an OpenCL-only Installation of ROCm

Some users may want to install a subset of the full ROCm installation. If you are trying to install on a system with a limited amount of storage space, or which will only run a small collection of known applications, you may want to install only the packages that are required to run OpenCL applications. To do that, you can run the following installation command instead of the command to install rocm-dkms.

```
sudo yum install rock-dkms rocm-opencl-devel
```

### 2.2.2 AMD ROCm MultiVersion Installation

Users can install and access multiple versions of the ROCm toolkit simultaneously.

Previously, users could install only a single version of the ROCm toolkit.

Now, users have the option to install multiple versions simultaneously and toggle to the desired version of the ROCm toolkit. From the v3.3 release, multiple versions of ROCm packages can be installed in the `/opt/rocm-<version>` folder.

Ensure the existing installations of ROCm, including `/opt/rocm`, are completely removed before the v3.3 ROCm toolkit installation. The ROCm v3.3 package requires a clean installation.

- To install a single instance of ROCm, use the rocm-dkms or rocm-dev packages to install all the required components. This creates a symbolic link `/opt/rocm` pointing to the corresponding version of ROCm installed on the system.
- To install individual ROCm components, create the `/opt/rocm` symbolic link pointing to the version of ROCm installed on the system. For example, `# ln -s /opt/rocm-3.3.0 /opt/rocm`
- To install multiple instance ROCm packages, create `/opt/rocm` symbolic link pointing to the version of ROCm installed/used on the system. For example, `# ln -s /opt/rocm-3.3.0 /opt/rocm`

Note: The Kernel Fusion Driver (KFD) must be compatible with all versions of the ROCm software installed on the system.

Review the following important notes:

#### Single Version Installation

To install a single instance of the ROCm package, access the non-versioned packages. You must not install any components from the multi-instance set.

For example,

- rocm-dkms
- rocm-dev
• hip

A fresh installation or an upgrade of the single-version installation will remove the existing version completely and install the new version in the /opt/rocm-<version> folder.

**ROCm 3.0**

Multi Version Installation

- To install a multi-instance of the ROCm package, access the versioned packages and components.

For example,

- rocm-dkms3.3.0
- rocm-dev3.3.0
- hip3.3.0

- The new multi-instance package enables you to install two versions of the ROCm toolkit simultaneously and provides the ability to toggle between the two versioned packages.

- The ROCm-DEV package does not create symlinks

- Users must create symlinks if required

- Multi-version installation with previous ROCm versions is not supported

- Kernel Fusion Driver (KFD) must be compatible with all versions of ROCm installations
IMPORTANT: A single instance ROCm package cannot co-exist with the multi-instance package.

NOTE: The multi-instance installation applies only to ROCm v3.3 and above. This package requires a fresh installation after the complete removal of existing ROCm packages. The multi-version installation is not backward compatible.

Note: If you install the multi-instance version of AMD ROCm and create a sym-link to `/opt/rocm`, you must run `Idconfig` to ensure the software stack functions correctly with the sym-link.

### 2.2.3 ROCm Installation Known Issues and Workarounds

The ROCm platform relies on some closed source components to provide functionalities like HSA image support. These components are only available through the ROCm repositories, and they may be deprecated or become open source components in the future. These components are made available in the following packages:

- hsa-ext-rocr-dev
2.2.4 Getting the ROCm Source Code

AMD ROCm is built from open source software. It is, therefore, possible to modify the various components of ROCm by downloading the source code and rebuilding the components. The source code for ROCm components can be cloned from each of the GitHub repositories using git. For easy access to download the correct versions of each of these tools, the ROCm repository contains a repo manifest file called default.xml. You can use this manifest file to download the source code for ROCm software.

2.2.4.1 Installing the Repo

The repo tool from Google® allows you to manage multiple git repositories simultaneously. Run the following commands to install the repo:

```
mkdir -p ~/bin/
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

Note: You can choose a different folder to install the repo into if you desire. ~/bin/ is used as an example.

2.2.5 Downloading the ROCm Source Code

The following example shows how to use the repo binary to download the ROCm source code. If you choose a directory other than ~/bin/ to install the repo, you must use that chosen directory in the code as shown below:

```
mkdir -p ~/ROCm/
cd ~/ROCm/
~/bin/repo init -u https://github.com/RadeonOpenCompute/ROCm.git -b roc-3.5.0
repo sync
```

Note: Using this sample code will cause the repo to download the open source code associated with this ROCm release. Ensure that you have ssh-keys configured on your machine for your GitHub ID prior to the download.

2.2.6 Building the ROCm Source Code

Each ROCm component repository contains directions for building that component. You can access the desired component for instructions to build the repository.

2.3 Machine Learning and High Performance Computing Software Stack for AMD GPU v3.5.0

2.3.1 Machine Learning and High Performance Computing Software Stack for AMD GPU v3.5.0

ROCm Version 3.5.0
2.3.1.1 ROCm Binary Package Structure

ROCm is a collection of software ranging from drivers and runtimes to libraries and developer tools. In AMD’s package distributions, these software projects are provided as a separate packages. This allows users to install only the packages they need, if they do not wish to install all of ROCm. These packages will install most of the ROCm software into /opt/rocm/ by default.

The packages for each of the major ROCm components are:

- ROCK Kernel Driver: rock-dkms
- ROCr Runtime: hsa-rocr-dev, hsa-ext-rocr-dev
- ROCt Thunk Interface: hsakmt-roct, hsakmt-roct-dev
- ROCm SMI: rocm-smi
- ROCm cmake: rocm-cmake
- rocminfo: rocminfo
- ROCm Bandwidth Test: rocm_bandwidth_test
- HCC compiler: hcc (in deprecation)
- HIP: hip_base, hip_doc, hip_hcc, hip_samples
- ROCM Clang-OCL Kernel Compiler: rocm-clang-ocl
  - ROCm Device Libraries: rocm-device-libs
  - ROCm OpenCL: rocm-opencl, rocm-opencl-devel (on RHEL/CentOS), rocm-opencl-dev (on Ubuntu)

ROCm Development ToolChain

- Asynchronous Task and Memory Interface (ATMI): atmi
- ROCm Debug Agent: rocm_debug_agent
- ROCm Code Object Manager: comgr
- ROC Profiler: rocprofiler-dev
- ROC Tracer: roctracer-dev
- Radeon Compute Profiler: rocm-profiler
- rocALUTION: rocalution
- rocBLAS: rocblas
- hipBLAS: hipblas
- hipCUB: hipCUB
- rocFFT: roccfft
- rocRAND: rocrand
- rocSPARSE: rocsparse
- hipSPARSE: hipsparse
- ROCm SMI Lib: rocm-smi-lib64
- rocThrust: rocThrust
• **MIOpen**: MIOpen-HIP (for the HIP version), MIOpen-OpenCL (for the OpenCL version)
• **MIOpenGEMM**: miopengemm
• **MIVisionX**: mivisionx
• **RCCL**: rccl

To make it easier to install ROCm, the AMD binary repositories provide a number of meta-packages that will automatically install multiple other packages. For example, `rocm-dkms` is the primary meta-package that is used to install most of the base technology needed for ROCm to operate. It will install the `rock-dkms` kernel driver, and another meta-package

| (rocm-dev) which installs most of the user-land ROCm core components, support software, and development tools.

The `rocm-utils` meta-package will install useful utilities that, while not required for ROCm to operate, may still be beneficial to have. Finally, the `rocm-libs` meta-package will install some (but not all) of the libraries that are part of ROCm.

The chain of software installed by these meta-packages is illustrated below:

```
rocm-dkms
|-- rock-dkms
 \-- rocm-dev
    |-- comgr
    |-- hcc
    |-- hip-base
    |-- hip-doc
    |-- hip-hcc
    |-- hip-samples
    |-- hsakmt-roct
    |-- hsakmt-roct-dev
    |-- hsa-amd-aqlprofile
    |-- hsa-ext-rocr-dev
    |-- hsa-rocr-dev
    |-- rocm-cmake
    |-- rocm-device-libs
    |-- rocm-smi-lib64
    |-- rocprofiler-dev
    |-- rocm-debug-agent
 \-- rocm-utils
    |-- rocminfo
    \-- rocm-clang-ocl # This will cause OpenCL to be installed

rocm-libs
 |-- hipblas
 |-- hipcub
 |-- hipsparse
 |-- rocalution
 |-- rocblas
 |-- rocfft
 |-- rocprim
 |-- rocrand
 |-- rocsparse
 \-- rocthrust
```

These meta-packages are not required but may be useful to make it easier to install ROCm on most systems.

Note: Some users may want to skip certain packages. For instance, a user that wants to use the upstream kernel drivers (rather than those supplied by AMD) may want to skip the rocm-dkms and rock-dkms packages. Instead, they could
directly install rocm-dev.

Similarly, a user that only wants to install OpenCL support instead of HCC and HIP may want to skip the rocm-dkms and rocm-dev packages. Instead, they could directly install rock-dkms, rocm-opencl, and rocm-opencl-dev and their dependencies.

### 2.3.1.2 ROCm Platform Packages

The following platform packages are for ROCm v3.3.0:

**Drivers, ToolChains, Libraries, and Source Code**

The latest supported version of the drivers, tools, libraries and source code for the ROCm platform have been released and are available from the following GitHub repositories:

**ROCm Core Components**

- ROCk Kernel Driver
- ROCr Runtime
- ROCt Thunk Interface

**ROCm Support Software**

- ROCm SMI
- ROCm cmake
- rocminfo
- ROCm Bandwidth Test

**ROCm Compilers**

- HCC compiler (in deprecation)
- HIP
  - ROCM Clang-OCL Kernel Compiler

**Example Applications:**

- HCC Examples (in deprecation)
- HIP Examples

**ROCm Device Libraries and Tools**

- ROCm Device Libraries
- ROCm OpenCL Runtime
- ROCm LLVM OCL
- ROCm Device Libraries OCL
- Asynchronous Task and Memory Interface
- ROCr Debug Agent
- ROCm Code Object Manager
- ROC Profiler
- ROC Tracer
- AOMP
• Radeon Compute Profiler
• ROCm Validation Suite

ROCm Libraries
• rocBLAS
• hipBLAS
• rocFFT
• rocRAND
• rocSPARSE
• hipSPARSE
• rocALUTION
• MIOpenGEMM
• mi open
• rocThrust
• ROCm SMI Lib
• RCCL
• MIVisionX
• hipCUB
• AMDMIGraphX

Note: HCC Compiler is in deprecation mode.
Features and enhancements introduced in previous versions of ROCm can be found in Current-Release-Notes.

2.4 AMD ROCm Version History

This file contains archived version history information for the ROCm project.

2.4.1 New features and enhancements in ROCm v3.3

Release Notes: https://github.com/RadeonOpenCompute/ROCm/tree/roc-3.3.0

Multi-Version Installation Users can install and access multiple versions of the ROCm toolkit simultaneously. Previously, users could install only a single version of the ROCm toolkit.

GPU Process Information A new functionality to display process information for GPUs is available in this release. For example, you can view the process details to determine if the GPU(s) must be reset.

Support for 3D Pooling Layers AMD ROCm is enhanced to include support for 3D pooling layers. The implementation of 3D pooling layers now allows users to run 3D convolutional networks, such as ResNext3D, on AMD Radeon Instinct GPUs.

ONNX Enhancements Open Neural Network eXchange (ONNX) is a widely-used neural net exchange format. The AMD model compiler & optimizer support the pre-trained models in ONNX, NNEF, & Caffe formats. Currently, ONNX versions 1.3 and below are supported.
2.4.2 New features and enhancements in ROCm v3.2

This release was not productized.

2.4.3 New features and enhancements in ROCm v3.1

‘Release Notes: https://github.com/RadeonOpenCompute/ROCm/tree/roc-3.1.0

Change in ROCm Installation Directory Structure

A fresh installation of the ROCm toolkit installs the packages in the /opt/rocm-<version> folder. Previously, ROCm toolkit packages were installed in the /opt/rocm folder.

Reliability, Accessibility, and Serviceability Support for Vega 7nm

The Reliability, Accessibility, and Serviceability (RAS) support for Vega7nm is now available.

SLURM Support for AMD GPU

SLURM (Simple Linux Utility for Resource Management) is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters.

2.4.4 New features and enhancements in ROCm v3.0

Release Notes: https://github.com/RadeonOpenCompute/ROCm/tree/roc-3.0.0

• Support for CentOS RHEL v7.7
• Support is extended for CentOS/RHEL v7.7 in the ROCm v3.0 release. For more information about the CentOS/RHEL v7.7 release, see:
• CentOS/RHEL
• Initial distribution of AOMP 0.7-5 in ROCm v3.0

The code base for this release of AOMP is the Clang/LLVM 9.0 sources as of October 8th, 2019. The LLVM-project branch used to build this release is AOMP-191008. It is now locked. With this release, an artifact tarball of the entire source tree is created. This tree includes a Makefile in the root directory used to build AOMP from the release tarball. You can use Spack to build AOMP from this source tarball or build manually without Spack.

• Fast Fourier Transform Updates

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform. Fast Fourier transforms are used in signal processing, image processing, and many other areas. The following real FFT performance change is made in the ROCm v3.0 release:

• Implement efficient real/complex 2D transforms for even lengths.

Other improvements:

• More 2D test coverage sizes.
• Fix buffer allocation error for large 1D transforms.
• C++ compatibility improvements.

MemCopy Enhancement for rocProf In the v3.0 release, the rocProf tool is enhanced with an additional capability to dump asynchronous GPU memcpy information into a .csv file. You can use the ‘-hsa-trace’ option to create the results_mcopy.csv file. Future enhancements will include column labels.
2.4.5 New features and enhancements in ROCm v2.10

rocBLAS Support for Complex GEMM

The rocBLAS library is a gpu-accelerated implementation of the standard Basic Linear Algebra Subroutines (BLAS). rocBLAS is designed to enable you to develop algorithms, including high performance computing, image analysis, and machine learning.

In the AMD ROCm release v2.10, support is extended to the General Matrix Multiply (GEMM) routine for multiple small matrices processed simultaneously for rocBLAS in AMD Radeon Instinct MI50. Both single and double precision, CGEMM and ZGEMM, are now supported in rocBLAS.

Support for SLES 15 SP1

In the AMD ROCm v2.10 release, support is added for SUSE Linux® Enterprise Server (SLES) 15 SP1. SLES is a modular operating system for both multimodal and traditional IT.

Code Marker Support for rocProfiler and rocTracer Libraries

Code markers provide the external correlation ID for the calling thread. This function indicates that the calling thread is entering and leaving an external API region.

2.4.6 New features and enhancements in ROCm 2.9

Initial release for Radeon Augmentation Library(RALI)

The AMD Radeon Augmentation Library (RALI) is designed to efficiently decode and process images from a variety of storage formats and modify them through a processing graph programmable by the user. RALI currently provides C API.

Quantization in MIGraphX v0.4

MIGraphX 0.4 introduces support for fp16 and int8 quantization. For additional details, as well as other new MIGraphX features, see MIGraphX documentation.

rocSparse csrgemm

csgemm enables the user to perform matrix-matrix multiplication with two sparse matrices in CSR format.

Singularity Support

ROCm 2.9 adds support for Singularity container version 2.5.2.

Initial release of rocTX

ROCm 2.9 introduces rocTX, which provides a C API for code markup for performance profiling. This initial release of rocTX supports annotation of code ranges and ASCII markers.

- Added support for Ubuntu 18.04.3
- Ubuntu 18.04.3 is now supported in ROCm 2.9.
2.4.7 New features and enhancements in ROCm 2.8

Support for NCCL 2.4.8 API
Implements ncclCommAbort() and ncclCommGetAsyncError() to match the NCCL 2.4.x API

2.4.8 New features and enhancements in ROCm 2.7.2

This release is a hotfix for ROCm release 2.7.

2.4.9 Issues fixed in ROCm 2.7.2

• A defect in upgrades from older ROCm releases has been fixed.
• rocprofiler –hiptrace and –hsatrace fails to load roctracer library
• In ROCm 2.7.2, rocprofiler –hiptrace and –hsatrace fails to load roctracer library defect has been fixed.
• To generate traces, please provide directory path also using the parameter: -d <$directoryPath> for example:
  /opt/rocm/bin/rocprof –hsa-trace -d $PWD/traces /opt/rocm/hip/samples/0_Intro/bit_extract/bit_extract All traces and results will be saved under $PWD/traces path

2.4.10 Upgrading from ROCm 2.7 to 2.7.2

To upgrade, please remove 2.7 completely as specified for ubuntu or for centos/rhel, and install 2.7.2 as per instructions

Other notes To use rocprofiler features, the following steps need to be completed before using rocprofiler:

Step-1: Install roctracer Ubuntu 16.04 or Ubuntu 18.04: sudo apt install roctracer-dev CentOS/RHEL 7.6: sudo yum install roctracer-dev
Step-2: Add /opt/rocm/roctracer/lib to LD_LIBRARY_PATH New features and enhancements in ROCm 2.7 [rocFFT]
Real FFT Functional Improved real/complex 1D even-length transforms of unit stride. Performance improvements of up to 4.5x are observed. Large problem sizes should see approximately 2x.
rocRand Enhancements and Optimizations
Added support for new datatypes: uchar, ushort, half.
Improved performance on “Vega 7nm” chips, such as on the Radeon Instinct MI50
mtgp32 uniform double performance changes due generation algorithm standardization. Better quality random numbers now generated with 30% decrease in performance
Up to 5% performance improvements for other algorithms
RAS
Added support for RAS on Radeon Instinct MI50, including:
  • Memory error detection
  • Memory error detection counter
  • ROCm-SMI enhancements
  • Added ROCm-SMI CLI and LIB support for FW version, compute running processes, utilization rates, utilization counter, link error counter, and unique ID.
2.4.11 New features and enhancements in ROCm 2.6

ROCmInfo enhancements

ROCmInfo was extended to do the following: For ROCr API call errors including initialization determine if the error could be explained by:

- ROck (driver) is not loaded / available
- User does not have membership in appropriate group - “video”
- If not above print the error string that is mapped to the returned error code
- If no error string is available, print the error code in hex

Thrust - Functional Support on Vega20

ROCm2.6 contains the first official release of rocThrust and hipCUB. rocThrust is a port of thrust, a parallel algorithm library. hipCUB is a port of CUB, a reusable software component library. Thrust/CUB has been ported to the HIP/ROCM platform to use the rocPRIM library. The HIP ported library works on HIP/ROCM platforms.

Note: rocThrust and hipCUB library replaces https://github.com/ROCmSoftwarePlatform/thrust (hip-thrust), i.e. hip-thrust has been separated into two libraries, rocThrust and hipCUB. Existing hip-thrust users are encouraged to port their code to rocThrust and/or hipCUB. Hip-thrust will be removed from official distribution later this year.

MIGraphX v0.3

MIGraphX optimizer adds support to read models frozen from Tensorflow framework. Further details and an example usage at https://github.com/ROCmSoftwarePlatform/AMDMIGraphX/wiki/Getting-started:-using-the-new-features-of-MIGraphX-0.3

MIOpen 2.0

This release contains several new features including an immediate mode for selecting convolutions, bfloat16 support, new layers, modes, and algorithms.

MIOpenDriver, a tool for benchmarking and developing kernels is now shipped with MIOpen. BFloat16 now supported in HIP requires an updated rocBLAS as a GEMM backend.

Immediate mode API now provides the ability to quickly obtain a convolution kernel.

MIOpen now contains HIP source kernels and implements the ImplicitGEMM kernels. This is a new feature and is currently disabled by default. Use the environmental variable “MIOPEN_DEBUG_CONV_IMPLICIT_GEMM=1” to activation this feature. ImplicitGEMM requires an up to date HIP version of at least 1.5.9211.

A new “loss” catagory of layers has been added, of which, CTC loss is the first. See the API reference for more details. 2.0 is the last release of active support for gfx803 architectures. In future releases, MIOpen will not actively debug and develop new features specifically for gfx803.

System Find-Db in memory cache is disabled by default. Please see build instructions to enable this feature. Additional documentation can be found here: https://rocmsoftwareplatform.github.io/MIOpen/doc/html/

Bfloat16 software support in rocBLAS/Tensile

Added mixed precision bfloat16/IEEE f32 to gemm_ex. The input and output matrices are bfloat16. All arithmetic is in IEEE f32.

AMD Infinity Fabric™ Link enablement

The ability to connect four Radeon Instinct MI60 or Radeon Instinct MI50 boards in two hives or two Radeon Instinct MI60 or Radeon Instinct MI50 boards in four hives via AMD Infinity Fabric™ Link GPU interconnect technology has been added.

ROCm-smi features and bug fixes

mGPU & Vendor check

Fix clock printout if DPM is disabled

Fix finding marketing info on CentOS
Clarify some error messages
ROCM-smi-lib enhancements
Documentation updates
Improvements to *name_get functions
RCCL2 Enablement
RCCL2 supports collectives intranode communication using PCIe, Infinity Fabric™, and pinned host memory, as well as internode communication using Ethernet (TCP/IP sockets) and Infiniband/RoCE (Infiniband Verbs). Note: For Infiniband/RoCE, RDMA is not currently supported.
rocFFT enhancements

2.4.12 New features and enhancements in ROCm 2.5

UCX 1.6 support
Support for UCX version 1.6 has been added.
BFloat16 GEMM in rocBLAS/Tensile
Software support for BFloat16 on Radeon Instinct MI50, MI60 has been added. This includes:
Mixed precision GEMM with BFloat16 input and output matrices, and all arithmetic in IEEE32 bit
Input matrix values are converted from BFloat16 to IEEE32 bit, all arithmetic and accumulation is IEEE32 bit. Output values are rounded from IEEE32 bit to BFloat16
Accuracy should be correct to 0.5 ULP
ROCM-SMI enhancements

CLI support for querying the memory size, driver version, and firmware version has been added to ROCm-smi.

[PyTorch] multi-GPU functional support (CPU aggregation/Data Parallel)
Multi-GPU support is enabled in PyTorch using Dataparallel path for versions of PyTorch built using the 06c8aa7a3bbd91cda2fd6255ec82aad21af1c0d5 commit or later.
rocSparse optimization on Radeon Instinct MI50 and MI60
This release includes performance optimizations for csrsv routines in the rocSparse library.

[Thrust] Preview
Preview release for early adopters. rocThrust is a port of thrust, a parallel algorithm library. Thrust has been ported to the HIP/ROCM platform to use the rocPRIM library. The HIP ported library works on HIP/ROCM platforms.
Note: This library will replace https://github.com/ROCmSoftwarePlatform/thrust in a future release. The package for rocThrust (this library) currently conflicts with version 2.5 package of thrust. They should not be installed together.
Support overlapping kernel execution in same HIP stream
HIP API has been enhanced to allow independent kernels to run in parallel on the same stream.

AMD Infinity Fabric™ Link enablement
The ability to connect four Radeon Instinct MI60 or Radeon Instinct MI50 boards in one hive via AMD Infinity Fabric™ Link GPU interconnect technology has been added.
2.4.13 New features and enhancements in ROCm 2.4

TensorFlow 2.0 support

ROCm 2.4 includes the enhanced compilation toolchain and a set of bug fixes to support TensorFlow 2.0 features natively.

AMD Infinity Fabric™ Link enablement

ROCm 2.4 adds support to connect two Radeon Instinct MI60 or Radeon Instinct MI50 boards via AMD Infinity Fabric™ Link GPU interconnect technology.

2.4.14 New features and enhancements in ROCm 2.3

Mem usage per GPU

Per GPU memory usage is added to rocm-smi. Display information regarding used/total bytes for VRAM, visible VRAM and GTT, via the --showmeminfo flag.

MIVisionX, v1.1 - ONNX

ONNX parser changes to adjust to new file formats.

MIGraphX, v0.2

MIGraphX 0.2 supports the following new features:

New Python API

- Support for additional ONNX operators and fixes that now enable a large set of Imagenet models.
- Support for RNN Operators.
- Support for multi-stream Execution.
- [Experimental] Support for Tensorflow frozen protobuf files.

See: Getting-started:-using-the-new-features-of-MIGraphX-0.2 for more details.

MIOpen, v1.8 - 3d convolutions and int8

This release contains full 3-D convolution support and int8 support for inference. Additionally, there are major updates in the performance database for major models including those found in Torchvision. See: MIOpen releases.

Caffe2 - mGPU support

Multi-gpu support is enabled for Caffe2.

rocTracer library, ROCm tracing API for collecting runtimes API and asynchronous GPU activity traces HIP/HCC domains support is introduced in rocTracer library.

BLAS - Int8 GEMM performance, Int8 functional and performance Introduces support and performance optimizations for Int8 GEMM, implements TRSV support, and includes improvements and optimizations with Tensile.

Prioritized L1/L2/L3 BLAS (functional) Functional implementation of BLAS L1/L2/L3 functions.

BLAS - tensile optimization Improvements and optimizations with tensile.

MIOpen Int8 support Support for int8.
2.4.15 New features and enhancements in ROCm 2.2

rocSparse Optimization on Vega20 Cache usage optimizations for csrsv (sparse triangular solve), coomv (SpMV in COO format) and ellmv (SpMV in ELL format) are available.

DGEMM and DTRSM Optimization 
Improved DGEMM performance for reduced matrix sizes (k=384, k=256)

Caffe2 Added support for multi-GPU training

2.4.16 New features and enhancements in ROCm 2.1

RocTracer v1.0 preview release – `rocprof` HSA runtime tracing and statistics support - Supports HSA API tracing and HSA asynchronous GPU activity including kernels execution and memory copy

Improvements to ROCM-SMI tool - Added support to show real-time PCIe bandwidth usage via the -b/–showbw flag

DGEMM Optimizations - Improved DGEMM performance for large square and reduced matrix sizes (k=384, k=256)

2.4.17 New features and enhancements in ROCm 2.0

Adds support for RHEL 7.6 / CentOS 7.6 and Ubuntu 18.04.1

Adds support for Vega 7nm, Polaris 12 GPUs

Introduces MIVisionX A comprehensive computer vision and machine intelligence libraries, utilities and applications bundled into a single toolkit. Improvements to ROCm Libraries rocSPARSE & hipSPARSE rocBLAS with improved DGEMM efficiency on Vega 7nm

MIOpen This release contains general bug fixes and an updated performance database Group convolutions backwards weights performance has been improved

RNNs now support fp16 Tensorflow multi-gpu and Tensorflow FP16 support for Vega 7nm TensorFlow v1.12 is enabled with fp16 support PyTorch/Caffe2 with Vega 7nm Support

fp16 support is enabled

Several bug fixes and performance enhancements

Known Issue: breaking changes are introduced in ROCm 2.0 which are not addressed upstream yet. Meanwhile, please continue to use ROCm fork at https://github.com/ROCmSoftwarePlatform/pytorch

Improvements to ROCProfiler tool

Support for Vega 7nm

Support for hipStreamCreateWithPriority

Creates a stream with the specified priority. It creates a stream on which enqueued kernels have a different priority for execution compared to kernels enqueued on normal priority streams. The priority could be higher or lower than normal priority streams.

OpenCL 2.0 support

ROCm 2.0 introduces full support for kernels written in the OpenCL 2.0 C language on certain devices and systems. Applications can detect this support by calling the “clGetDeviceInfo” query function with “parame_name” argument set to “CL_DEVICE_OPENCL_C_VERSION”.

In order to make use of OpenCL 2.0 C language features, the application must include the option “-cl-std=CL2.0” in options passed to the runtime API calls responsible for compiling or building device programs. The complete specification for the OpenCL 2.0 C language can be obtained using the following link: https://www.khronos.org/registry/OpenCL/specs/opencl-2.0-opencl.pdf
Improved Virtual Addressing (48 bit VA) management for Vega 10 and later GPUs
Fixes Clang AddressSanitizer and potentially other 3rd-party memory debugging tools with ROCm
Small performance improvement on workloads that do a lot of memory management
Removes virtual address space limitations on systems with more VRAM than system memory Kubernetes support

2.4.18 New features and enhancements in ROCm 1.9.2

- RDMA(MPI) support on Vega 7nm
- Support ROCnRDMA based on Mellanox InfiniBand
- Improvements to HCC
- Improved link time optimization
- Improvements to ROCProfiler tool
- General bug fixes and implemented versioning APIs
- New features and enhancements in ROCm 1.9.2

2.4.19 New features and enhancements in ROCm 1.9.1

- Added DPM support to Vega 7nm
- Dynamic Power Management feature is enabled on Vega 7nm.
- Fix for ‘ROCm profiling’ that used to fail with a “Version mismatch between HSA runtime and libhsa-runtime-tools64.so.1” error

2.4.20 New features and enhancements in ROCm 1.9.0

- Preview for Vega 7nm Enables developer preview support for Vega 7nm
- System Management Interface Adds support for the ROCm SMI (System Management Interface) library, which provides monitoring and management capabilities for AMD GPUs.
- Improvements to HIP/HCC Support forgfx906
- Added deprecation warning for C++AMP. This will be the last version of HCC supporting C++AMP.
- Improved optimization for global address space pointers passing into a GPU kernel
- Fixed several race conditions in the HCC runtime
- Performance tuning to the unpinned copy engine
Several codegen enhancement fixes in the compiler backend

Preview for rocprof Profiling Tool

Developer preview (alpha) of profiling tool rocProfiler. It includes a command-line front-end, rpl_run.sh, which enables:
Cmd-line tool for dumping public per kernel perf-counters/metrics and kernel timestamps
Input file with counters list and kernels selecting parameters
Multiple counters groups and app runs supported
Output results in CSV format

The tool can be installed from the rocprofiler-dev package. It will be installed into: /opt/rocm/bin/rpl_run.sh

Preview for rocr Debug Agent rocr_debug_agent

The ROCr Debug Agent is a library that can be loaded by ROCm Platform Runtime to provide the following functionality:
Print the state for wavefronts that report memory violation or upon executing a “s_trap 2” instruction. Allows SIGINT (ctrl c) or SIGTERM (kill -15) to print wavefront state of aborted GPU dispatches. It is enabled on Vega10 GPUs on ROCm1.9. The ROCm1.9 release will install the ROCr Debug Agent library at /opt/rocm/lib/librocr_debug_agent64.so

New distribution support Binary package support for Ubuntu 18.04 ROCm 1.9 is ABI compatible with KFD in upstream Linux kernels. Upstream Linux kernels support the following GPUs in these releases: 4.17: Fiji, Polaris 10, Polaris 11 4.18: Fiji, Polaris 10, Polaris 11, Vega10

Some ROCm features are not available in the upstream KFD:
More system memory available to ROCm applications Interoperability between graphics and compute RDMA IPC To try ROCm with an upstream kernel, install ROCm as normal, but do not install the rock-dkms package. Also add a udev rule to control /dev/kfd permissions:

```
    echo 'SUBSYSTEM=""kfd", KERNEL=""kfd", TAG+="uaccess", GROUP="video"' | sudo tee /etc/udev/rules.d/70-kfd.rules
```

### 2.4.21 New features as of ROCm 1.8.3

ROCm 1.8.3 is a minor update meant to fix compatibility issues on Ubuntu releases running kernel 4.15.0-33

### 2.4.22 New features as of ROCm 1.8

DKMS driver installation

Debian packages are provided for DKMS on Ubuntu

RPM packages are provided for CentOS/RHEL 7.4 and 7.5

See the ROCT-Thunk-Interface and ROCK-Kernel-Driver for additional documentation on driver setup

New distribution support

Binary package support for Ubuntu 16.04 and 18.04

Binary package support for CentOS 7.4 and 7.5

Binary package support for RHEL 7.4 and 7.5

Improved OpenMPI via UCX support
UCX support for OpenMPI
ROCm RDMA

2.4.23 New Features as of ROCm 1.7

DKMS driver installation
New driver installation uses Dynamic Kernel Module Support (DKMS)
Only amdkfd and amdgpu kernel modules are installed to support AMD hardware
Currently only Debian packages are provided for DKMS (no Fedora support available)
See the ROCT-Thunk-Interface and ROCK-Kernel-Driver for additional documentation on driver setup

2.4.24 New Features as of ROCm 1.5

Developer preview of the new OpenCL 1.2 compatible language runtime and compiler
OpenCL 2.0 compatible kernel language support with OpenCL 1.2 compatible runtime
Supports offline ahead of time compilation today; during the Beta phase we will add in-process/in-memory compilation.
Binary Package support for Ubuntu 16.04
Binary Package support for Fedora 24 is not currently available
Dropping binary package support for Ubuntu 14.04, Fedora 23
IPC support

2.5 HCC Deprecation Notice

In the v3.5 release, the Heterogeneous Compute Compiler (HCC) compiler is deprecated and the HIP-Clang compiler is introduced for compiling Heterogeneous-Compute Interface for Portability (HIP) programs.

NOTE: The HCC environment variables will be gradually deprecated in subsequent releases.

The majority of the codebase for the HIP-Clang compiler has been upstreamed to the Clang trunk. The HIP-Clang implementation has undergone a strict code review by the LLVM/Clang community and comprehensive tests consisting of LLVM/Clang build bots. These reviews and tests resulted in higher productivity, code quality, and lower cost of maintenance.

FAQ Transition to HIP from HCC
HIP Porting Guide
2.6 HIP: Heterogeneous-Computing Interface for Portability

2.6.1 ROCm Supported Languages

2.6.1.1 ROCm, Lingua Franca, C++, OpenCL and Python

The open-source ROCm stack offers multiple programming-language choices. The goal is to give you a range of tools to help solve the problem at hand. Here, we describe some of the options and how to choose among them.

2.6.2 HIP Programming Guide

What is Heterogeneous-Computing Interface for Portability (HIP)? It’s a C++ dialect designed to ease conversion of Cuda applications to portable C++ code. It provides a C-style API and a C++ kernel language. The C++ interface can use templates and classes across the host/kernel boundary.

The HIPify tool automates much of the conversion work by performing a source-to-source transformation from Cuda to HIP. HIP code can run on AMD hardware (through the HCC compiler) or Nvidia hardware (through the NVCC compiler) with no performance loss compared with the original Cuda code.

Programmers familiar with other GPGPU languages will find HIP very easy to learn and use. AMD platforms implement this language using the HC dialect described above, providing similar low-level control over the machine.

Use HIP when converting Cuda applications to portable C++ and for new projects that require portability between AMD and NVIDIA. HIP provides a C++ development language and access to the best development tools on both platforms.

2.6.2.1 OpenCL™: Open Compute Language

What is OpenCL? It’s a framework for developing programs that can execute across a wide variety of heterogeneous platforms. AMD, Intel and Nvidia GPUs support version 1.2 of the specification, as do x86 CPUs and other devices (including FPGAs and DSPs). OpenCL provides a C run-time API and C99-based kernel language.

2.6.2.1.1 When to Use OpenCL

Use OpenCL when you have existing code in that language and when you need portability to multiple platforms and devices. It runs on Windows, Linux and Mac OS, as well as a wide variety of hardware platforms (described above).

2.6.2.2 Anaconda Python With Numba

What is Anaconda? It’s a modern open-source analytics platform powered by Python. Continuum Analytics, a ROCm platform partner, is the driving force behind it. Anaconda delivers high-performance capabilities including acceleration of HSA APUs, as well as ROCm-enabled discrete GPUs via Numba. It gives superpowers to the people who are changing the world.
2.6.2.2.1 Numba

Numba gives you the power to speed up your applications with high-performance functions written directly in Python. Through a few annotations, you can just-in-time compile array-oriented and math-heavy Python code to native machine instructions—offering performance similar to that of C, C++ and Fortran—without having to switch languages or Python interpreters.

Numba works by generating optimized machine code using the LLVM compiler infrastructure at import time, run time or statically (through the included Pycc tool). It supports Python compilation to run on either CPU or GPU hardware and is designed to integrate with Python scientific software stacks, such as NumPy.

- Anaconda® with Numba acceleration

2.6.2.2.2 When to Use Anaconda

Use Anaconda when you’re handling large-scale data-analytics, scientific and engineering problems that require you to manipulate large data arrays.

2.6.2.3 Wrap-Up

From a high-level perspective, ROCm delivers a rich set of tools that allow you to choose the best language for your application.

- HCC (Heterogeneous Compute Compiler) supports HC dialects
- HIP is a run-time library that layers on top of HCC (for AMD ROCm platforms; for Nvidia, it uses the NVCC compiler)
- The following will soon offer native compiler support for the GCN ISA:
  - OpenCL 1.2+
  - Anaconda (Python) with Numba

All are open-source projects, so you can employ a fully open stack from the language down to the metal. AMD is committed to providing an open ecosystem that gives developers the ability to choose; we are excited about innovating quickly using open source and about interacting closely with our developer community. More to come soon!
### 2.6.2.3.1 Table Comparing Syntax for Different Compute APIs

<table>
<thead>
<tr>
<th>Term</th>
<th>CUDA</th>
<th>HIP</th>
<th>HC</th>
<th>C++-AMP</th>
<th>OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>int deviceId</td>
<td>int deviceId</td>
<td>hc::accelerator</td>
<td>concurrency::accelerator</td>
<td>cl_device</td>
</tr>
<tr>
<td>Queue</td>
<td>cudaStream_t</td>
<td>hipStream_t</td>
<td>hc::accelerator_view</td>
<td>concurrency::accelerator_view</td>
<td>cl_command_queue</td>
</tr>
<tr>
<td>Event</td>
<td>cudaEvent_t</td>
<td>hipEvent_t</td>
<td>hc::completion_future</td>
<td>concurrency::completion_future</td>
<td>cl_event</td>
</tr>
<tr>
<td>Memory</td>
<td>void *</td>
<td>void *</td>
<td>void *; concurrency::array; concurrency::array_view</td>
<td>cl_mem</td>
<td></td>
</tr>
</tbody>
</table>

|                     | grid                  | grid                   | extent                  | extent                  | NDRange                 |
|                     | block                 | block                  | tile                    | tile                    | work-group              |
|                     | thread                | thread                 | wavefront               | N/A                     | work-item               |
|                     | warp                  | warp                   | N/A                     | N/A                     | N/A                     |

| Thread index        | threadIdx.x           | hipThreadIdx_x         | t_idx.local[0]          | t_idx.local[0]          | get_local_id(0)         |
| Block index         | blockIdx.x            | hipBlockIdx_x          | t_idx.tile[0]           | t_idx.tile[0]           | get_group_id(0)         |
| Block dim           | blockDim.x            | hipDimBlock_x          | t_ext.title[dim]        | t_ext.title[dim]        | get_local_size(0)       |
| Grid-dim            | gridDim.x             | hipGridDim_X           | t_ext[0]                | t_ext[0]                | get_global_size(0)      |

| Device Function     | __device__            | __device__             | [hc]) (detected automatically in many case) | restrict(amp)          | Implied in device Compilation |
| Host Function       | __host__              | __host__ (default)     | [cpu]) (default)        | strict(cpu) (default)  | Implied in host Compilation |
| Host + Device Function | __host__ __device__  | __host__ __device__    | [hc]) [cpu]             | restrict(amp,cpu)      | No equivalent            |

| Kernel Launch       | <<< >>>               | __host__ __device__    | hc::parallel_for_each   | concurrency::parallel_for_each | clEnqueueNDRangeKernel |

| Global Memory       | __global__            | __global__             | Unnecessary/Implied     | Unnecessary/Implied        | global                  |
| Group Memory        | __shared__            | __shared__             | tile_static             | tile_static               | __local                 |
| Constant            | __constant__          | __constant__           | Unnecessary/Implied     | Unnecessary / Implied     | __constant              |
|                     | __syncthreads         | __syncthreads          | tile_static.barrier()   | t_id.barrier()            | barrier(CLK_LOCAL_MEMFENCE) |
| Atomic Builtin      | atomicAdd             | atomicAdd              | hc::atomic_fetch_add    | concurrency::atomic_fetch_add | atomic_add               |
|                     | __cos(f)              | __cos(f)               | hc::fast_math::cos(f)   | concurrency::fast_math::cos(f) | native_cos(f)           |

---

2.6. HIP: Heterogeneous-Computing Interface for Portability
2.6.2.3.2 Notes

1. For HC and C++AMP, assume a captured _tiled_ext_ named “t_ext” and captured _extent_ named “ext”. These languages use captured variables to pass information to the kernel rather than using special built-in functions so the exact variable name may vary.

2. The indexing functions (starting with `thread-index`) show the terminology for a 1D grid. Some APIs use reverse order of xyz / 012 indexing for 3D grids.

3. HC allows tile dimensions to be specified at runtime while C++AMP requires that tile dimensions be specified at compile-time. Thus hc syntax for tile dims is `t_ext.tile_dim[0]` while C++AMP is `t_ext.tile_dim0`.

4. From ROCm version 2.0 onwards C++AMP is no longer available in HCC.

2.6.3 HIP Repository Information

What is this repository for?

HIP allows developers to convert CUDA code to portable C++. The same source code can be compiled to run on NVIDIA or AMD GPUs. Key features include:

- HIP is very thin and has little or no performance impact over coding directly in CUDA or hcc “HC” mode.
- HIP allows coding in a single-source C++ programming language including features such as templates, C++11 lambdas, classes, namespaces, and more.
- HIP allows developers to use the “best” development environment and tools on each target platform.
- The “hipify” tool automatically converts source from CUDA to HIP.
- Developers can specialize for the platform (CUDA or hcc) to tune for performance or handle tricky cases.

New projects can be developed directly in the portable HIP C++ language and can run on either NVIDIA or AMD platforms. Additionally, HIP provides porting tools which make it easy to port existing CUDA codes to the HIP layer, with no loss of performance as compared to the original CUDA application. HIP is not intended to be a drop-in replacement for CUDA, and developers should expect to do some manual coding and performance tuning work to complete the port.

Repository branches:

The HIP repository maintains several branches. The branches that are of importance are:

master branch: This is the stable branch. All stable releases are based on this branch. developer-preview branch: This is the branch were the new features still under development are visible. While this maybe of interest to many, it should be noted that this branch and the features under development might not be stable.

Release tagging:

HIP releases are typically of two types. The tag naming convention is different for both types of releases to help differentiate them.

release_x.yy.zzzz: These are the stable releases based on the master branch. This type of release is typically made once a month. preview_x.yy.zzzz: These denote pre-release code and are based on the developer-preview branch. This type of release is typically made once a week.

More Info

HIP provides a C++ syntax that is suitable for compiling most code that commonly appears in compute kernels, including classes, namespaces, operator overloading, templates and more. Additionally, it defines other language features designed specifically to target accelerators, such as the following:
• A kernel-launch syntax that uses standard C++, resembles a function call and is portable to all HIP targets
• Short-vector headers that can serve on a host or a device
• Math functions resembling those in the “math.h” header included with standard C++ compilers
• Built-in functions for accessing specific GPU hardware capabilities

This section describes the built-in variables and functions accessible from the HIP kernel. It’s intended for readers who are familiar with Cuda kernel syntax and want to understand how HIP is different.

• HIP-GUIDE

2.6.4 Transitioning from HCC to HIP

• HIP-FAQ
• HIP-porting-guide
• hip-pro

2.6.5 HIP Best Practices

• HIP-IN
• Kernel_language
• HIP Runtime API (Doxygen)
• hip-p
• hip_profiling
• HIP_Debugging
• HIP-terminology
• HIP-Term2
• hipify-clang

Supported CUDA APIs:
• CUDAAPICUDA
• CUDAAPICUPTEXHERE
• cuComplex API
• cuBLAS
• cuRAND
• cuDNN
• cuFFT
• cuSPARSE
• Developer/CONTRIBUTING Info
• Release Notes
Simple Example

The HIP API includes functions such as hipMalloc, hipMemcpy, and hipFree. Programmers familiar with CUDA will also be able to quickly learn and start coding with the HIP API. Compute kernels are launched with the “hipLaunchKernelGGL” macro call. Here is simple example showing a snippet of HIP API code:

```c
hipMalloc(&A_d, Nbytes); hipMalloc(&C_d, Nbytes));
hipMemcpy(A_d, A_h, Nbytes, hipMemcpyHostToDevice);
const unsigned blocks = 512; const unsigned threadsPerBlock = 256; hipLaunchKernelGGL(vector_square, /* compute kernel*/
    dim3(blocks), dim3(threadsPerBlock), 0/dynamic shared/, 0/stream/, /* launch config*/ C_d, A_d, N); /* arguments to the compute kernel */
hipMemcpy(C_h, C_d, Nbytes, hipMemcpyDeviceToHost);
```

The HIP kernel language defines builtins for determining grid and block coordinates, math functions, short vectors, atomics, and timer functions. It also specifies additional defines and keywords for function types, address spaces, and optimization controls. (See the Kernel language for a full description). Here’s an example of defining a simple ‘vector_square’ kernel.

```c
template<typename T> __global__ void vector_square(T *C_d, const T *A_d, size_t N) {
    size_t offset = (hipBlockIdx_x * hipBlockDim_x + hipThreadIdx_x); size_t stride = hipBlockDim_x * hipGridDim_x ;
    for (size_t i=offset; i<N; i+=stride) { C_d[i] = A_d[i] * A_d[i]; }
}
```

The HIP Runtime API code and compute kernel definition can exist in the same source file - HIP takes care of generating host and device code appropriately.

HIP Portability and Compiler Technology

HIP C++ code can be compiled with either:

- On the NVIDIA CUDA platform, HIP provides header file which translate from the HIP runtime APIs to CUDA runtime APIs. The header file contains mostly inlined functions and thus has very low overhead - developers coding in HIP should expect the same performance as coding in native CUDA. The code is then compiled with nvcc, the standard C++ compiler provided with the CUDA SDK. Developers can use any tools supported by the CUDA SDK including the CUDA profiler and debugger.

- On the AMD ROCm platform, HIP provides a header and runtime library built on top of hcc compiler. The HIP runtime implements HIP streams, events, and memory APIs, and is a object library that is linked with the application. The source code for all headers and the library implementation is available on GitHub.

HIP developers on ROCm can use AMD’s CodeXL for debugging and profiling.

Thus HIP source code can be compiled to run on either platform. Platform-specific features can be isolated to a specific platform using conditional compilation. Thus HIP provides source portability to either platform. HIP provides the hipcc compiler driver which will call the appropriate toolchain depending on the desired platform.

Examples and Getting Started:

- A sample and blog that uses hipify to convert a simple app from CUDA to HIP:
  ```sh
cd samples/01_Intro/square # follow README / blog steps to hipify the application.
```
• A sample and blog demonstrating platform specialization:
  cd samples/01_Intro/bit_extract make

• Guide to Porting a New Cuda Project

More Examples
The GitHub repository HIP-Examples contains a hipified version of the popular Rodinia benchmark suite. The README with the procedures and tips the team used during this porting effort is here: Rodinia Porting Guide

Tour of the HIP Directories
  include:
  • hip_runtime_api.h : Defines HIP runtime APIs and can be compiled with many standard Linux compilers (hcc, GCC, ICC, CLANG, etc), in either C or C++ mode.
  • hip_runtime.h : Includes everything in hip_runtime_api.h PLUS hipLaunchKernelGGL and syntax for writing device kernels and device functions. hip_runtime.h can only be compiled with hcc.
  • hcc_detail/ , nvcc_detail/** : Implementation details for specific platforms. HIP applications should not include these files directly.
  • hcc.h : Includes interop APIs for HIP and HCC
  • bin: Tools and scripts to help with hip porting
    – hipify : Tool to convert CUDA code to portable CPP. Converts CUDA APIs and kernel builtins.
    – hipcc : Compiler driver that can be used to replace nvcc in existing CUDA code. hipcc will call nvcc or hcc depending on platform, and include appropriate platform-specific headers and libraries.
    – hipconfig : Print HIP configuration (HIP_PATH, HIP_PLATFORM, CXX config flags, etc)
    – hipexamine.sh : Script to scan directory, find all code, and report statistics on how much can be ported with HIP (and identify likely features not yet supported)
  • doc: Documentation - markdown and doxygen info

2.7 OpenCL Programming Guide

• Opencl-Programming-Guide
• Optimization-Opencl

2.8 HCC Programming Guide

What is the Heterogeneous Compute (HC) API?
It’s a C++ dialect with extensions to launch kernels and manage accelerator memory. It closely tracks the evolution of C++ and will incorporate parallelism and concurrency features as the C++ standard does. For example, HC includes early support for the C++17 Parallel STL. At the recent ISO C++ meetings in Kona and Jacksonville, the committee was excited about enabling the language to express all forms of parallelism, including multcore CPU, SIMD and GPU. We’ll be following these developments closely, and you’ll see HC move quickly to include standard C++ capabilities.

The Heterogeneous Compute Compiler (HCC) provides two important benefits:
Ease of development

• A full C++ API for managing devices, queues and events
• C++ data containers that provide type safety, multidimensional-array indexing and automatic data management
• C++ kernel-launch syntax using parallel_for_each plus C++11 lambda functions
• A single-source C++ programming environment—the host and source code can be in the same source file and use the same C++ language; templates and classes work naturally across the host/device boundary
• HCC generates both host and device code from the same compiler, so it benefits from a consistent view of the source code using the same Clang-based language parser

Full control over the machine

• Access AMD scratchpad memories ("LDS")
• Fully control data movement, prefetch and discard
• Fully control asynchronous kernel launch and completion
• Get device-side dependency resolution for kernel and data commands (without host involvement)
• Obtain HSA agents, queues and signals for low-level control of the architecture using the HSA Runtime API
• Use direct-to-ISA compilation

When to Use HC Use HC when you’re targeting the AMD ROCm platform: it delivers a single-source, easy-to-program C++ environment without compromising performance or control of the machine.

HC comes with two header files as of now:

• hc.hpp : Main header file for HC
• hc_math.hpp : Math functions for HC

Most HC APIs are stored under “hc” namespace, and the class name is the same as their counterpart in C++AMP “Concurrency” namespace. Users of C++AMP should find it easy to switch from C++AMP to HC.

<table>
<thead>
<tr>
<th>C++AMP</th>
<th>HC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency::accelerator</td>
<td>hc::accelerator</td>
</tr>
<tr>
<td>Concurrency::accelerator_view</td>
<td>hc::accelerator_view</td>
</tr>
<tr>
<td>Concurrency::extent</td>
<td>hc::extent</td>
</tr>
<tr>
<td>Concurrency::index</td>
<td>hc::index</td>
</tr>
<tr>
<td>Concurrency::completion_future</td>
<td>hc::completion_future</td>
</tr>
<tr>
<td>Concurrency::array</td>
<td>hc::array</td>
</tr>
<tr>
<td>Concurrency::array_view</td>
<td>hc::array_view</td>
</tr>
</tbody>
</table>

• relaxed rules in operations allowed in kernels
• new syntax of tiled_extent and tiled_index
• dynamic group segment memory allocation
• true asynchronous kernel launching behavior
• additional HSA-specific APIs

Despite HC and C++ AMP sharing many similar program constructs (e.g. parallel_for_each, array, array_view, etc.), there are several significant differences between the two APIs.

Support for explicit asynchronous parallel_for_each In C++ AMP, the parallel_for_each appears as a synchronous function call in a program (i.e. the host waits for the kernel to complete); however, the compiler may optimize it to execute the kernel asynchronously and the host would synchronize with the device on the first access of the data.
modified by the kernel. For example, if a parallel_for_each writes the an array_view, then the first access to this 
array_view on the host after the parallel_for_each would block until the parallel_for_each completes.

HC supports the automatic synchronization behavior as in C++ AMP. In addition, HC’s parallel_for_each supports 
explicit asynchronous execution. It returns a completion_future (similar to C++ std::future) object that other asyn-
cronous operations could synchronize with, which provides better flexibility on task graph construction and enables 
more precise control on optimization.

Annotation of device functions

C++ AMP uses the restrict(amp) keyword to annotate functions that runs on the device.

```c
void foo() restrict(amp) { .. } ... parallel_for_each(..., [=] () restrict(amp) {
  foo(); });
```

HC uses a function attribute ([hc] or __attribute__((hc)) ) to annotate a device function.

```c
void foo() [hc] { .. } ... parallel_for_each(..., [=] () [hc] { foo(); });
```

The [hc] annotation for the kernel function called by parallel_for_each is optional as it is automatically annotated 
as a device function by the hcc compiler. The compiler also supports partial automatic [hc] annotation for functions 
that are called by other device functions within the same source file:

Since bar is called by foo, which is a device function, the hcc compiler will automatically annotate bar as a device 
function

```c
void bar() { ... } void foo() [hc] { bar(); }
```

Dynamic tile size

C++ AMP doesn’t support dynamic tile size. The size of each tile dimensions has to be a compile-time constant 
specified as template arguments to the tile_extent object:

```c
extent<2> ex(x, y)
```

To create a tile extent of 8x8 from the extent object, note that the tile dimensions have to be constant 
values:

```c
tiled_extent<8,8> t_ex(ex)
```

```c
parallel_for_each(t_ex, [=](tiled_index<8,8> t_id) restrict(amp) { ... });
```

HC supports both static and dynamic tile size:

```c
extent<2> ex(x,y)
```

To create a tile extent from dynamically calculated values, note that the the tiled_extent template takes the rank instead 
of dimensions

```c
tx = test_x ? tx_a : tx_b;
ty = test_y ? ty_a : ty_b;
tiled_extent<2> t_ex(tx, ty);
```

```c
parallel_for_each(t_ex, [=](tiled_index<2> t_id) [[hc]] { ... });
```

Support for memory pointer

C++ AMP doesn’t support lambda capture of memory pointer into a GPU kernel.

HC supports capturing memory pointer by a GPU kernel.

allocate GPU memory through the HSA API .. code:: sh

```c
int* gpu_pointer; hsa_memory_allocate(..., &gpu_pointer); ... parallel_for_each(ext, [=](index i) [[hc]]
{ gpu_pointer[i[0]]++; }
```
For HSA APUs that supports system wide shared virtual memory, a GPU kernel can directly access system memory allocated by the host:

```sh
int* cpu_memory = (int*) malloc(...); ... parallel_for_each(ext, [=](index i) [[hc]] {
    cpu_memory[i[0]]++;
});
```

# 2.9 Compiler SDK

## 2.9.1 GCN Native ISA LLVM Code Generator

- ROCm-Native-ISA

## 2.9.2 ROCm Code Object Format

- ROCm-Codeobj-format

## 2.9.3 ROCm Device Library

### 2.9.3.1 Overview

This repository contains the following libraries:

<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>oclc*</td>
<td>Open Compute library controls</td>
<td>oclc*</td>
</tr>
<tr>
<td>ocml</td>
<td>Open Compute Math library</td>
<td>oclc*</td>
</tr>
<tr>
<td>ockl</td>
<td>Open Compute Kernel library</td>
<td>oclc*</td>
</tr>
<tr>
<td>opencl</td>
<td>OpenCL built-in library</td>
<td>ocml,ockl,oclc*</td>
</tr>
<tr>
<td>hip</td>
<td>HIP built in library</td>
<td>ocml,ockl,oclc*</td>
</tr>
<tr>
<td>hc</td>
<td>Heterogeneous Compute built-in library</td>
<td>ocml,ockl,oclc*</td>
</tr>
</tbody>
</table>

### 2.9.3.2 Building

The library sources should be compiled using a clang compiler built from sources in the amd-stg-open branch of AMD-modified llvm-project repository. Use the following commands:

```bash
git clone https://github.com/RadeonOpenCompute/llvm-project.git -b amd-stg-open llvm_amd
cd llvm_amd
mkdir -p build
cd build
cmake \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_INSTALL_PREFIX=/opt/rocm/llvm \
    -DLLVM_ENABLE_PROJECTS="clang;lld" \
    -DLLVM_TARGETS_TO_BUILD="AMDGPU;X86" \
    ../llvm
make
```

To build the library bitcodes, clone the amd_stg_open branch of this repository. Run the following commands:

```bash
:: git clone https://github.com/RadeonOpenCompute/ROCM-Device-Libs.git -b amd-stg-open
```
and from its top level run the following commands:

```bash
mkdir -p build
cd build
export LLVM_BUILD=... (path to LLVM build directory created above)
CC=$LLVM_BUILD/bin/clang cmake -DLLVM_DIR=$LLVM_BUILD ..
make
```

It is also possible to use compiler that only has AMDGPU target enabled if you build prepare-builtins separately with host compiler and pass explicit target option to CMake:

```bash
export LLVM_BUILD=... (path to LLVM build)
# Build prepare-builtins
cd utils
dir build
cd build
cmake -DLLVM_DIR=$LLVM_BUILD ..
mak
# Build bitcode libraries
cd ../..
dir build
cmake -DLLVM_DIR=$LLVM_BUILD -DAMDHSACOD=$HSA_DIR/bin/x86_64/amdhsacod -DCMAKE_C_FLAGS="-target amdgcn--amdhsa" DCMAKE_CXX_FLAGS="-target amdgcn--amdhsa" -DPREPARE_BUILTINS=cd ../utils/build/prepare-builtins;/pwd'/
```

To install artifacts: `make install`

To create packages for the library: `make package`

### 2.9.3.3 Using Bitcode Libraries

The ROCm language runtimes automatically add the required bitcode files during the LLVM linking stage invoked during the process of creating a code object. There are options to display the exact commands executed, but an approximation of the command the OpenCL runtime might use is as follows:

```bash
$LLVM_BUILD/bin/clang -x cl -Xclang -finclude-default-header 
   -target amdgcn-amd-amdhsa -mcpu=gfx900 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/opencl/openc1.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/ocml/ocml.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/oclc/oclc_correctly_rounded_sqrt_off.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/oclc/oclc_daz_opt_off.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/oclc/oclc_finite_only_off.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/oclc/oclc_unsafe_math_off.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/oclc/oclc_wavfrontsize64_off.amdgcn.bc 
   -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/oclc/oclc_isa_version_900.amdgcn.bc 
```

(continues on next page)
2.9.3.4 Using from Cmake

The bitcode libraries are exported as CMake targets, organized in a CMake package. You can depend on this package using find_package(AMDDeviceLibs REQUIRED CONFIG) after ensuring the CMAKE_PREFIX_PATH includes either the build directory or install prefix of the bitcode libraries. The package defines a variable AMD_DEVICE_LIBS_TARGETS containing a list of the exported CMake targets.

2.9.4 ROCr Runtime

Github link of ROCr Runtime check Here

2.9.4.1 HSA Runtime API and runtime for ROCm

This repository includes the user-mode API interfaces and libraries necessary for host applications to launch compute kernels to available HSA ROCm kernel agents. Reference source code for the core runtime is also available. Initial target platform requirements

- CPU: Intel Haswell or newer, Core i5, Core i7, Xeon E3 v4 & v5; Xeon E5 v3
- GPU: Fiji ASIC (AMD R9 Nano, R9 Fury and R9 Fury X)
- GPU: Polaris ASIC (AMD RX480)

2.9.4.2 Source code

The HSA core runtime source code for the ROCR runtime is located in the src subdirectory. Please consult the associated README.md file for contents and build instructions.

2.9.4.3 Binaries for Ubuntu & Fedora and installation instructions

Pre-built binaries are available for installation from the ROCm package repository. For ROCR, they include:

Core runtime package:

- HSA include files to support application development on the HSA runtime for the ROCR runtime
- A 64-bit version of AMD’s HSA core runtime for the ROCR runtime

Runtime extension package:

- A 64-bit version of AMD’s runtime tools library
- A 64-bit version of AMD’s runtime image library, which supports the HSAIL image implementation only.

The contents of these packages are installed in /opt/rocsm/hsa and /opt/rocsm by default. The core runtime package depends on the hsakmt-roct-dev package.

Installation instructions can be found in the ROCm Documentation
2.9.4.4 Infrastructure

The HSA runtime is a thin, user-mode API that exposes the necessary interfaces to access and interact with graphics hardware driven by the AMDGPU driver set and the ROCK kernel driver. Together they enable programmers to directly harness the power of AMD discrete graphics devices by allowing host applications to launch compute kernels directly to the graphics hardware.

The capabilities expressed by the HSA Runtime API are:

- Error handling
- Runtime initialization and shutdown
- System and agent information
- Signals and synchronization
- Architected dispatch
- Memory management
- HSA runtime fits into a typical software architecture stack.

The HSA runtime provides direct access to the graphics hardware to give the programmer more control of the execution. An example of low level hardware access is the support of one or more user mode queues provides programmers with a low-latency kernel dispatch interface, allowing them to develop customized dispatch algorithms specific to their application.

The HSA Architected Queuing Language is an open standard, defined by the HSA Foundation, specifying the packet syntax used to control supported AMD/ATI Radeon (c) graphics devices. The AQL language supports several packet types, including packets that can command the hardware to automatically resolve inter-packet dependencies (barrier AND & barrier OR packet), kernel dispatch packets and agent dispatch packets.

In addition to user mode queues and AQL, the HSA runtime exposes various virtual address ranges that can be accessed by one or more of the system’s graphics devices, and possibly the host. The exposed virtual address ranges either support a fine grained or a coarse grained access. Updates to memory in a fine grained region are immediately visible to all devices that can access it, but only one device can have access to a coarse grained allocation at a time. Ownership of a coarse grained region can be changed using the HSA runtime memory APIs, but this transfer of ownership must be explicitly done by the host application.

Programmers should consult the HSA Runtime Programmer’s Reference Manual for a full description of the HSA Runtime APIs, AQL and the HSA memory policy.

2.9.4.5 Known issues

- Each HSA process creates an internal DMA queue, but there is a system-wide limit of four DMA queues. When the limit is reached HSA processes will use internal kernels for copies.

Disclaimer

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale.
2.10 System Management Interface Event API

An System Management Interface (SMI) event interface is added to the kernel and a ROCm SMI library for system administrators to get notified when specific events occur. On the kernel side, AMDKFD_IOC_SMI_EVENTS input/output control is enhanced to allow notifications propagation to user mode through the event channel.

On the ROCm SMI lib side, APIs are added to set an event mask and receive event notifications with a timeout option. Further, ROCm SMI API details can be found in the PDF generated by Doxygen from source or by referring to the rocm_smi.h header file (see the rsmi_event_notification_* functions).

For more information, see the System Management Interface API guide at:

2.11 System Management Interface Tool

This repository includes the AMD ROCm-SMI tool. This tool exposes functionality for clock and temperature management of the ROCm-enabled system.

**Installation**

You may find rocm-smi at the following location after installing the rocm package:

```
/opt/rocm/bin/rocm-smi
```

Alternatively, you may clone this repository and run the tool directly.

**Version**

The SMI will report a “version” which is the version of the kernel installed:

AMD ROCm System Management Interface v$(uname)

For ROCk installations, this will be the AMDGPU module version (e.g. 5.0.71) For non-ROCk or monolithic ROCk installations, this will be the kernel version, which will be equivalent to the following bash command:

```
$(uname -a) | cut -d ' ' -f 3)
```

**Usage**

For detailed and up to date usage information, we recommend consulting the help:

```
/opt/rocm/bin/rocm-smi -h
```

For convenience purposes, following is the output from the -h flag:

```
AMD ROCm System Management Interface | ROCM-SMI version: 1.3.1

```

-h, --help show this help message and exit
--gpureset Reset specified GPU (One GPU must be specified)
--load FILE Load Clock, Fan, Performance and Profile settings
--save FILE Save Clock, Fan, Performance and Profile settings

-d DEVICE [DEVICE ...], --device DEVICE [DEVICE ...] Execute command on specified device

Display Options:
--alldevices Execute command on non-AMD devices as well as AMD devices
--showhw Show Hardware details
-a, --showallinfo Show Temperature, Fan and Clock values

Topology:
-i, --showid Show GPU ID
-v, --showvbios Show VBIOS version
-showdriverversion Show kernel driver version
-showfwinfo [BLOCK [BLOCK ...]] Show FW information
-showmclkrange Show mclk range
-showmemvendor Show GPU memory vendor
-showsclkrange Show sclk range
-showproductname Show SKU/Vendor name
-showserial Show GPU’s Serial Number
-showuniqueid Show GPU’s Unique ID
-showvoltage Show voltage range
-showbus Show PCI bus number

Pages information:
-showpagesinfo Show retired, pending and unreservable pages
-showpendingpages Show pending retired pages
-showretiredpages Show retired pages
-showunreservablepages Show unreservable pages

Hardware-related information:
-f, --showfan Show current fan speed
-P, --showpower Show current Average Graphics Package Power Consumption
-t, --showtemp Show current temperature
-u, --showuse Show current GPU use
-showmemuse Show current GPU memory used
-showvoltage Show current GPU voltage

Software-related/controlled information:
-b, --showbw Show estimated PCIe use
-c, --showclocks Show current clock frequencies
-g, --showgpuclocks Show current GPU clock frequencies
-l, --showprofile Show Compute Profile attributes
-M, --showmaxpower Show maximum graphics package power this GPU will consume

2.11. System Management Interface Tool 53
Table 1 – continued from previous page

-m, --showmemoverdrive   Show current GPU Memory Clock OverDrive level
-o, --showoverdrive     Show current GPU Clock OverDrive level
-p, --showperfilvel     Show current DPM Performance Level
-S, --showclckvlt      Show supported GPU and Memory Clocks and Voltages
-s, --showclckfrq      Show supported GPU and Memory Clock
--showmeminfo TYPE [TYPE . . . ]  Show Memory usage information for given block(s) TYPE
--showpids             Show current running KFD PIDs
--showreplaycount      Show PCIe Replay Count
--showrasinfo BLOCK [BLOCK . . . ] Show RAS enablement information and error counts for the specified block(s)
--showvc               Show voltage curve
--showxgmierr          Show XGMI error information since last read

Set options:

--setsclk LEVEL [LEVEL . . . ] Set GPU Clock Frequency Level(s) (requires manual Perf level)
--setmclk LEVEL [LEVEL . . . ] Set GPU Memory Clock Frequency Level(s) (requires manual Perf level)
--setpcie LEVEL [LEVEL . . . ] Set PCIE Clock Frequency Level(s) (requires manual Perf level)
--setsllevel SCLKLEVEL SCLK SVOLT Change GPU Clock frequency (MHz) and Voltage (mV) for a specific Level
--setmlevel MCLKLEVEL MCLK MVOLT Change GPU Memory clock frequency (MHz) and Voltage for (mV) a specific Level
--setvc POINT SCLK SVOLT Change SCLK Voltage Curve (MHz mV) for a specific point
--setrange MINMAX SCLK Set min(0) or max(1) SCLK speed
--setmrange MINMAX SCLK Set min(0) or max(1) MCLK speed
--setfan LEVEL         Set GPU Fan Speed (Level or %)
--setperflevel LEVEL  Set Performance Level
--setoverdrive %       Set GPU OverDrive level (requires manual high Perf level)
--setmemoverdrive %    Set GPU Memory Overclock OverDrive level (requires manual high Perf level)
--setpoweroverdrive WATTS Set the maximum GPU power using Power OverDrive in Watts
--setprofile SETPROFILE Specify Power Profile level (#) or a quoted string of CUSTOM Profile attributes “## # #”
--rasenable BLOCK ERRTYPE Enable RAS for specified block and error type
--rasdisable BLOCK ERRTYPE Disable RAS for specified block and error type
--rasinject BLOCK      Inject RAS poison for specified block (ONLY WORKS ON UNSECURE BOARDS)

Reset options:

-r, --resetclocks     Reset clocks and OverDrive to default
--resetfans           Reset fans to automatic (driver) control
--resetprofile        Reset Power Profile back to default
--resetpoweroverdrive Set the maximum GPU power back to the default state
--resetxgmierr        Reset XGMI error count

Auto-response options:

--autorespond RESPONSE Response to automatically provide for all prompts (NOT RECOMMENDED)

Output options:

--loglevel LEVEL      How much output will be printed for what program is doing, one of debug/info/warning/error/critical
--json                Print output in JSON format

Detailed Option Descriptions

--setsclk|--setmclk # [# # . . . ]: This allows you to set a mask for the levels. For example, if a GPU has 8 clock levels, you can set a mask to use levels 0, 5, 6 and 7 with --setsclk 0 5 6 7. This will only use the base level, and the top 3 clock levels. This will allow you to keep the GPU at base level when there is no GPU load, and the top 3 levels when the GPU load increases.

--setfan LEVEL: This sets the fan speed to a value ranging from 0 to 255 (not from 0-100%). If the level ends with a %, the fan speed is calculated as pct*maxlevel/100 (maxlevel is usually 255, but is determined by the ASIC). NOTE: While the hardware is usually capable of overriding this value when required, it is
recommended to not set the fan level lower than the default value for extended periods of time

-setperlevel LEVEL: This lets you use the pre-defined Performance Level values, which can include: auto (Automatically change PowerPlay values based on GPU workload) low (Keep PowerPlay values low, regardless of workload) high (Keep PowerPlay values high, regardless of workload) manual (Only use values defined in sysfs values)

-setoverdrive/–setmemoverdrive #: DEPRECATED IN NEWER KERNEL VERSIONS (use –set-slevel/–setmlevel instead) This sets the percentage above maximum for the max Performance Level. For example, –setoverdrive 20 will increase the top sclk level by 20%. If the maximum sclk level is 1000MHz, then –setoverdrive 20 will increase the maximum sclk to 1200MHz

-setpoweroverdrive/–resetpoweroverdrive #: This allows users to change the maximum power available to a GPU package. The input value is in Watts. This limit is enforced by the hardware, and some cards allow users to set it to a higher value than the default that ships with the GPU. This Power OverDrive mode allows the GPU to run at higher frequencies for longer periods of time, though this may mean the GPU uses more power than it is allowed to use per power supply specifications. Each GPU has a model-specific maximum Power OverDrive that is will take; attempting to set a higher limit than that will cause this command to fail.

-setprofile SETPROFILE: The Compute Profile accepts 1 or n parameters, either the Profile to select (see –show-profile for a list of preset Power Profiles) or a quoted string of values for the CUSTOM profile. NOTE: These values can vary based on the ASIC, and may include: SCLK_PROFILE_ENABLE - Whether or not to apply the 3 following SCLK settings (0=disable,1=enable) NOTE: This is a hidden field. If set to 0, the following 3 values are displayed as ‘-’ SCLK_UP_HYST - Delay before sclk is increased (in milliseconds) SCLK_DOWN_HYST - Delay before sclk is decreased (in milliseconds) SCLK_ACTIVE_LEVEL - Workload required before sclk levels change (in %) MCLK_PROFILE_ENABLE - Whether or not to apply the 3 following MCLK settings (0=disable,1=enable) NOTE: This is a hidden field. If set to 0, the following 3 values are displayed as ‘-’ MCLK_UP_HYST - Delay before mclk is increased (in milliseconds) MCLK_DOWN_HYST - Delay before mclk is decreased (in milliseconds) MCLK_ACTIVE_LEVEL - Workload required before mclk levels change (in %)

BUSY_SET_POINT - Threshold for raw activity level before levels change FPS - Frames Per Second
USE_RLC_BUSY - When set to 1, DPM is switched up as long as RLC busy message is received
MIN_ACTIVE_LEVEL - Workload required before levels change (in %)

Note: When a compute queue is detected, these values will be automatically applied to the system Compute Power Profiles are only applied when the Performance Level is set to “auto”
The CUSTOM Power Profile is only applied when the Performance Level is set to “manual” so using this flag will automatically set the performance level to “manual”

It is not possible to modify the non-CUSTOM Profiles. These are hard-coded by the kernel

-P, –showpower: Show Average Graphics Package power consumption
“Graphics Package” refers to the GPU plus any HBM (High-Bandwidth memory) modules, if present

-M, –showmaxpower: Show the maximum Graphics Package power that the GPU will attempt to consume. This limit is enforced by the hardware.

-loglevel: This will allow the user to set a logging level for the SMI’s actions. Currently this is only implemented for sysfs writes, but can easily be expanded upon in the future to log other things from the SMI

-showmeminfo: This allows the user to see the amount of used and total memory for a given block (vram, vis_vram, gtt). It returns the number of bytes used and total number of bytes for each block ‘all’ can be passed as a field to return all blocks, otherwise a quoted-string is used for multiple values (e.g. “vram vis_vram”) vram refers to the Video RAM, or graphics memory, on the specified device vis_vram refers to Visible VRAM, which is the CPU-accessible video memory on the device gtt refers to the Graphics Translation Table

2.11. System Management Interface Tool
-b, --showbw: This shows an approximation of the number of bytes received and sent by the GPU over the last second through the PCIe bus. Note that this will not work for APUs since data for the GPU portion of the APU goes through the memory fabric and does not ‘enter/exit’ the chip via the PCIe interface, thus no accesses are generated, and the performance counters can’t count accesses that are not generated. NOTE: It is not possible to easily grab the size of every packet that is transmitted in real time, so the kernel estimates the bandwidth by taking the maximum payload size (mps), which is the max size that a PCIe packet can be. and multiplies it by the number of packets received and sent. This means that the SMI will report the maximum estimated bandwidth, the actual usage could (and likely will be) less

--showrasinfo: This shows the RAS information for a given block. This includes enablement of the block (currently GFX, SDMA and UMC are the only supported blocks) and the number of errors ue - Uncorrectable errors ce - Correctable errors

Clock Type Descriptions
DCEFCLK - DCE (Display) FCLK - Data fabric (VG20 and later) - Data flow from XGMI, Memory, PCIe SCLK - GFXCLK (Graphics core)

Note: SOCCLK split from SCLK as of Vega10. Pre-Vega10 they were both controlled by SCLK

MCLK - GPU Memory (VRAM) PCLK - PCIe bus

Note: This gives 2 speeds, PCIe Gen1 x1 and the highest available based on the hardware

SOCCLK - System clock (VG10 and later) - Data Fabric (DF), MM HUB, AT HUB, SYSTEM HUB, OSS, DFD Note - DF split from SOCCLK as of Vega20. Pre-Vega20 they were both controlled by SOCCLK

--gpureset: This flag will attempt to reset the GPU for a specified device. This will invoke the GPU reset through the kernel debugfs file amdgpu_gpu_recover. Note that GPU reset will not always work, depending on the manner in which the GPU is hung.

--showdriverversion: This flag will print out the AMDGPU module version for amdgpu-pro or ROCK kernels. For other kernels, it will simply print out the name of the kernel (uname)

--showserial: This flag will print out the serial number for the graphics card NOTE: This is currently only supported on Vega20 server cards that support it. Consumer cards and cards older than Vega20 will not support this feature.

--showproductname: This uses the pci.ids file to print out more information regarding the GPUs on the system. ‘update-pciids’ may need to be executed on the machine to get the latest PCI ID snapshot, as certain newer GPUs will not be present in the stock pci.ids file, and the file may even be absent on certain OS installation types

--showpagesinfo | --showretiredpages | --showpendingpages | --showunreservablepages: These flags display the different “bad pages” as reported by the kernel. The three types of pages are: Retired pages (reserved pages) - These pages are reserved and are unable to be used Pending pages - These pages are pending for reservation, and will be reserved/retired Unreservable pages - These pages are not reservable for some reason.

--showmemuse | --showuse | --showmeminfo --showuse and --showmemuse are used to indicate how busy the respective blocks are. For example, for --showmeminfo (gpu_busy_percent sysfs file), the SMU samples every ms so to see if any GPU block (RLC, MEC, PFP, CP) is busy. If so, that’s 1 (or high). If not, that’s 0 (low). If we have 5 high and 5 low samples, that means 50% utilization (50% GPU busy, or 50% GPU use). The windows and sampling vary from generation to generation, but that is how GPU and VRAM use is calculated in a generic sense. --showmemuse (and VRAM% in concise output) will show the amount of VRAM used (visible, total, GTT), as well as the total available for those partitions. The percentage shown there indicates the amount of used memory in terms of current allocations

OverDrive settings

• Enabling OverDrive requires both a card that support OverDrive and a driver parameter that enables its use.
Because OverDrive features can damage your card, most workstation and server GPUs cannot use OverDrive.

Consumer GPUs that can use OverDrive must enable this feature by setting bit 14 in the amdgpu driver’s ppfeaturemask module parameter.

For OverDrive functionality, the OverDrive bit (bit 14) must be enabled (by default, the OverDrive bit is disabled on the ROCK and upstream kernels). This can be done by setting amdgpu.ppfeaturemask accordingly in the kernel parameters, or by changing the default value inside amdgpu_drv.c (if building your own kernel).

As an example, if the ppfeaturemask is set to 0xffffffff (11111111111111111111111111111111), then enabling the OverDrive bit would make it 0xffffffff (11111111111111111111111111111111). These are the flags that require OverDrive functionality to be enabled for the flag to work:

```bash
--showclkvolt
--showvoltagerange
--showvc
--showclkrange
--showmclkrange
--setlevel
--setmlevel
--setoverdrive
--setpoweroverdrive
--resetpoweroverdrive
--setvc
--setrange
--setmrange
```

Testing changes

After making changes to the SMI, run the test script to ensure that all functionality remains intact before uploading the patch. This can be done using:

```bash
./test-rocm-smi.sh /opt/rocm/bin/rocm-smi
```

The test can run all flags for the SMI, or specific flags can be tested with the -s option.

Any new functionality added to the SMI should have a corresponding test added to the test script.

Disclaimer

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Copyright (c) 2014-2017 Advanced Micro Devices, Inc. All rights reserved.
2.11.1 Programming ROCm-SMI

2.11.1.1 SYSFS Interface

2.11.2 Naming and data format standards for sysfs files

The libsensors library offers an interface to the raw sensors data through the sysfs interface. Since lm-sensors 3.0.0, libsensors is completely chip-independent. It assumes that all the kernel drivers implement the standard sysfs interface described in this document. This makes adding or updating support for any given chip very easy, as libsensors, and applications using it, do not need to be modified. This is a major improvement compared to lm-sensors 2.

Note that motherboards vary widely in the connections to sensor chips. There is no standard that ensures, for example, that the second temperature sensor is connected to the CPU, or that the second fan is on the CPU. Also, some values reported by the chips need some computation before they make full sense. For example, most chips can only measure voltages between 0 and +4V. Other voltages are scaled back into that range using external resistors. Since the values of these resistors can change from motherboard to motherboard, the conversions cannot be hard coded into the driver and have to be done in user space.

For this reason, even if we aim at a chip-independent libsensors, it will still require a configuration file (e.g. /etc/sensors.conf) for proper values conversion, labeling of inputs and hiding of unused inputs.

An alternative method that some programs use is to access the sysfs files directly. This document briefly describes the standards that the drivers follow, so that an application program can scan for entries and access this data in a simple and consistent way. That said, such programs will have to implement conversion, labeling and hiding of inputs. For this reason, it is still not recommended to bypass the library.

Each chip gets its own directory in the sysfs /sys/devices tree. To find all sensor chips, it is easier to follow the device symlinks from /sys/class/hwmon/hwmon*.

Up to lm-sensors 3.0.0, libsensors looks for hardware monitoring attributes in the “physical” device directory. Since lm-sensors 3.0.1, attributes found in the hwmon “class” device directory are also supported. Complex drivers (e.g. drivers for multifunction chips) may want to use this possibility to avoid namespace pollution. The only drawback will be that older versions of libsensors won’t support the driver in question.

All sysfs values are fixed point numbers.

There is only one value per file, unlike the older /proc specification. The common scheme for files naming is: <type><number>_<item>. Usual types for sensor chips are “in” (voltage), “temp” (temperature) and “fan” (fan). Usual items are “input” (measured value), “max” (high threshold, “min” (low threshold). Numbering usually starts from 1, except for voltages which start from 0 (because most data sheets use this). A number is always used for elements that can be present more than once, even if there is a single element of the given type on the specific chip. Other files do not refer to a specific element, so they have a simple name, and no number.

Alarms are direct indications read from the chips. The drivers do NOT make comparisons of readings to thresholds. This allows violations between readings to be caught and alarmed. The exact definition of an alarm (for example, whether a threshold must be met or must be exceeded to cause an alarm) is chip-dependent.

When setting values of hwmon sysfs attributes, the string representation of the desired value must be written, note that strings which are not a number are interpreted as 0! For more on how written strings are interpreted see the “sysfs attribute writes interpretation” section at the end of this file.

| [0-]* | denotes any positive number starting from 0 |
| [1-]* | denotes any positive number starting from 1 |
| RO | read only value |
| WO | write only value |
| RW | read/write value |
Read/write values may be read-only for some chips, depending on the hardware implementation.

All entries (except name) are optional, and should only be created in a given driver if the chip has the feature.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The chip name. This should be a short, lowercase string, not containing whitespace, dashes, or the wildcard character <code>*</code>. This attribute represents the chip name. It is the only mandatory attribute. I2C devices get this attribute created automatically. RO</td>
</tr>
<tr>
<td>update_interval</td>
<td>The interval at which the chip will update readings. Unit: millisecond RW Some devices have a variable update rate or interval. This attribute can be used to change it to the desired value.</td>
</tr>
</tbody>
</table>
### Voltage Input Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Unit</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in[0-9]_min</code></td>
<td>Voltage min value.</td>
<td>millivolt</td>
<td>RW</td>
</tr>
<tr>
<td><code>in[0-9]_crit</code></td>
<td>Voltage critical min value.</td>
<td>millivolt</td>
<td>RW</td>
</tr>
<tr>
<td><code>in[0-9]_max</code></td>
<td>Voltage max value.</td>
<td>millivolt</td>
<td>RW</td>
</tr>
<tr>
<td><code>in[0-9]_crit</code></td>
<td>Voltage critical max value.</td>
<td>millivolt</td>
<td>RW</td>
</tr>
<tr>
<td><code>in[0-9]_input</code></td>
<td>Voltage input value.</td>
<td>millivolt</td>
<td>RO</td>
</tr>
<tr>
<td><code>in[0-9]_average</code></td>
<td>Average voltage</td>
<td>millivolt</td>
<td>RO</td>
</tr>
<tr>
<td><code>in[0-9]_lowest</code></td>
<td>Historical minimum voltage</td>
<td>millivolt</td>
<td>RO</td>
</tr>
<tr>
<td><code>in[0-9]_highest</code></td>
<td>Historical maximum voltage</td>
<td>millivolt</td>
<td>RO</td>
</tr>
</tbody>
</table>

**Notes:**
- Values are generally NOT scaled by the chip driver and must be done by the application.
- Drivers should report the voltage values at the “pins” of the chip.
- Not all drivers (notably lm87 and via686a) do scale, because of internal resistors built into a chip. These drivers will output the actual voltage. Rule of thumb: drivers should report the voltage values at the “pins” of the chip.
Also see the Alarms section for status flags associated with voltages.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fan[1-]*_min</td>
<td>Fan minimum value&lt;br&gt;Unit: revolution/min (RPM)&lt;br&gt;RW</td>
</tr>
<tr>
<td>fan[1-]*_max</td>
<td>Fan maximum value&lt;br&gt;Unit: revolution/min (RPM)&lt;br&gt;Only rarely supported by the hardware.&lt;br&gt;RW</td>
</tr>
<tr>
<td>fan[1-]*_input</td>
<td>Fan input value.&lt;br&gt;Unit: revolution/min (RPM)&lt;br&gt;RO</td>
</tr>
<tr>
<td>fan[1-]*_div</td>
<td>Fan divisor.&lt;br&gt;Integer value in powers of two (1, 2, 4, 8, 16, 32, 64, 128).&lt;br&gt;RW&lt;br&gt;Some chips only support values 1, 2, 4 and 8.&lt;br&gt;Note that this is actually an internal clock divisor, which affects the measurable speed range, not the read value.</td>
</tr>
<tr>
<td>fan[1-]*_pulses</td>
<td>Number of tachometer pulses per fan revolution.&lt;br&gt;Integer value, typically between 1 and 4.&lt;br&gt;RW&lt;br&gt;This value is a characteristic of the fan connected to the device’s input, so it has to be set in accordance with the fan model. Should only be created if the chip has a register to configure the number of pulses. In the absence of such a register (and thus attribute) the value assumed by all devices is 2 pulses per fan revolution.</td>
</tr>
<tr>
<td>fan[1-]*_target</td>
<td>Desired fan speed&lt;br&gt;Unit: revolution/min (RPM)&lt;br&gt;RW&lt;br&gt;Only makes sense if the chip supports closed-loop fan speed control based on the measured fan speed.</td>
</tr>
<tr>
<td>fan[1-]*_label</td>
<td>Suggested fan channel label.&lt;br&gt;Text string&lt;br&gt;Should only be created if the driver has hints about what this fan channel is being used for, and user-space doesn’t. In all other cases, the label is provided by user-space.</td>
</tr>
</tbody>
</table>
Also see the Alarms section for status flags associated with fans.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm[1-]*</td>
<td>Pulse width modulation fan control. Integer value in the range 0 to 255</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>255 is max or 100%.</td>
</tr>
<tr>
<td>pwm[1-]*_enable</td>
<td>Fan speed control method:</td>
</tr>
<tr>
<td></td>
<td>0: no fan speed control (i.e. fan at full speed)</td>
</tr>
<tr>
<td></td>
<td>1: manual fan speed control enabled (using pwm[1-]*)</td>
</tr>
<tr>
<td></td>
<td>2+: automatic fan speed control enabled</td>
</tr>
<tr>
<td></td>
<td>Check individual chip documentation files for automatic mode details.</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>pwm[1-]*_mode</td>
<td>0: DC mode (direct current)</td>
</tr>
<tr>
<td></td>
<td>1: PWM mode (pulse-width modulation)</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>pwm[1-]*_freq</td>
<td>Base PWM frequency in Hz.</td>
</tr>
<tr>
<td></td>
<td>Only possibly available when pwmN_mode is PWM, but not always present even</td>
</tr>
<tr>
<td></td>
<td>then.</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>pwm[1-]*_auto_channels_temp</td>
<td>Select which temperature channels affect this PWM output in auto mode.</td>
</tr>
<tr>
<td></td>
<td>Bitfield, 1 is temp1, 2 is temp2, 4 is temp3 etc…</td>
</tr>
<tr>
<td></td>
<td>Which values are possible depend on the chip used.</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>pwm[1-]_auto_point[1-]_pwm</td>
<td>Define the PWM vs temperature curve. Number of trip points is chip-dependent.</td>
</tr>
<tr>
<td></td>
<td>Use this for chips which associate trip points to PWM output channels.</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>temp[1-]_auto_point[1-]_pwm</td>
<td>Define the PWM vs temperature curve. Number of trip points is chip dependent.</td>
</tr>
<tr>
<td></td>
<td>Use this for chips which associate trip points to temperature channels.</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
</tbody>
</table>
There is a third case where trip points are associated to both PWM output channels and temperature channels: the PWM values are associated to PWM output channels while the temperature values are associated to temperature channels. In that case, the result is determined by the mapping between temperature inputs and PWM outputs. When several temperature inputs are mapped to a given PWM output, this leads to several candidate PWM values. The actual result is up to the chip, but in general the highest candidate value (fastest fan speed) wins.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>temp[*]_type</code></td>
<td>Sensor type selection. Integers 1 to 6</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>1: CPU embedded diode</td>
</tr>
<tr>
<td></td>
<td>2: 3904 transistor</td>
</tr>
<tr>
<td></td>
<td>3: thermal diode</td>
</tr>
<tr>
<td></td>
<td>4: thermistor</td>
</tr>
<tr>
<td></td>
<td>5: AMD AMDSI</td>
</tr>
<tr>
<td></td>
<td>6: Intel PECI</td>
</tr>
<tr>
<td></td>
<td>Not all types are supported by all chips</td>
</tr>
<tr>
<td><code>temp[*]_max</code></td>
<td>Temperature max value.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius (or millivolt, see below)</td>
</tr>
<tr>
<td><code>temp[*]_min</code></td>
<td>Temperature min value.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius</td>
</tr>
<tr>
<td><code>temp[*]_max_hyst</code></td>
<td>Temperature hysteresis value for max limit.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius</td>
</tr>
<tr>
<td></td>
<td>Must be reported as an absolute temperature, NOT a delta from the max value.</td>
</tr>
<tr>
<td><code>temp[*]_min_hyst</code></td>
<td>Temperature hysteresis value for min limit.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius</td>
</tr>
<tr>
<td></td>
<td>Must be reported as an absolute temperature, NOT a delta from the min value.</td>
</tr>
<tr>
<td><code>temp[*]_input</code></td>
<td>Temperature input value.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius</td>
</tr>
<tr>
<td><code>temp[*]_crit</code></td>
<td>Temperature critical max value, typically greater than corresponding temp_max values.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius</td>
</tr>
<tr>
<td><code>temp[*]_crit_hyst</code></td>
<td>Temperature hysteresis value for critical limit.</td>
</tr>
<tr>
<td></td>
<td>Unit: millidegree Celsius</td>
</tr>
<tr>
<td></td>
<td>Must be reported as an absolute temperature, NOT a delta from the critical value.</td>
</tr>
</tbody>
</table>
Some chips measure temperature using external thermistors and an ADC, and report the temperature measurement as a voltage. Converting this voltage back to a temperature (or the other way around for limits) requires mathematical functions not available in the kernel, so the conversion must occur in user space. For these chips, all temp* files described above should contain values expressed in millivolt instead of millidegree Celsius. In other words, such temperature channels are handled as voltage channels by the driver.

Also see the Alarms section for status flags associated with temperatures.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>curr[1-*].max</td>
<td>Current max value</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].min</td>
<td>Current min value</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].lcrit</td>
<td>Current critical low value</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].crit</td>
<td>Current critical high value</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].input</td>
<td>Current input value</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].average</td>
<td>Average current use</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].lowest</td>
<td>Historical minimum current</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].highest</td>
<td>Historical maximum current</td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>Unit: milliampere</td>
<td></td>
</tr>
<tr>
<td>curr[1-*].reset_history</td>
<td>Reset currX.lowest and currX.highest</td>
<td>WO</td>
</tr>
<tr>
<td>curr.reset_history</td>
<td>Reset currX.lowest and currX.highest for all sensors</td>
<td>WO</td>
</tr>
<tr>
<td>curr[1-*].enable</td>
<td>Enable or disable the sensors</td>
<td></td>
</tr>
</tbody>
</table>
Also see the Alarms section for status flags associated with currents.
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>power[1-*]_average</td>
<td>Average power use</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_average_interval</td>
<td>Power use averaging interval. A poll notification is sent to this file if the hardware changes the averaging interval.</td>
</tr>
<tr>
<td></td>
<td>Unit: milliseconds</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_average_interval_max</td>
<td>Maximum power use averaging interval</td>
</tr>
<tr>
<td></td>
<td>Unit: milliseconds</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_average_interval_min</td>
<td>Minimum power use averaging interval</td>
</tr>
<tr>
<td></td>
<td>Unit: milliseconds</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_average_highest</td>
<td>Historical average maximum power use</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_average_lowest</td>
<td>Historical average minimum power use</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_average_max</td>
<td>A poll notification is sent to power[1-*]_average when power use rises above this value.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_average_min</td>
<td>A poll notification is sent to power[1-*]_average when power use sinks below this value.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_input</td>
<td>Instantaneous power use</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_input_highest</td>
<td>Historical maximum power use</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_input_lowest</td>
<td>Historical minimum power use</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_reset_history</td>
<td>Reset input_highest, input_lowest, average_highest and average_lowest.</td>
</tr>
<tr>
<td></td>
<td>WO</td>
</tr>
<tr>
<td>power[1-*]_accuracy</td>
<td>Accuracy of the power meter.</td>
</tr>
<tr>
<td></td>
<td>Unit: Percent</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_cap</td>
<td>If power use rises above this limit, the system should take action to reduce power use. A poll notification is sent to this file if the cap is changed by the hardware. The *_cap files only appear if the cap is known to be enforced by hardware.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_cap_hyst</td>
<td>Margin of hysteresis built around capping and notification.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_cap_max</td>
<td>Maximum cap that can be set.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_cap_min</td>
<td>Minimum cap that can be set.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>power[1-*]_max</td>
<td>Maximum power.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_crit</td>
<td>Critical maximum power. If power rises to or above this limit, the system is expected to take drastic action to reduce power consumption, such as a system shutdown or a forced powerdown of some devices.</td>
</tr>
<tr>
<td></td>
<td>Unit: microWatt</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td>power[1-*]_enable</td>
<td>Enable or disable the sensors.</td>
</tr>
<tr>
<td></td>
<td>When disabled the sensor read will return -ENODATA</td>
</tr>
<tr>
<td></td>
<td>1: Enable</td>
</tr>
<tr>
<td></td>
<td>0: Disable</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
</tbody>
</table>

**Chapter 2. Solid Compilation Foundation and Language Support**
Also see the Alarms section for status flags associated with power readings.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>energy[1-8]_input</td>
<td>Cumulative energy use</td>
</tr>
<tr>
<td></td>
<td>Unit: microJoule</td>
</tr>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td>energy[1-8]_enable</td>
<td>Enable or disable the sensors</td>
</tr>
<tr>
<td></td>
<td>When disabled the sensor read will return -ENODATA</td>
</tr>
<tr>
<td></td>
<td>1: Enable</td>
</tr>
<tr>
<td></td>
<td>0: Disable</td>
</tr>
<tr>
<td>humidity[1-8]_input</td>
<td>Humidity</td>
</tr>
<tr>
<td></td>
<td>Unit: milli-percent (per cent mille, pcm)</td>
</tr>
<tr>
<td>humidity[1-8]_enable</td>
<td>Enable or disable the sensors</td>
</tr>
<tr>
<td></td>
<td>When disabled the sensor read will return -ENODATA</td>
</tr>
<tr>
<td></td>
<td>1: Enable</td>
</tr>
<tr>
<td></td>
<td>0: Disable</td>
</tr>
</tbody>
</table>

Each channel or limit may have an associated alarm file, containing a boolean value. 1 means there is an alarm condition exists, 0 means no alarm.

Usually a given chip will either use channel-related alarms, or limit-related alarms, not both. The driver should just reflect the hardware implementation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in[0-8]_alarm</td>
<td>Channel alarm</td>
</tr>
<tr>
<td>curr[1-8]_alarm</td>
<td>0: no alarm</td>
</tr>
<tr>
<td>power[1-8]_alarm</td>
<td>1: alarm</td>
</tr>
<tr>
<td>fan[1-8]_alarm</td>
<td>RO</td>
</tr>
<tr>
<td>temp[1-8]_alarm</td>
<td></td>
</tr>
</tbody>
</table>

OR
Each input channel may have an associated fault file. This can be used to notify open diodes, unconnected fans etc. where the hardware supports it. When this boolean has value 1, the measurement for that channel should not be trusted.

Some chips also offer the possibility to get beeped when an alarm occurs:

```plaintext
beep_enable

Master beep enable
0: no beeps
1: beeps
RW

in[0-9]_beep
curr[1-9]_beep
fan[1-9]_beep
temp[1-9]_beep

Channel beep
0: disable
1: enable
RW
```
In theory, a chip could provide per-limit beep masking, but no such chip was seen so far.

Old drivers provided a different, non-standard interface to alarms and beeps. These interface files are deprecated, but will be kept around for compatibility reasons:

<table>
<thead>
<tr>
<th>alarms</th>
<th>Alarm bitmask.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RO</td>
</tr>
<tr>
<td></td>
<td>Integer representation of one to four bytes.</td>
</tr>
<tr>
<td></td>
<td>A ‘1’ bit means an alarm.</td>
</tr>
<tr>
<td></td>
<td>Chips should be programmed for ‘comparator’ mode so that</td>
</tr>
<tr>
<td></td>
<td>the alarm will ‘come back’ after you read the register if it is still valid.</td>
</tr>
<tr>
<td></td>
<td>Generally a direct representation of a chip’s internal alarm registers; there is no standard for the position of individual bits. For this reason, the use of this interface file for new drivers is discouraged. Use individual *.alarm and *.fault files instead.</td>
</tr>
<tr>
<td></td>
<td>Bits are defined in kernel/include/sensors.h.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>beep_mask</th>
<th>Bitmask for beep.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Same format as ‘alarms’ with the same bit locations, use discouraged for the same reason. Use individual *.beep files instead.</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>intrusion[0-]*.alarm</th>
<th>Chassis intrusion detection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0: OK</td>
</tr>
<tr>
<td></td>
<td>1: intrusion detected</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>Contrary to regular alarm flags which clear themselves automatically when read, this one sticks until cleared by the user. This is done by writing 0 to the file. Writing other values is unsupported.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>intrusion[0-]*.beep</th>
<th>Chassis intrusion beep</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0: disable</td>
</tr>
<tr>
<td></td>
<td>1: enable</td>
</tr>
<tr>
<td></td>
<td>RW</td>
</tr>
</tbody>
</table>
Devices allowing for reading \{in,power,curr,temp\}_average values may export attributes for controlling number of samples used to compute average.

Application software needs to understand the properties of the underlying hardware to leverage the performance capabilities of the platform for feature utilization and task scheduling. The sysfs topology exposes this information in a loosely hierarchal order. The information is populated by the KFD driver is gathered from ACPI (CRAT) and AMDGPU base driver.

The sysfs topology is arranged hierarchically as following. The root directory of the topology is

\[ \text{/sys/devices/virtual/kfd/kfd/topology/nodes/} \]

Based on the platform inside this directory there will be sub-directories corresponding to each HSA Agent. A system with N HSA Agents will have N directories as shown below.

\[
\begin{align*}
\text{/sys/devices/virtual/kfd/kfd/topology/nodes/0/} \\
\text{/sys/devices/virtual/kfd/kfd/topology/nodes/1/} \\
\text{.} \\
\text{.} \\
\text{/sys/devices/virtual/kfd/kfd/topology/nodes/N-1/}
\end{align*}
\]

### 2.11.3 HSA Agent Information

The HSA Agent directory and the sub-directories inside that contains all the information about that agent. The following are the main information available.

### 2.11.4 Node Information

This is available in the root directory of the HSA agent. This provides information about the compute capabilities of the agent which includes number of cores or compute units, SIMD count and clock speed.

### 2.11.5 Memory

The memory bank information attached to this agent is populated in “mem_banks” subdirectory.

\[ \text{/sys/devices/virtual/kfd/kfd/topology/nodes/N/mem_banks} \]

### 2.11.6 Cache

The caches available for this agent is populated in “cache” subdirectory

\[ \text{/sys/devices/virtual/kfd/kfd/topology/nodes/N/cache} \]
2.11.7 IO-LINKS

The IO links provides HSA agent interconnect information with latency (cost) between agents. This is useful for peer-to-peer transfers.

2.11.8 How to use topology information

The information provided in sysfs should not be directly used by application software. Application software should always use Thunk library API (libhsakmt) to access topology information. Please refer to Thunk API for more information.

The data are associated with a node ID, forming a per-node element list which references the elements contained at relative offsets within that list. A node associates with a kernel agent or agent. Node ID’s should be 0-based, with the “0” ID representing the primary elements of the system (e.g., “boot cores”, memory) if applicable. The enumeration order and—if applicable—values of the ID should match other information reported through mechanisms outside of the scope of the requirements;

For example, the data and enumeration order contained in the ACPI SRAT table on some systems should match the memory order and properties reported through HSA. Further detail is out of the scope of the System Architecture and outlined in the Runtime API specification.

![Topology sample](image)

Each of these nodes is interconnected with other nodes in more advanced systems to the level necessary to adequately describe the topology.
Where applicable, the node grouping of physical memory follows NUMA principles to leverage memory locality in software when multiple physical memory blocks are available in the system and agents have a different “access cost” (e.g., bandwidth/latency) to that memory.

**KFD Topology structure for AMDGPU :**

sysfsclasskfd
sysfsclasskfdtopology
sysfsclasskfdtopologynodes0
sysfsclasskfdtopologynodes0iolinks01
sysfsclasskfdtopologynodes0membanks0
sysfs-class-kfd-topology-nodes-N-caches

```
[–setsclk LEVEL [LEVEL … ]] [–setmclk LEVEL [LEVEL … ]] [–setpcie LEVEL [LEVEL … ]]
[–setslevel
```
2.11.8.1 SMI Event Interface and Library

An SMI event interface is added to the kernel and ROCm SMI lib for system administrators to get notified when specific events occur. On the kernel side, AMDKFD_IOC_SMI_EVENTS input/output control is added to allow notifications propagation to user mode through the event channel.

On the ROCm SMI lib side, APIs are added to set an event mask and receive event notifications with a timeout option. Further, ROCm SMI API details can be found in the PDF generated by Doxygen from source or by referring to the rocm_smi.h header file (see the rsmi_event_notification_* functions).

2.11.8.2 ROCm GPU/GCD Isolation

2.11.9 ROCR_VISIBLE_DEVICES

It is possible to rearrange or isolate the collection of ROCm GPU/GCD devices that are available on a ROCm platform. This can be achieved at the start of an application by way of ROCR_VISIBLE_DEVICES environment variable.

Devices to be made visible to an application should be specified as a comma-separated list of enumerable devices. For example, to use devices 0 and 2 from a ROCm platform with four devices, set ROCR_VISIBLE_DEVICES=0,2 before launching the application. The application will then enumerate these devices as device 0 and device 1, respectively.

This can be used by cooperating applications to effectively allocate GPU/GCDs among themselves.

2.11.10 Device cgroup

At a system administration level, the GPU/GCD isolation is possible using the device control group (cgroup). For all the AMD GPUs in a compute node, the ROck-Kernel-Driver exposes a single compute device file /dev/kfd and a separate (Direct Rendering Infrastructure) render device files /dev/dri/renderDN for each device. To participate in the Linux kernel’s cgroup infrastructure, the ROck driver relies on the render device files.

For example, consider a compute node with the two AMD GPUs. The ROck-Kernel-Driver exposes the following device files:

```
crw-rw-rw- 1 root root 240, 0 Apr 22 10:31 /dev/kfd

crw-rw—- 1 root video 226, 128 Apr 22 10:31 /dev/dri/renderD128

crw-rw—- 1 root video 226, 129 Apr 22 10:31 /dev/dri/renderD129
```

A ROCm application running on this compute node can use both GPUs only if it has access to all the above-listed device files. The administrator can restrict the devices an application can access by using device cgroup. The device cgroup subsystem allows or denies access to devices by applications in a cgroup. If a cgroup has whitelisted only /dev/kfd and /dev/dri/renderD129, then applications in that cgroup will have access only to that single GPU.

Refer to the Linux kernel’s cgroup documentation for information on how to create a cgroup and whitelist devices.

For cgroup-v1, refer: https://www.kernel.org/doc/Documentation/cgroup-v1/devices.txt

For cgroup-v2, refer: https://www.kernel.org/doc/Documentation/cgroup-v2.txt
2.12 ROCm Libraries

2.12.1 rocBLAS


A BLAS implementation on top of AMD’s Radeon Open Compute ROCm runtime and toolchains. rocBLAS is implemented in the HIP programming language and optimized for AMD’s latest discrete GPUs.

2.12.1.1 Prerequisites

- A ROCm enabled platform, more information here.
- Base software stack, which includes * HIP

2.12.1.2 Installing pre-built packages

Download pre-built packages either from ROCm’s package servers or by clicking the github releases tab and manually downloading, which could be newer. Release notes are available for each release on the releases tab.

```
sudo apt update && sudo apt install rocblas
```

2.12.1.3 Quickstart rocBLAS build

Bash helper build script (Ubuntu only)

The root of this repository has a helper bash script install.sh to build and install rocBLAS on Ubuntu with a single command. It does not take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it’s a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access, so it may prompt you for a password.

```
./install -h -- shows help
./install -id -- build library, build dependencies and install (-d flag only needs to be passed once on a system)
```

2.12.1.4 Manual build (all supported platforms)

If you use a distro other than Ubuntu, or would like more control over the build process, the rocblasiwiki has helpful information on how to configure cmake and manually build.

Functions supported

A list of exported functions. from rocblas can be found on the wiki.
2.12.1.5 rocBLAS interface examples

In general, the rocBLAS interface is compatible with CPU oriented Netlib BLAS and the cuBLAS-v2 API, with the explicit exception that traditional BLAS interfaces do not accept handles. The cuBLAS’ cublasHandle_t is replaced with rocblas_handle everywhere. Thus, porting a CUDA application which originally calls the cuBLAS API to a HIP application calling rocBLAS API should be relatively straightforward. For example, the rocBLAS SGEMV interface is

2.12.1.6 GEMV API

rocblas_status
rocblas_sgemv(rocblas_handle handle,
   rocblas_operation trans,
   rocblas_int m, rocblas_int n,
   const float* alpha,
   const float* A, rocblas_int lda,
   const float* x, rocblas_int incx,
   const float* beta,
   float* y, rocblas_int incy);

2.12.1.7 Batched and strided GEMM API

rocBLAS GEMM can process matrices in batches with regular strides. There are several permutations of these API’s, the following is an example that takes everything

rocblas_status
rocblas_sgemm_strided_batched(
   rocblas_handle handle,
   rocblas_operation transa, rocblas_operation transb,
   rocblas_int m, rocblas_int n, rocblas_int k,
   const float* alpha,
   const float* A, rocblas_int ls_a, rocblas_int ld_a, rocblas_int bs_a,
   const float* B, rocblas_int ls_b, rocblas_int ld_b, rocblas_int bs_b,
   const float* beta,
   float* C, rocblas_int ls_c, rocblas_int ld_c, rocblas_int bs_c,
   rocblas_int batch_count )

rocBLAS assumes matrices A and vectors x, y are allocated in GPU memory space filled with data. Users are responsible for copying data from/to the host and device memory. HIP provides memcpy style API’s to facilitate data management.

2.12.1.8 Asynchronous API

Except a few routines (like TRSM) having memory allocation inside preventing asynchronicity, most of the library routines (like BLAS-1 SCAL, BLAS-2 GEMV, BLAS-3 GEMM) are configured to operate in asynchronous fashion with respect to CPU, meaning these library functions return immediately.

For more information regarding rocBLAS library and corresponding API documentation, refer rocBLAS
2.12.1.9 API

This section provides details of the library API

2.12.1.9.1 Types

2.12.1.9.1.1 Definitions

2.12.1.9.1.2 rocblas_int

typedef int32_t rocblas_int
To specify whether int32 or int64 is used.

2.12.1.9.1.3 rocblas_stride

typedef int64_t rocblas_stride

2.12.1.9.1.4 rocblas_half

struct rocblas_half
Represents a 16 bit floating point number.

2.12.1.9.1.5 rocblas_handle

typedef struct _rocblas_handle *rocblas_handle
rocblas_handle is a structure holding the rocblas library context. It must be initialized using
rocblas_create_handle() and the returned handle must be passed to all subsequent library function calls. It
should be destroyed at the end using rocblas_destroy_handle().

2.12.1.9.1.6 Enums

Enumeration constants have numbering that is consistent with CBLAS, ACML and most standard C BLAS libraries.

2.12.1.9.1.7 rocblas_operation

enum rocblas_operation
Used to specify whether the matrix is to be transposed or not.

Values:

enumerator rocblas_operation_none = 111
Operate with the matrix.

enumerator rocblas_operation_transpose = 112
Operate with the transpose of the matrix.

enumerator rocblas_operation_conjugate_transpose = 113
Operate with the conjugate transpose of the matrix.
2.12.1.9.1.8 rocblas_fill

enum rocblas_fill
Used by the Hermitian, symmetric and triangular matrix routines to specify whether the upper or lower triangle is being referenced.

Values:

enumerator rocblas_fill_upper = 121
Upper triangle.

enumerator rocblas_fill_lower = 122
Lower triangle.

enumerator rocblas_fill_full = 123

2.12.1.9.1.9 rocblas_diagonal

enum rocblas_diagonal
It is used by the triangular matrix routines to specify whether the matrix is unit triangular.

Values:

enumerator rocblas_diagonal_non_unit = 131
Non-unit triangular.

enumerator rocblas_diagonal_unit = 132
Unit triangular.

2.12.1.9.1.10 rocblas_side

enum rocblas_side
Indicates the side matrix A is located relative to matrix B during multiplication.

Values:

enumerator rocblas_side_left = 141
Multiply general matrix by symmetric, Hermitian or triangular matrix on the left.

enumerator rocblas_side_right = 142
Multiply general matrix by symmetric, Hermitian or triangular matrix on the right.

enumerator rocblas_side_both = 143

2.12.1.9.1.11 rocblas_status

enum rocblas_status
rocblas status codes definition

Values:

enumerator rocblas_status_success = 0
success

enumerator rocblas_status_invalid_handle = 1
handle not initialized, invalid or null
enumerator rocblas_status_not_implemented = 2
    function is not implemented
enumerator rocblas_status_invalid_pointer = 3
    invalid pointer argument
enumerator rocblas_status_invalid_size = 4
    invalid size argument
enumerator rocblas_status_memory_error = 5
    failed internal memory allocation, copy or dealloc
enumerator rocblas_status_internal_error = 6
    other internal library failure
enumerator rocblas_status_perf_degraded = 7
    performance degraded due to low device memory
enumerator rocblas_status_size_query_mismatch = 8
    unmatched start/stop size query
enumerator rocblas_status_size_increased = 9
    queried device memory size increased
enumerator rocblas_status_size_unchanged = 10
    queried device memory size unchanged
enumerator rocblas_status_invalid_value = 11
    passed argument not valid
enumerator rocblas_status_continue = 12
    nothing preventing function to proceed

2.12.1.9.1.12 rocblas_datatype

enum rocblas_datatype
    Indicates the precision width of data stored in a blas type.
    Values:
    enumerator rocblas_datatype_f16_r = 150
        16 bit floating point, real
    enumerator rocblas_datatype_f32_r = 151
        32 bit floating point, real
    enumerator rocblas_datatype_f64_r = 152
        64 bit floating point, real
    enumerator rocblas_datatype_f16_c = 153
        16 bit floating point, complex
    enumerator rocblas_datatype_f32_c = 154
        32 bit floating point, complex
    enumerator rocblas_datatype_f64_c = 155
        64 bit floating point, complex
    enumerator rocblas_datatype_i8_r = 160
        8 bit signed integer, real
enumerator rocblas_datatype_u8_r = 161
  8 bit unsigned integer, real
enumerator rocblas_datatype_i32_r = 162
  32 bit signed integer, real
enumerator rocblas_datatype_u32_r = 163
  32 bit unsigned integer, real
enumerator rocblas_datatype_i8_c = 164
  8 bit signed integer, complex
enumerator rocblas_datatype_u8_c = 165
  8 bit unsigned integer, complex
enumerator rocblas_datatype_i32_c = 166
  32 bit signed integer, complex
enumerator rocblas_datatype_u32_c = 167
  32 bit unsigned integer, complex
enumerator rocblas_datatype_bf16_r = 168
  16 bit bfloat, real
enumerator rocblas_datatype_bf16_c = 169
  16 bit bfloat, complex

2.12.1.9.1.13 rocblas_pointer_mode

define rocblas_pointer_mode
  Indicates the pointer is device pointer or host pointer. This is typically used for scalars such as alpha and beta.

  Values:

  enumerator rocblas_pointer_mode_host = 0
    Scalar values affected by this variable will be located on the host.

  enumerator rocblas_pointer_mode_device = 1
    Scalar values affected by this variable will be located on the device.

2.12.1.9.1.14 rocblas_layer_mode

define rocblas_layer_mode
  Indicates if layer is active with bitmask.

  Values:

  enumerator rocblas_layer_mode_none = 0b0000000000
    No logging will take place.

  enumerator rocblas_layer_mode_log_trace = 0b0000000001
    A line containing the function name and value of arguments passed will be printed with each rocBLAS function call.

  enumerator rocblas_layer_mode_log_bench = 0b00000000010
    Outputs a line each time a rocBLAS function is called, this line can be used with rocblas-bench to make the same call again.
enumerator rocblas_layer_mode_log_profile = 0b0000000100
Outputs a YAML description of each rocBLAS function called, along with its arguments and number of times it was called.

2.12.1.9.1.15 rocblas_gemm_algo

denum rocblas_gemm_algo
Indicates if layer is active with bitmask.
Values:
enumerator rocblas_gemm_algo_standard = 0b0000000000

2.12.1.9.2 Functions

2.12.1.9.2.1 Level 1 BLAS

2.12.1.9.2.2 rocblas_<type>scal()

rocblas_status rocblas_dscal (rocblas_handle handle, rocblas_int n, const double *alpha, double *x, rocblas_int incx)
rocblas_status rocblas_sscal (rocblas_handle handle, rocblas_int n, const float *alpha, float *x, rocblas_int incx)

BLAS Level 1 API.
scal scales each element of vector x with scalar alpha.

\[ x := \alpha \times x \]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x.
- [in] alpha: device pointer or host pointer for the scalar alpha.
- [inout] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.

rocblas_status rocblas_cscal (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, rocblas_float_complex *x, rocblas_int incx)
rocblas_status rocblas_zscal (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, rocblas_double_complex *x, rocblas_int incx)
rocblas_status rocblas_csscal (rocblas_handle handle, rocblas_int n, const float *alpha, rocblas_float_complex *x, rocblas_int incx)
rocblas_status rocblas_zdscal (rocblas_handle handle, rocblas_int n, const double *alpha, rocblas_double_complex *x, rocblas_int incx)
2.12.1.9.2.3 rocblas_<type>_scal_batched()

rocblas_status rocblas_sscal_batched(rocblas_handle handle, rocblas_int n, const float *alpha, float *const x[], rocblas_int incx, rocblas_int batch_count)

BLAS Level 1 API.
scal_batched scales each element of vector x_i with scalar alpha, for i = 1, . . . , batch_count.

\[ x_i := \alpha \times x_i \]

where \((x_i)\) is the i-th instance of the batch.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each x_i.
- [in] alpha: host pointer or device pointer for the scalar alpha.
- [inout] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
- [in] batch_count: [rocblas_int] specifies the number of batches in x.

rocblas_status rocblas_dscal_batched(rocblas_handle handle, rocblas_int n, const double *alpha, double *const x[], rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_cscal_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_zscal_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_csscal_batched(rocblas_handle handle, rocblas_int n, const float *alpha, rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_zdscal_batched(rocblas_handle handle, rocblas_int n, const double *alpha, rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count)

2.12.1.9.2.4 rocblas_<type>_scal_strided_batched()

rocblas_status rocblas_sscal_strided_batched (rocblas_handle handle, rocblas_int n, const float *alpha, float *x, rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_cscal_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, rocblas_float_complex *x, rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_zscal_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, rocblas_double_complex *x, rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_csscal_strided_batched (rocblas_handle handle, rocblas_int n, const float *alpha, rocblas_float_complex *x, rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_zdscal_strided_batched (rocblas_handle handle, rocblas_int n, const double *alpha, rocblas_double_complex *x, rocblas_int incx, rocblas_int batch_count)

BLAS Level 1 API.
scał_strided_batched scales each element of vector x_i with scalar alpha, for i = 1, . . . , batch_count.

\[ x_i := \alpha \times x_i \]

where \((x_i)\) is the i-th instance of the batch.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] \( n \): [rocblas_int] the number of elements in each \( x_i \).
- [in] alpha: host pointer or device pointer for the scalar alpha.
- [inout] \( x \): device pointer to the first vector (\( x_1 \)) in the batch.
- [in] incx: [rocblas_int] specifies the increment for the elements of \( x \).
- [in] stride_\( x \): [rocblas_stride] stride from the start of one vector (\( x_i \)) and the next one (\( x_i+1 \)). There are no restrictions placed on stride_\( x \), however the user should take care to ensure that stride_\( x \) is of appropriate size, for a typical case this means stride_\( x \) \( \geq n \times incx \).
- [in] batch_count: [rocblas_int] specifies the number of batches in \( x \).

\[
\begin{align*}
\text{rocblas_status } \text{rocblas_dscal_strided_batched} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \text{ const double } *alpha, \text{ double } *x, \text{ rocblas_int } incx, \\
& \text{rocblas_stride } stride_\text{x}, \text{rocblas_int } batch\_\text{count}) \\
\text{rocblas_status } \text{rocblas_cscal_strided_batched} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \\
& \text{const rocblas_float_complex } *alpha, \text{ rocblas_float_complex } *x, \text{ rocblas_int } incx, \\
& \text{rocblas_stride } stride_\text{x}, \text{rocblas_int } batch\_\text{count}) \\
\text{rocblas_status } \text{rocblas_zscal_strided_batched} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \\
& \text{const rocblas_double_complex } *alpha, \text{ rocblas_double_complex } *x, \text{ rocblas_int } incx, \\
& \text{rocblas_stride } stride_\text{x}, \text{rocblas_int } batch\_\text{count}) \\
\text{rocblas_status } \text{rocblas_csscal_strided_batched} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \\
& \text{const float } *alpha, \text{ rocblas_float_complex } *x, \\
& \text{rocblas_int } incx, \text{ rocblas_stride } stride_\text{x}, \text{rocblas_int } batch\_\text{count}) \\
\text{rocblas_status } \text{rocblas_zdscal_strided_batched} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \\
& \text{const double } *alpha, \text{ rocblas_double_complex } *x, \\
& \text{rocblas_int } incx, \text{ rocblas_stride } stride_\text{x}, \text{rocblas_int } batch\_\text{count})
\end{align*}
\]

### 2.12.1.9.2.5 rocblas_<type>copy()

\[
\begin{align*}
\text{rocblas_status } \text{rocblas_dcopy} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \text{ const double } *x, \text{ rocblas_int } incx, \\
& \text{double } *y, \text{ rocblas_int } incy) \\
\text{rocblas_status } \text{rocblas_scopy} & (\text{rocblas_handle } handle, \text{ rocblas_int } n, \text{ const float } *x, \text{ rocblas_int } incx, \\
& \text{float } *y, \text{ rocblas_int } incy)
\end{align*}
\]

BLAS Level 1 API.

\[
\begin{align*}
y & := x,
\end{align*}
\]

**Parameters**

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] \( n \): [rocblas_int] the number of elements in \( x \) to be copied to \( y \).
- [in] \( x \): device pointer storing vector \( x \).
- [in] incx: [rocblas_int] specifies the increment for the elements of \( x \).
- [out] \( y \): device pointer storing vector \( y \).
• [in] incy: [rocblas_int] specifies the increment for the elements of y.

rocblas_status rocblas_ccopy (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy)

rocblas_status rocblas_zcopy (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy)

2.12.1.9.2.6 rocblas_<type>copy_batched()

rocblas_status rocblas_scopy_batched (rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, float *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_dcopy_batched (rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, double *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_ccopy_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_zcopy_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)

BLAS Level 1 API.

copy_batched copies each element \( x_i[j] \) into \( y_i[j] \), for \( j = 1, \ldots, n \); \( i = 1, \ldots, \text{batch\_count} \)

\[
y_i := x_i,
\]

where \((x_i, y_i)\) is the \(i\)-th instance of the batch. \(x_i\) and \(y_i\) are vectors.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each \(x_i\) to be copied to \(y_i\).
- [in] x: device array of device pointers storing each vector \(x_i\).
- [in] incx: [rocblas_int] specifies the increment for the elements of each vector \(x_i\).
- [out] y: device array of device pointers storing each vector \(y_i\).
- [in] incy: [rocblas_int] specifies the increment for the elements of each vector \(y_i\).
- [in] batch_count: [rocblas_int] number of instances in the batch
2.12.1.9.2.7 rocblas_<type>copy_strided_batched()

rocblas_status rocblas_sccopy_strided_batched(rocblas_handle handle, rocblas_int n, const
float *x, rocblas_int incx, rocblas_stride stridex,
float *y, rocblas_int incy, rocblas_stride stridey,
rocblas_int batch_count)

BLAS Level 1 API.

copy_strided_batched copies each element x_i[j] into y_i[j], for j = 1, ..., n; i = 1, ..., batch_count

\[ y_i := x_i, \]

where \((x_i, y_i)\) is the i-th instance of the batch. \(x_i\) and \(y_i\) are vectors.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each \(x_i\) to be copied to \(y_i\).
- [in] x: device pointer to the first vector (\(x_1\)) in the batch.
- [in] incx: [rocblas_int] specifies the increments for the elements of vectors \(x_i\).
- [in] stridex: [rocblas_stride] stride from the start of one vector (\(x_i\)) and the next one (\(x_{i+1}\)).
  There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x
  is of appropriate size, for a typical case this means stride_x >= n * incx.
- [out] y: device pointer to the first vector (\(y_1\)) in the batch.
- [in] incy: [rocblas_int] specifies the increment for the elements of vectors \(y_i\).
- [in] stridey: [rocblas_stride] stride from the start of one vector (\(y_i\)) and the next one (\(y_{i+1}\)).
  There are no restrictions placed on stride_y, however the user should take care to ensure that stride_y
  is of appropriate size, for a typical case this means stride_y >= n * incy. stridey should be non zero.
- [in] incy: [rocblas_int] specifies the increment for the elements of \(y\).
- [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dcopy_strided_batched (rocblas_handle handle, rocblas_int n, const
double *x, rocblas_int incx, rocblas_stride stridex,
double *y, rocblas_int incy, rocblas_stride stridey,
rocblas_int batch_count)

rocblas_status rocblas_ccopy_strided_batched (rocblas_handle handle, rocblas_int n, const
rocblas_float_complex *x, rocblas_int incx,
rocblas_stride stridex, rocblas_float_complex *
y, rocblas_int incy, rocblas_stride stridey,
rocblas_int batch_count)

rocblas_status rocblas_zcopy_strided_batched (rocblas_handle handle, rocblas_int n, const
rocblas_double_complex *x, rocblas_int incx,
rocblas_stride stridex, rocblas_double_complex *
y, rocblas_int incy, rocblas_stride stridey,
rocblas_int batch_count)
2.12.1.9.2.8 rocblas_<type>dot()

rocblas_status rocblas_ddot (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, 
const double *y, rocblas_int incy, double *result)

rocblas_status rocblas_sdot (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, 
const float *y, rocblas_int incy, float *result)

BLAS Level 1 API.
dot(u) performs the dot product of vectors x and y

\[
\text{result} = x \cdot y;
\]
dotc performs the dot product of the conjugate of complex vector x and complex vector y

\[
\text{result} = \text{conjugate} (x) \cdot y;
\]

Parameters
- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x and y.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of y.
- [in] y: device pointer storing vector y.
- [in] incy: [rocblas_int] specifies the increment for the elements of y.
- [inout] result: device pointer or host pointer to store the dot product. return is 0.0 if \( n \leq 0 \).

rocblas_status rocblas_hdot (rocblas_handle handle, rocblas_int n, const rocblas_half *x, rocblas_int 
incx, const rocblas_half *y, rocblas_int incy, rocblas_half *result)

rocblas_status rocblas_bfddot (rocblas_handle handle, rocblas_int n, const rocblas_bfloat16 *x, 
rocblas_int incx, const rocblas_bfloat16 *y, rocblas_int incy, rocblas_bfloat16 *result)

rocblas_status rocblas_cdotu (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, 
rocblas_int incx, const rocblas_float_complex *y, rocblas_int incy, rocblas_float_complex *result)

rocblas_status rocblas_cdotc (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, 
rocblas_int incx, const rocblas_float_complex *y, rocblas_int incy, rocblas_float_complex *result)

rocblas_status rocblas_zdotu (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, 
rocblas_int incx, const rocblas_double_complex *y, rocblas_int incy, rocblas_double_complex *result)

rocblas_status rocblas_zdotc (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, 
rocblas_int incx, const rocblas_double_complex *y, rocblas_int incy, rocblas_double_complex *result)
2.12.1.9.2.9 rocblas\_<type>dot\_batched()

rocblas\_status rocblas\_sdot\_batched (rocblas\_handle handle, rocblas\_int n, const float *const x[],
rocblas\_int incx, const float *const y[], rocblas\_int incy,
rocblas\_int batch\_count, float *result)

BLAS Level 1 API.

dot\_batched(u) performs a batch of dot products of vectors x and y

\[
\text{result}_i = x_i \times y_i;
\]

dotc\_batched performs a batch of dot products of the conjugate of complex vector x and complex vector y

\[
\text{result}_i = \overline{x_i} \times y_i;
\]

where \((x_i, y_i)\) is the i-th instance of the batch. \(x_i\) and \(y_i\) are vectors, for \(i = 1, \ldots, \text{batch\_count}\)

Parameters

- [in] handle: [rocblas\_handle] handle to the rocblas library context queue.
- [in] n: [rocblas\_int] the number of elements in each \(x_i\) and \(y_i\).
- [in] x: device array of device pointers storing each vector \(x_i\).
- [in] incx: [rocblas\_int] specifies the increment for the elements of each \(x_i\).
- [in] y: device array of device pointers storing each vector \(y_i\).
- [in] incy: [rocblas\_int] specifies the increment for the elements of each \(y_i\).
- [in] batch\_count: [rocblas\_int] number of instances in the batch
- [inout] result: device array or host array of batch\_count size to store the dot products of each batch. return 0.0 for each element if \(n \leq 0\).

rocblas\_status rocblas\_ddot\_batched (rocblas\_handle handle, rocblas\_int n, const double *const x[],
rocblas\_int incx, const double *const y[], rocblas\_int incy,
rocblas\_int batch\_count, double *result)

rocblas\_status rocblas\_hdot\_batched (rocblas\_handle handle, rocblas\_int n, const rocblas\_half
*const x[], rocblas\_int incx, const rocblas\_half *const y[],
rocblas\_int incy, rocblas\_int batch\_count, rocblas\_half *result)

rocblas\_status rocblas\_bfdot\_batched (rocblas\_handle handle, rocblas\_int n, const rocblas\_bfloat16
*const x[], rocblas\_int incx, const rocblas\_bfloat16 *const y[],
rocblas\_int incy, rocblas\_int batch\_count, rocblas\_bfloat16 *result)

rocblas\_status rocblas\_cdotu\_batched (rocblas\_handle handle, rocblas\_int n, const
rocblas\_float\_complex *const x[], rocblas\_int incx,
const rocblas\_float\_complex *const y[], rocblas\_int incy,
rocblas\_int batch\_count, rocblas\_float\_complex *result)

rocblas\_status rocblas\_cdotc\_batched (rocblas\_handle handle, rocblas\_int n, const
rocblas\_float\_complex *const x[], rocblas\_int incx,
const rocblas\_float\_complex *const y[], rocblas\_int incy,
rocblas\_int batch\_count, rocblas\_float\_complex *result)
rocblas_status rocblas_zdotu_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count, rocblas_double_complex *result)

rocblas_status rocblas_zdotc_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count, rocblas_double_complex *result)

2.12.1.9.2.10 rocblas_<type>dot_strided_batched()

rocblas_status rocblas_sdot_strided_batched (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, const float *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, float *result)

rocblas_status rocblas_ddot_strided_batched (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, const double *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, double *result)

BLAS Level 1 API.

dot_strided_batched(u) performs a batch of dot products of vectors x and y

result_i = x_i * y_i;


dotc_strided_batched performs a batch of dot products of the conjugate of complex vector x and complex vector y

result_i = conjugate (x_i) * y_i;

where (x_i, y_i) is the i-th instance of the batch. x_i and y_i are vectors, for i = 1, ..., batch_count

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] the number of elements in each x_i and y_i.
• [in] x: device pointer to the first vector (x_1) in the batch.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
• [in] stridex: [rocblas_stride] stride from the start of one vector (x_i) and the next one (x_i+1)
• [in] y: device pointer to the first vector (y_1) in the batch.
• [in] incy: [rocblas_int] specifies the increment for the elements of each y_i.
• [in] stridey: [rocblas_stride] stride from the start of one vector (y_i) and the next one (y_i+1)
• [in] batch_count: [rocblas_int] number of instances in the batch
• [inout] result: device array or host array of batch_count size to store the dot products of each batch. return 0.0 for each element if n <= 0.
rocblas_status rocblas_hdot_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_half *x, rocblas_int incx, rocblas_stride stridex, const rocblas_half *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_half *result)

rocblas_status rocblas_bfdot_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_bfloat16 *x, rocblas_int incx, rocblas_stride stridex, const rocblas_bfloat16 *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_bfloat16 *result)

rocblas_status rocblas_cdotu_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_float_complex *result)

rocblas_status rocblas_cdotc_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_float_complex *result)

rocblas_status rocblas_zdotu_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_double_complex *result)

rocblas_status rocblas_zdotc_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_double_complex *result)

2.12.1.9.2.11 rocblas_<type>swap()

rocblas_status rocblas_sswap (rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, float *y, rocblas_int incy)

BLAS Level 1 API.

swap interchanges vectors x and y.

\[
y := x; \quad x := y
\]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x and y.
- [inout] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
• [inout] y: device pointer storing vector y.
• [in] incy: [rocblas_int] specifies the increment for the elements of y.

rocblas_status rocblas_dswap (rocblas_handle handle, rocblas_int n, double *x, rocblas_int inx, double *y, rocblas_int incy)

rocblas_status rocblas_cswap (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int inx, rocblas_float_complex *y, rocblas_int incy)

rocblas_status rocblas_zswap (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int inx, rocblas_double_complex *y, rocblas_int incy)

2.12.1.9.2.12 rocblas_<type>swap_batched()

rocblas_status rocblas_sswap_batched (rocblas_handle handle, rocblas_int n, float *x[], rocblas_int inx, float *y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_dswap_batched (rocblas_handle handle, rocblas_int n, double *x[], rocblas_int inx, double *y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_cswap_batched (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x[], rocblas_int inx, rocblas_float_complex *y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_zswap_batched (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x[], rocblas_int inx, rocblas_double_complex *y[], rocblas_int incy, rocblas_int batch_count)

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] the number of elements in each x_i and y_i.
• [inout] x: device array of device pointers storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
• [inout] y: device array of device pointers storing each vector y_i.
• [in] incy: [rocblas_int] specifies the increment for the elements of each y_i.
• [in] batch_count: [rocblas_int] number of instances in the batch.
2.12.1.9.2.13 rocblas_<type>swap_strided_batched()

rocblas_status rocblas_sswap_strided_batched(rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, rocblas_stride stridex, float *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

BLAS Level 1 API.

swap_strided_batched interchanges vectors \( x_i \) and \( y_i \), for \( i = 1, \ldots, \text{batch\_count} \)

\[
y_i := x_i; x_i := y_i
\]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each \( x_i \) and \( y_i \).
- [inout] x: device pointer to the first vector \( x_1 \).
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
- [in] stridex: [rocblas_stride] stride from the start of one vector (\( x_i \)) and the next one (\( x_{i+1} \)). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size, for a typical case this means \( \text{stride}_x >= n \times \text{incx} \).
- [inout] y: device pointer to the first vector \( y_1 \).
- [in] incy: [rocblas_int] specifies the increment for the elements of y.
- [in] stridey: [rocblas_stride] stride from the start of one vector (\( y_i \)) and the next one (\( y_{i+1} \)). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_y is of appropriate size, for a typical case this means \( \text{stride}_y >= n \times \text{incy} \). stridey should be non zero.
- [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_dswap_strided_batched (rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, rocblas_stride stridex, double *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_cswap_strided_batched (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_zswap_strided_batched (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
2.12.1.9.2.14 rocblas<_type>axpy()

rocblas_status rocblas_daxpy(rocblas_handle handle, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, double *y, rocblas_int incy)
rocblas_status rocblas_saxpy(rocblas_handle handle, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, float *y, rocblas_int incy)

BLAS Level 1 API.

axpy computes constant alpha multiplied by vector x, plus vector y

\[ y := alpha \times x + y \]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x and y.
- [in] alpha: device pointer or host pointer to specify the scalar alpha.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
- [out] y: device pointer storing vector y.
- [inout] incy: [rocblas_int] specifies the increment for the elements of y.

rocblas_status rocblas_haxpy(rocblas_handle handle, rocblas_int n, const rocblas_half *alpha, const rocblas_half *x, rocblas_int incx, rocblas_half *y, rocblas_int incy)
rocblas_status rocblas_caxpy(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy)
rocblas_status rocblas_zaxpy(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy)

2.12.1.9.2.15 rocblas<_type>asum()

rocblas_status rocblas_dasum(rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, double *result)
rocblas_status rocblas_sasum(rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, float *result)

BLAS Level 1 API.

asum computes the sum of the magnitudes of elements of a real vector x, or the sum of magnitudes of the real and imaginary parts of elements if x is a complex vector

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x and y.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x. incx must be > 0.
- [in] result: device pointer or host pointer to store the asum product. return is 0.0 if n <= 0.
rocblas_status rocblas_scasum(const rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, float *result)
rocblas_status rocblas_dzasum(const rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, double *result)

2.12.1.9.2.16 rocblas_<type>asum_batched()

rocblas_status rocblas_sasum_batched(const rocblas_handle handle, rocblas_int n, const float *x[], rocblas_int incx, rocblas_int batch_count, float *results)
rocblas_status rocblas_dasum_batched(const rocblas_handle handle, rocblas_int n, const double *x[], rocblas_int incx, rocblas_int batch_count, double *results)
rocblas_status rocblas_scasum_batched(const rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x[], rocblas_int incx, rocblas_int batch_count, float *results)
rocblas_status rocblas_dzasum_batched(const rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x[], rocblas_int incx, rocblas_int batch_count, double *results)

2.12.1.9.2.17 rocblas_<type>asum_strided_batched()

rocblas_status rocblas_sasum_strided_batched(const rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)
rocblas_status rocblas_dasum_strided_batched(const rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

Parameters

- **handle**: [rocblas_handle] handle to the rocblas library context queue.
- **n**: [rocblas_int] number of elements in each vector x_i.
- **x**: device array of device pointers storing each vector x_i.
- **incx**: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- **results**: device array or host array of batch_count size for results. return is 0.0 if n, incx<=0.
- **batch_count**: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_sasum_strided_batched(const rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)
rocblas_status rocblas_dasum_strided_batched(const rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

Parameters

- **handle**: [rocblas_handle] handle to the rocblas library context queue.
- **n**: [rocblas_int] number of elements in each vector x_i.
- **x**: device pointer to the first vector x_1.
- **incx**: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• **[in]** `stridex` ([rocblas_stride] stride from the start of one vector \((x_i)\) and the next one \((x_{i+1})\). There are no restrictions placed on `stride_x`, however the user should take care to ensure that `stride_x` is of appropriate size, for a typical case this means `stride_x >= n \times incx`.

• **[out]** `results` : device pointer or host pointer to array for storing contiguous `batch_count` results. return is 0.0 if `n`, `incx`<=0.

• **[in]** `batch_count` : [rocblas_int] number of instances in the batch

```c
rocblas_status rocblas_dasum_strided_batched (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)
rocblas_status rocblas_scasum_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)
rocblas_status rocblas_dzasum_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)
```

### 2.12.1.9.2.18 rocblas_<type>nrm2()

```c
rocblas_status rocblas_dnrm2 (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, double *result)
rocblas_status rocblas_snrm2 (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, float *result)
```

**nrm2** computes the euclidean norm of a real or complex vector

```
result := sqrt( x'*x ) for real vectors
result := sqrt( x**H*x ) for complex vectors
```

**Parameters**

- **[in]** `handle` : [rocblas_handle] handle to the rocblas library context queue.
- **[in]** `n` : [rocblas_int] the number of elements in `x`.
- **[in]** `x` : device pointer storing vector `x`.
- **[in]** `incx` : [rocblas_int] specifies the increment for the elements of `y`.
- **[in/out]** `result` : device pointer or host pointer to store the `nrm2` product. return is 0.0 if `n`, `incx`<=0.

```c
rocblas_status rocblas_scnrm2 (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, float *result)
rocblas_status rocblas_dznrm2 (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, double *result)
```
2.12.1.9.2.19 rocblas_<type>nrm2_batched()

rocblas_status rocblas_snrm2_batched(rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, rocblas_int batch_count, float *results)

BLAS Level 1 API.

nrm2_batched computes the euclidean norm over a batch of real or complex vectors

\[
\text{result} := \sqrt{x_i'x_i} \quad \text{for real vectors } x, \text{ for } i = 1, \ldots, \text{batch_count}
\]

\[
\text{result} := \sqrt{x_i^*x_i} \quad \text{for complex vectors } x, \text{ for } i = 1, \ldots, \text{batch_count}
\]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each x_i.
- [in] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- [in] batch_count: [rocblas_int] number of instances in the batch
- [out] results: device pointer or host pointer to array of batch_count size for nrm2 results. return is 0.0 for each element if n <= 0, incx<=0.

rocblas_status rocblas_dnrm2_batched(rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, rocblas_int batch_count, double *results)

rocblas_status rocblas_scnrm2_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count, float *results)

rocblas_status rocblas_dznrm2_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count, double *results)

2.12.1.9.2.20 rocblas_<type>nrm2_strided_batched()

rocblas_status rocblas_snrm2_strided_batched(rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)

BLAS Level 1 API.
nrm2_strided_batched computes the euclidean norm over a batch of real or complex vectors

\[
\text{result} := \sqrt{x_i'x_i} \quad \text{for real vectors } x, \text{ for } i = 1, \ldots, \text{batch_count}
\]

\[
\text{result} := \sqrt{x_i^*x_i} \quad \text{for complex vectors, } \text{for } i = 1, \ldots, \text{batch_count}
\]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each x_i.
- [in] x: device pointer to the first vector x_1.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• [in] stridex: [rocblas_stride] stride from the start of one vector (x_i) and the next one (x_{i+1}). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size, for a typical case this means stride_x \geq n \times incx.

• [in] batch_count: [rocblas_int] number of instances in the batch

• [out] results: device pointer or host pointer to array for storing contiguous batch_count results. return is 0.0 for each element if n \leq 0, incx \leq 0.

rocblas_status rocblas_dnrm2_strided_batched (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

rocblas_status rocblas_scnrm2_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)

rocblas_status rocblas_dznrm2_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

2.12.1.9.2.21 rocblas_i<type>amax()

rocblas_status rocblas_idamax (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_int *result)

rocblas_status rocblas_isamax (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_int *result)

BLAS Level 1 API.

amax finds the first index of the element of maximum magnitude of a vector x. vector

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.

• [in] n: [rocblas_int] the number of elements in x.

• [in] x: device pointer storing vector x.

• [in] incx: [rocblas_int] specifies the increment for the elements of y.

• [inout] result: device pointer or host pointer to store the amax index. return is 0.0 if n, incx \leq 0.

rocblas_status rocblas_icamax (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_int *result)

rocblas_status rocblas_izamax (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_int *result)
2.12.1.9.2.22 rocblas_i<type>amax_batched()

rocblas_status rocblas_isamax_batched(rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.

amax_batched finds the first index of the element of maximum magnitude of each vector x_i in a batch, for i = 1, ..., batch_count.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in each vector x_i
• [in] x: device array of device pointers storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• [in] batch_count: [rocblas_int] number of instances in the batch, must be > 0.
• [out] result: device or host array of pointers of batch_count size for results. return is 0 if n, incx<=0.

rocblas_status rocblas_idamax_batched(rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_icamax_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_izamax_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

2.12.1.9.2.23 rocblas_i<type>amax_strided_batched()

rocblas_status rocblas_isamax_strided_batched(rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.

amax_strided_batched finds the first index of the element of maximum magnitude of each vector x_i in a batch, for i = 1, ..., batch_count.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in each vector x_i
• [in] x: device pointer to the first vector x_1.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• [in] stridex: [rocblas_stride] specifies the pointer increment between one x_i and the next x_(i + 1).
• [in] batch_count: [rocblas_int] number of instances in the batch
rocblas_status rocblas_idamax_strided_batched(rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_icamax_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_izamax_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

2.12.1.9.2.24 rocblas_i<type>amin()

rocblas_status rocblas_idamin (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_int *result)

rocblas_status rocblas_isamin (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_int *result)

BLAS Level 1 API.

amin finds the first index of the element of minimum magnitude of a vector x.

vector

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] the number of elements in x.
• [in] x: device pointer storing vector x.
• [in] incx: [rocblas_int] specifies the increment for the elements of y.
• [inout] result: device pointer or host pointer to store the amin index. return is 0.0 if n, incx<=0.

rocblas_status rocblas_icamin (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_int *result)

rocblas_status rocblas_izamin (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_int *result)

2.12.1.9.2.25 rocblas_i<type>amin_batched()
• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in each vector x_i
• [in] x: device array of device pointers storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• [in] batch_count: [rocblas_int] number of instances in the batch, must be > 0.
• [out] result: device or host pointers to array of batch_count size for results. return is 0 if n, incx<=0.

rocblas_status rocblas_idamin_batched(rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_icamin_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_izamin_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

2.12.1.9.2.26 rocblas_i<type>amin_strided_batched()

rocblas_status rocblas_isamin_strided_batched(rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.

amin_strided_batched finds the first index of the element of minimum magnitude of each vector x_i in a batch, for i = 1, . . . , batch_count.

Parameters
• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in each vector x_i
• [in] x: device pointer to the first vector x_1.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• [in] stridex: [rocblas_stride] specifies the pointer increment between one x_i and the next x_(i + 1)
• [in] batch_count: [rocblas_int] number of instances in the batch
• [out] result: device or host pointer to array for storing contiguous batch_count results. return is 0 if n <= 0, incx<=0.

rocblas_status rocblas_idamin_strided_batched(rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_icamin_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_izamin_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

### 2.12.1.9.2.27 rocblas_\<type\>_rotate()

rocblas_status rocblas_srot (rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, float *y, rocblas_int incy, const float *c, const float *s)

<table>
<thead>
<tr>
<th>BLAS Level 1 API</th>
</tr>
</thead>
<tbody>
<tr>
<td>rot applies the Givens rotation matrix defined by c=cos(alpha) and s=sin(alpha) to vectors x and y. Scalars c and s may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.</td>
</tr>
</tbody>
</table>

**Parameters**

- **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.
- **[in] n**: [rocblas_int] number of elements in the x and y vectors.
- **[inout] x**: device pointer storing vector x.
- **[in] incx**: [rocblas_int] specifies the increment between elements of x.
- **[inout] y**: device pointer storing vector y.
- **[in] incy**: [rocblas_int] specifies the increment between elements of y.
- **[in] c**: device pointer or host pointer storing scalar cosine component of the rotation matrix.
- **[in] s**: device pointer or host pointer storing scalar sine component of the rotation matrix.

rocblas_status rocblas_drot (rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, double *y, rocblas_int incy, const double *c, const double *s)

rocblas_status rocblas_crot (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy, const float *c, const rocblas_float_complex *s)

rocblas_status rocblas_csrot (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy, const float *c, const rocblas_float_complex *s)

rocblas_status rocblas_zrot (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy, const double *c, const rocblas_double_complex *s)

rocblas_status rocblas_zdrot (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy, const double *c, const double *s)
2.12.1.9.2.28 rocblas_<type>_rot_batched()

rocblas_status rocblas_srot_batched(rocblas_handle handle, rocblas_int n, float *const x[], rocblas_int incx, float *const y[], rocblas_int incy, const float *c, const float *s, rocblas_int batch_count)

BLAS Level 1 API.

rocblas_srot_batched applies the Givens rotation matrix defined by c=cos(alpha) and s=sin(alpha) to batched vectors x_i and y_i, for i = 1, ..., batch_count. Scalars c and s may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each x_i and y_i vectors.
- [inout] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment between elements of each x_i.
- [inout] y: device array of device pointers storing each vector y_i.
- [in] incy: [rocblas_int] specifies the increment between elements of each y_i.
- [in] c: device pointer or host pointer to scalar cosine component of the rotation matrix.
- [in] s: device pointer or host pointer to scalar sine component of the rotation matrix.
- [in] batch_count: [rocblas_int] the number of x and y arrays, i.e. the number of batches.

rocblas_status rocblas_drot_batched(rocblas_handle handle, rocblas_int n, double *const x[], rocblas_int incx, double *const y[], rocblas_int incy, const double *c, const double *s, rocblas_int batch_count)

rocblas_status rocblas_crot_batched(rocblas_handle handle, rocblas_int n, rocblas_float_complex *const x[], rocblas_int incx, rocblas_float_complex *const y[], rocblas_int incy, const float *c, const rocblas_float_complex *s, rocblas_int batch_count)

rocblas_status rocblas_zrot_batched(rocblas_handle handle, rocblas_int n, rocblas_double_complex *const x[], rocblas_int incx, rocblas_double_complex *const y[], rocblas_int incy, const float *c, const rocblas_double_complex *s, rocblas_int batch_count)

rocblas_status rocblas_zdrot_batched(rocblas_handle handle, rocblas_int n, rocblas_double_complex *const x[], rocblas_int incx, rocblas_double_complex *const y[], rocblas_int incy, const double *c, const double *s, rocblas_int batch_count)
### 2.12.1.9.2.29 rocblas\_<type>\_rot\_strided\_batched()

**rocblas_status rocblas_srot_strided_batched**

```c
rocblas_status rocblas_srot_strided_batched(rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, rocblas_stride stride_x, float *y, rocblas_int incy, rocblas_stride stride_y, const float *c, const float *s, rocblas_int batch_count)
```

**rocblas_status rocblas_drot_strided_batched**

```c
rocblas_status rocblas_drot_strided_batched(rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, rocblas_stride stride_x, double *y, rocblas_int incy, rocblas_stride stride_y, const double *c, const double *s, rocblas_int batch_count)
```

**rocblas_status rocblas_crot_strided_batched**

```c
rocblas_status rocblas_crot_strided_batched(rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stride_y, const float *c, const rocblas_float_complex *s, rocblas_int batch_count)
```

**rocblas_status rocblas_csrot_strided_batched**

```c
rocblas_status rocblas_csrot_strided_batched(rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stride_y, const float *c, const float *s, rocblas_int batch_count)
```

**BLAS Level 1 API.**

`rocblas\_<type>\_rot\_strided\_batched` applies the Givens rotation matrix defined by \( c=\cos(\alpha) \) and \( s=\sin(\alpha) \) to strided batched vectors \( x_i \) and \( y_i \), for \( i = 1, \ldots, \text{batch\_count} \). Scalars \( c \) and \( s \) may be stored in either host or device memory, location is specified by calling `rocblas_set_pointer_mode`.

**Parameters**

- **[in]** `handle`: [rocblas\_handle] handle to the rocblas library context queue.
- **[in]** `n`: [rocblas\_int] number of elements in each \( x_i \) and \( y_i \) vectors.
- **[inout]** `x`: device pointer to the first vector \( x_1 \).
- **[in]** `incx`: [rocblas\_int] specifies the increment between elements of each \( x_i \).
- **[in]** `stride_x`: [rocblas\_stride] specifies the increment from the beginning of \( x_i \) to the beginning of \( x_{i+1} \).
- **[inout]** `y`: device pointer to the first vector \( y_1 \).
- **[in]** `incy`: [rocblas\_int] specifies the increment between elements of each \( y_i \).
- **[in]** `stride_y`: [rocblas\_stride] specifies the increment from the beginning of \( y_i \) to the beginning of \( y_{i+1} \).
- **[in]** `c`: device pointer or host pointer to scalar cosine component of the rotation matrix.
- **[in]** `s`: device pointer or host pointer to scalar sine component of the rotation matrix.
- **[in]** `batch\_count`: [rocblas\_int] the number of \( x \) and \( y \) arrays, i.e. the number of batches.
rocblas_status rocblas_zrot_strided_batched(rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stride_y, const double *c, const rocblas_double_complex *s, rocblas_int batch_count)

rocblas_status rocblas_zdrot_strided_batched(rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stride_y, const double *c, const double *s, rocblas_int batch_count)

2.12.1.9.2.30 rocblas_<type>rotg()

rocblas_status rocblas_srotg (rocblas_handle handle, float *a, float *b, float *c, float *s)

BLAS Level 1 API.

rocrot creates the Givens rotation matrix for the vector (a b). Scalars c and s and arrays a and b may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [inout] a: device pointer or host pointer to input vector element, overwritten with r.
• [inout] b: device pointer or host pointer to input vector element, overwritten with z.
• [inout] c: device pointer or host pointer to cosine element of Givens rotation.
• [inout] s: device pointer or host pointer sine element of Givens rotation.

rocblas_status rocblas_drotg (rocblas_handle handle, double *a, double *b, double *c, double *s)

rocblas_status rocblas_crotg (rocblas_handle handle, rocblas_float_complex *a, rocblas_float_complex *b, float *c, rocblas_float_complex *s)

rocblas_status rocblas_zrotg (rocblas_handle handle, rocblas_double_complex *a, rocblas_double_complex *b, double *c, rocblas_double_complex *s)

2.12.1.9.2.31 rocblas_<type>rotg_batched()

rocblas_status rocblas_srotg_batched (rocblas_handle handle, float *const a[], float *const b[], float *const c[], float *const s[], rocblas_int batch_count)

BLAS Level 1 API.

rocrot_batched creates the Givens rotation matrix for the batched vectors (a_i b_i), for i = 1, ..., batch_count. a, b, c, and s may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.
Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [inout] a: device array of device pointers storing each single input vector element \( a_i \), overwritten with \( r_i \).
- [inout] b: device array of device pointers storing each single input vector element \( b_i \), overwritten with \( z_i \).
- [inout] c: device array of device pointers storing each cosine element of Givens rotation for the batch.
- [inout] s: device array of device pointers storing each sine element of Givens rotation for the batch.
- [in] batch_count: [rocblas_int] number of batches (length of arrays a, b, c, and s).

\[
\text{rocblas_status rocblas_drotg_batched(rocblas_handle handle, double *const a[], double *const b[], double *const c[], double *const s[], rocblas_int batch_count)}
\]

\[
\text{rocblas_status rocblas_crotg_batched(rocblas_handle handle, rocblas_float_complex *const a[], rocblas_float_complex *const b[], float *const c[], rocblas_float_complex *const s[], rocblas_int batch_count)}
\]

\[
\text{rocblas_status rocblas_zrotg_batched(rocblas_handle handle, rocblas_double_complex *const a[], rocblas_double_complex *const b[], double *const c[], rocblas_double_complex *const s[], rocblas_int batch_count)}
\]

2.12.9.2.32 rocblas_<type>rotg_strided_batched()

\[
\text{rocblas_status rocblas_srotg_strided_batched(rocblas_handle handle, float *a, rocblas_stride stride_a, float *b, rocblas_stride stride_b, float *c, rocblas_stride stride_c, float *s, rocblas_stride stride_s, rocblas_int batch_count)}
\]

BLAS Level 1 API

roc\_strided\_batched creates the Givens rotation matrix for the strided batched vectors \((a_i b_i)\), for \(i = 1, \ldots, \text{batch\_count}\). a, b, c, and s may be stored in either host or device memory, location is specified by calling rocblas_set\_pointer\_mode. If the pointer mode is set to rocblas\_pointer\_mode\_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas\_pointer\_mode\_device, this function returns immediately and synchronization is required to read the results.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [inout] a: device strided\_batched pointer or host strided\_batched pointer to first single input vector element \( a_1 \), overwritten with \( r \).
- [in] stride\_a: [rocblas_stride] distance between elements of a in batch (distance between \( a_i \) and \( a_{(i + 1)} \))
- [inout] b: device strided\_batched pointer or host strided\_batched pointer to first single input vector element \( b_1 \), overwritten with \( z \).
- [in] stride\_b: [rocblas_stride] distance between elements of b in batch (distance between \( b_i \) and \( b_{(i + 1)} \))
• [inout] c: device strided_batched pointer or host strided_batched pointer to first cosine element of Givens rotations c_1.
• [in] stride_c: [rocblas_stride] distance between elements of c in batch (distance between c_i and c_(i + 1))
• [inout] s: device strided_batched pointer or host strided_batched pointer to sine element of Givens rotations s_1.
• [in] stride_s: [rocblas_stride] distance between elements of s in batch (distance between s_i and s_(i + 1))
• [in] batch_count: [rocblas_int] number of batches (length of arrays a, b, c, and s).

rocblas_status rocblas_drotg_strided_batched(rocblas_handle handle, double *a, rocblas_stride stride_a, double *b, rocblas_stride stride_b, double *c, rocblas_stride stride_c, double *s, rocblas_stride stride_s, rocblas_int batch_count)

rocblas_status rocblas_crotg_strided_batched(rocblas_handle handle, rocblas_float_complex *a, rocblas_stride stride_a, rocblas_float_complex *b, rocblas_stride stride_b, float *c, rocblas_stride stride_c, rocblas_float_complex *s, rocblas_stride stride_s, rocblas_int batch_count)

rocblas_status rocblas_zrotg_strided_batched(rocblas_handle handle, rocblas_double_complex *a, rocblas_stride stride_a, rocblas_double_complex *b, rocblas_stride stride_b, double *c, rocblas_stride stride_c, rocblas_double_complex *s, rocblas_stride stride_s, rocblas_int batch_count)

2.12.1.9.2.33 rocblas_<type>rotm()

rocblas_status rocblas_srotm(rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, float *y, rocblas_int incy, const float *param)

BLAS Level 1 API.

rotm applies the modified Givens rotation matrix defined by param to vectors x and y.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in the x and y vectors.
• [inout] x: device pointer storing vector x.
• [in] incx: [rocblas_int] specifies the increment between elements of x.
• [inout] y: device pointer storing vector y.
• [in] incy: [rocblas_int] specifies the increment between elements of y.
• [in] param: device vector or host vector of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = 2 => H = ( 1.0 0.0 0.0 1.0 ) param may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.
rocblas_status rocblas_drotm(rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, double *y, rocblas_int incy, const double *param)

2.12.1.9.2.34 rocblas_<type>rotm_batched()

rocblas_status rocblas_srotm_batched(rocblas_handle handle, rocblas_int n, float *const x[], rocblas_int inx, float *const y[], rocblas_int incy, const float *const param[], rocblas_int batch_count)

BLAS Level 1 API.

rotm_batched applies the modified Givens rotation matrix defined by param_i to batched vectors x_i and y_i, for i = 1, ..., batch_count.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in the x and y vectors.
- [inout] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment between elements of each x_i.
- [inout] y: device array of device pointers storing each vector y_i.
- [in] incy: [rocblas_int] specifies the increment between elements of each y_i.
- [in] param: device array of device vectors of 5 elements defining the rotation. param[0] = flag
  form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H
  = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may ONLY be stored on the device
  for the batched version of this function.
- [in] batch_count: [rocblas_int] the number of x and y arrays, i.e. the number of batches.

rocblas_status rocblas_drotm_batched(rocblas_handle handle, rocblas_int n, double *const x[], rocblas_int inx, double *const y[], rocblas_int incy, const double *const param[], rocblas_int batch_count)

2.12.1.9.2.35 rocblas_<type>rotm_strided_batched()

rocblas_status rocblas_srotm_strided_batched(rocblas_handle handle, rocblas_int n, float *x, rocblas_int inx, float *stride_x, float *y, rocblas_int incy, rocblas_stride stride_y, const float *param, rocblas_stride stride_param, rocblas_int batch_count)

BLAS Level 1 API.

rotm_strided_batched applies the modified Givens rotation matrix defined by param_i to strided batched vectors x_i and y_i, for i = 1, ..., batch_count

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in the x and y vectors.
- [inout] x: device pointer pointing to first strided batched vector x_1.
• [in] incx: [rocblas_int] specifies the increment between elements of each x_i.
• [in] stride_x: [rocblas_stride] specifies the increment between the beginning of x_i and x_(i + 1)
• [inout] y: device pointer pointing to first strided batched vector y_1.
• [in] incy: [rocblas_int] specifies the increment between elements of each y_i.
• [in] stride_y: [rocblas_stride] specifies the increment between the beginning of y_i and y_(i + 1)
• [in] param: device pointer pointing to first array of 5 elements defining the rotation (param_1). param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may ONLY be stored on the device for the strided_batched version of this function.
• [in] stride_param: [rocblas_stride] specifies the increment between the beginning of param_i and param_(i + 1)
• [in] batch_count: [rocblas_int] the number of x and y arrays, i.e. the number of batches.

rocblas_status rocblas_drotm_strided_batched (rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, rocblas_stride stride_x, double *y, rocblas_int incy, rocblas_stride stride_y, const double *param, rocblas_stride stride_param, rocblas_int batch_count)

2.12.1.9.2.36 rocblas_<type>rotmg()

rocblas_status rocblas_srotmg (rocblas_handle handle, float *d1, float *d2, float *x1, const float *y1, float *param)

BLAS Level 1 API.

rotmg creates the modified Givens rotation matrix for the vector (d1 * x1, d2 * y1). Parameters may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [inout] d1: device pointer or host pointer to input scalar that is overwritten.
• [inout] d2: device pointer or host pointer to input scalar that is overwritten.
• [inout] x1: device pointer or host pointer to input scalar that is overwritten.
• [in] y1: device pointer or host pointer to input scalar.
• [out] param: device vector or host vector of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.
rocblas_status rocblas_drotmg (rocblas_handle handle, double *d1, double *d2, double *x1, const double *y1, double *param)

2.12.1.9.2.37 rocblas_<type>rotmg_batched()

rocblas_status rocblas_srotmg_batched (rocblas_handle handle, float *const d1[], float *const d2[], float *const x1[], const float *const y1[], float *const param[], rocblas_int batch_count)

BLAS Level 1 API.

rotmg_batched creates the modified Givens rotation matrix for the batched vectors (d1_i * x1_i, d2_i * y1_i), for i = 1, ..., batch_count. Parameters may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [inout] d1: device batched array or host batched array of input scalars that is overwritten.
- [inout] d2: device batched array or host batched array of input scalars that is overwritten.
- [inout] x1: device batched array or host batched array of input scalars that is overwritten.
- [in] y1: device batched array or host batched array of input scalars.
- [out] param: device batched array or host batched array of vectors of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.
- [in] batch_count: [rocblas_int] the number of instances in the batch.

rocblas_status rocblas_drotmg_batched (rocblas_handle handle, double *const d1[], double *const d2[], double *const x1[], const double *const y1[], double *const param[], rocblas_int batch_count)

2.12.1.9.2.38 rocblas_<type>rotmg_strided_batched()

rocblas_status rocblas_srotmg_strided_batched (rocblas_handle handle, float d1, rocblas_stride stride_d1, float d2, rocblas_stride stride_d2, float x1, rocblas_stride stride_x1, const float y1, rocblas_stride stride_y1, float param, rocblas_stride stride_param, rocblas_int batch_count)

BLAS Level 1 API.

rotmg_strided_batched creates the modified Givens rotation matrix for the strided batched vectors (d1_i * x1_i, d2_i * y1_i), for i = 1, ..., batch_count. Parameters may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

2.12. ROCm Libraries 111
Parameters

- **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.
- **[inout] d1**: device strided_batched array or host strided_batched array of input scalars that is overwritten.
- **[in] stride_d1**: [rocblas_stride] specifies the increment between the beginning of d1_i and d1_(i+1)
- **[inout] d2**: device strided_batched array or host strided_batched array of input scalars that is overwritten.
- **[in] stride_d2**: [rocblas_stride] specifies the increment between the beginning of d2_i and d2_(i+1)
- **[inout] x1**: device strided_batched array or host strided_batched array of input scalars.
- **[in] stride_x1**: [rocblas_stride] specifies the increment between the beginning of x1_i and x1_(i+1)
- **[in] y1**: device strided_batched array or host strided_batched array of input scalars.
- **[in] stride_y1**: [rocblas_stride] specifies the increment between the beginning of y1_i and y1_(i+1)
- **[out] param**: device strided_batched array or host strided_batched array of vectors of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.
- **[in] stride_param**: [rocblas_stride] specifies the increment between the beginning of param_i and param_(i + 1)
- **[in] batch_count**: [rocblas_int] the number of instances in the batch.

```c
rocblas_status rocblas_drotmg_strided_batched(rocblas_handle handle, double *d1, rocblas_stride stride_d1, double *d2, rocblas_stride stride_d2, double *x1, rocblas_stride stride_x1, const double *y1, rocblas_stride stride_y1, const double *param, rocblas_stride stride_param, rocblas_int batch_count)
```

### 2.12.1.9.2.39 Level 2 BLAS

#### 2.12.1.9.2.40 rocblas_<type>gemv()

```c
rocblas_status rocblas_dgemv(rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, const double *x, rocblas_int incx, const double *beta, double *y, rocblas_int incy)
rocblas_status rocblas_sgemv(rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, const float *x, rocblas_int incx, const float *beta, float *y, rocblas_int incy)
```

xGEMV performs one of the matrix-vector operations
\[ y := \alpha A^\ast x + \beta y, \quad \text{or} \]
\[ y := \alpha A^\ast T x + \beta y, \quad \text{or} \]
\[ y := \alpha A^\ast H x + \beta y, \]

where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are vectors and \( A \) is an \( m \times n \) matrix.

**Parameters**

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] trans: [rocblas_operation] indicates whether matrix \( A \) is transposed (conjugated) or not.
- [in] m: [rocblas_int] number of rows of matrix \( A \)
- [in] n: [rocblas_int] number of columns of matrix \( A \)
- [in] alpha: device pointer or host pointer to scalar \( \alpha \)
- [in] A: device pointer storing matrix \( A \).
- [in] lda: [rocblas_int] specifies the leading dimension of \( A \).
- [in] x: device pointer storing vector \( x \).
- [in] incx: [rocblas_int] specifies the increment for the elements of \( x \).
- [in] beta: device pointer or host pointer to scalar \( \beta \).
- [in] y: device pointer storing vector \( y \).
- [in] incy: [rocblas_int] specifies the increment for the elements of \( y \).

\[
\text{rocblas_status rocblas_cgemv} \left( \text{rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *x, rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy} \right)
\]

\[
\text{rocblas_status rocblas_zgemv} \left( \text{rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, const rocblas_double_complex *x, rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy} \right)
\]

\[
\text{rocblas_status rocblas_chemv} \left( \text{rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *x, rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy} \right)
\]

**2.12.1.9.2.41 rocblas_<type>hemv()**

\[
\text{rocblas_status rocblas_chemv} \left( \text{rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *x, rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy} \right)
\]

**BLAS Level 2 API.**

xHEMV performs one of the matrix-vector operations

\[ y := \alpha A^\ast x + \beta y \]

where \( \alpha \) and \( \beta \) are scalars, \( x \) and \( y \) are \( n \) element vectors and \( A \) is an \( n \times n \) hermitian matrix.

2.12. ROCm Libraries
Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** uplo: [rocblas_fill] rocblas_fill_upper: A is an upper banded triangular matrix. rocblas_fill_lower: A is a lower banded triangular matrix.
- **[in]** n: [rocblas_int] the order of the matrix A.
- **[in]** alpha: device pointer or host pointer to scalar alpha.
- **[in]** A: device pointer storing matrix A. Of dimension (lda, n). if uplo == rocblas_fill_upper: The upper triangular part of A must contain the upper triangular part of a hermitian matrix. The lower triangular part of A will not be referenced. if uplo == rocblas_fill_lower: The lower triangular part of A must contain the lower triangular part of a hermitian matrix. The upper triangular part of A will not be referenced. As a hermitian matrix, the imaginary part of the main diagonal of A will not be referenced and is assumed to be == 0.
- **[in]** lda: [rocblas_int] specifies the leading dimension of A. must be >= max(1, n)
- **[in]** x: device pointer storing vector x.
- **[in]** incx: [rocblas_int] specifies the increment for the elements of x.
- **[in]** beta: device pointer or host pointer to scalar beta.
- **[inout]** y: device pointer storing vector y.
- **[in]** incy: [rocblas_int] specifies the increment for the elements of y.

```c
rocblas_status rocblas_zhemv(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, const rocblas_double_complex *x, rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy)
```

2.12.1.9.2.42 rocblas_<type>hemv_batched()

```c
rocblas_status rocblas_chemv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *const A[], rocblas_int lda, const rocblas_float_complex *const x[], rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)
```

BLAS Level 2 API.

xHEMV_BATCHED performs one of the matrix-vector operations

\[ y_i := \alpha A_i x_i + \beta y_i \]

where alpha and beta are scalars, x_i and y_i are n element vectors and A_i is an n by n hermitian matrix, for each batch in i = [1, batch_count].

Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** uplo: [rocblas_fill] rocblas_fill_upper: A is an upper banded triangular matrix. rocblas_fill_lower: A is a lower banded triangular matrix.
- **[in]** n: [rocblas_int] the order of each matrix A_i.
• [in] alpha: device pointer or host pointer to scalar alpha.
• [in] A: device array of device pointers storing each matrix $A_i$ of dimension $(lda, n)$. if uplo == rocblas_fill_upper: The upper triangular part of each $A_i$ must contain the upper triangular part of a hermitian matrix. The lower triangular part of each $A_i$ will not be referenced. if uplo == rocblas_fill_lower: The lower triangular part of each $A_i$ must contain the lower triangular part of a hermitian matrix. The upper triangular part of each $A_i$ will not be referenced. As a hermitian matrix, the imaginary part of the main diagonal of each $A_i$ will not be referenced and is assumed to be $== 0$.
• [in] lda: [rocblas_int] specifies the leading dimension of each $A_i$. must be $>= \max(1, n)$
• [in] x: device array of device pointers storing each vector $x_i$.
• [in] incx: [rocblas_int] specifies the increment for the elements of each $x_i$.
• [in] beta: device pointer or host pointer to scalar beta.
• [inout] y: device array of device pointers storing each vector $y_i$.
• [in] incy: [rocblas_int] specifies the increment for the elements of y.
• [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_zhemv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *const A[], rocblas_int lda, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)

2.12.1.9.2.43 rocblas_<type>hemv_strided_batched()

rocblas_status rocblas_chemv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *const A, rocblas_int lda, rocblas_stride stride_A, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stride_y, rocblas_int batch_count)

BLAS Level 2 API.

xHEMV_STRIDED_BATCHED performs one of the matrix-vector operations

$$y_i := \alpha A_i x_i + \beta y_i$$

where alpha and beta are scalars, $x_i$ and $y_i$ are $n$ element vectors and $A_i$ is an $n$ by $n$ hermitian matrix, for each batch in $i = [1, \text{batch\_count}]$.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] uplo: [rocblas_fill] rocblas_fill_upper: $A$ is an upper banded triangular matrix. rocblas_fill_lower: $A$ is a lower banded triangular matrix.
• [in] n: [rocblas_int] the order of each matrix $A_i$.  

2.12. ROCm Libraries
- [in] alpha: device pointer or host pointer to scalar alpha.
- [in] A: device array of device pointers storing each matrix A_i of dimension (lda, n). if uplo == rocblas_fill_upper: The upper triangular part of each A_i must contain the upper triangular part of a hermitian matrix. The lower triangular part of each A_i will not be referenced. if uplo == rocblas_fill_lower: The lower triangular part of each A_i must contain the lower triangular part of a hermitian matrix. The upper triangular part of each A_i will not be referenced. As a hermitian matrix, the imaginary part of the main diagonal of each A_i will not be referenced and is assumed to be == 0.
- [in] lda: [rocblas_int] specifies the leading dimension of each A_i. must be >= max(1, n)
- [in] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
- [in] beta: device pointer or host pointer to scalar beta.
- [inout] y: device array of device pointers storing each vector y_i.
- [in] incy: [rocblas_int] specifies the increment for the elements of y.
- [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_zhemv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stride_A, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stride_y, rocblas_int batch_count)

2.12.1.9.2.44 rocblas_<type>gemv_batched()

rocblas_status rocblas_sgemv_batched(rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const float *alpha, const float *const A[], rocblas_int lda, const float *const x[], rocblas_int incx, const float *beta, float *const y[], rocblas_int incy, rocblas_int batch_count)

BLAS Level 2 API.

xGEMV_BATCHED performs a batch of matrix-vector operations

\[
\begin{align*}
    y_i := \alpha A_i x_i + \beta y_i, & \quad \text{or} \\
    y_i := \alpha A_i^T x_i + \beta y_i, & \quad \text{or} \\
    y_i := \alpha A_i^H x_i + \beta y_i,
\end{align*}
\]

where \((A_i, x_i, y_i)\) is the \(i\)-th instance of the batch. alpha and beta are scalars, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

**Parameters**

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] trans: [rocblas_operation] indicates whether matrices \(A_i\) are tranposed (conjugated) or not
- [in] m: [rocblas_int] number of rows of each matrix \(A_i\)
- [in] n: [rocblas_int] number of columns of each matrix \(A_i\)
• [in] alpha: device pointer or host pointer to scalar alpha.
• [in] A: device array of device pointers storing each matrix A_i.
• [in] lda: [rocblas_int] specifies the leading dimension of each matrix A_i.
• [in] x: device array of device pointers storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of each vector x_i.
• [in] beta: device pointer or host pointer to scalar beta.
• [inout] y: device array of device pointers storing each vector y_i.
• [in] incy: [rocblas_int] specifies the increment for the elements of each vector y_i.
• [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_dgemv_batched (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const double *alpha, const double *const A[], rocblas_int lda, const double *const x[], rocblas_int incx, const double *beta, double *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_cgemv_batched (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *const A[], rocblas_int lda, const rocblas_float_complex *const x[], rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_zgemv_batched (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *const A[], rocblas_int lda, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)

2.12.1.9.2.45 rocblas_<type>gemv_strided_batched()

rocblas_status rocblas_sgemv_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, rocblas_stride strideA, const float *x, rocblas_int incx, rocblas_stride stridex, const float *beta, float *y, rocblas_int incy, rocblas_stride stridy, rocblas_int batch_count)

BLAS Level 2 API.

xGEMV_STRIDED_BATCHED performs a batch of matrix-vector operations

\[
\begin{align*}
y_i &= \alpha A_i x_i + \beta y_i, \\
y_i &= \alpha A_i^T x_i + \beta y_i, \\
y_i &= \alpha A_i^H x_i + \beta y_i,
\end{align*}
\]

where \((A_i, x_i, y_i)\) is the \(i\)-th instance of the batch. \(\alpha\) and \(\beta\) are scalars, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

Parameters
• `[in] handle`: rocblas library context queue.

• `[in] transA`: indicates whether matrices A_i are tranposed (conjugated) or not

• `[in] m`: number of rows of matrices A_i

• `[in] n`: number of columns of matrices A_i

• `[in] alpha`: device pointer or host pointer to scalar alpha.

• `[in] A`: device pointer to the first matrix (A_1) in the batch.

• `[in] lda`: specifies the leading dimension of matrices A_i.

• `[in] strideA`: stride from the start of one matrix (A_i) and the next one (A_i+1)

• `[in] x`: device pointer to the first vector (x_1) in the batch.

• `[in] incx`: specifies the increment for the elements of vectors x_i.

• `[in] stridex`: stride from the start of one vector (x_i) and the next one (x_i+1). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size. When trans equals rocblas_operation_none this typically means stride_x >= n * incx, otherwise stride_x >= m * incx.

• `[in] beta`: device pointer or host pointer to scalar beta.

• `[inout] y`: device pointer to the first vector (y_1) in the batch.

• `[in] incy`: specifies the increment for the elements of vectors y_i.

• `[in] stridex`: stride from the start of one vector (y_i) and the next one (y_i+1). There are no restrictions placed on stride_y, however the user should take care to ensure that stride_y is of appropriate size. When trans equals rocblas_operation_none this typically means stride_y >= m * incy, otherwise stride_y >= n * incy. stridexy should be non-zero.

• `[in] batch_count`: number of instances in the batch

```c
rocblas_status rocblas_dgemv_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, rocblas_stride strideA, const double *x, rocblas_int incx, rocblas_stride stridex, const double *beta, double *y, rocblas_int incy, rocblas_stride stridy, rocblas_int batch_count)
```

```c
rocblas_status rocblas_cgemv_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride strideA, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridy, rocblas_int batch_count)
```
rocblas_status rocblas_zgemv_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride strideA, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridy, rocblas_int batch_count)

2.12.1.9.2.46 rocblas_<type>trmv()

rocblas_status rocblas_strmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, float *x, rocblas_int incx)

rocblas_status rocblas_dtrmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, double *x, rocblas_int incx)

rocblas_status rocblas_ctrmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_float_complex *A, rocblas_int lda, rocblas_float_complex *x, rocblas_int incx)

rocblas_status rocblas_ztrmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_double_complex *A, rocblas_int lda, rocblas_double_complex *x, rocblas_int incx)

BLAS Level 2 API.

trmv performs one of the matrix-vector operations

\[
x = A \cdot x \text{ or } x = A^{\ast \ast T} \cdot x,
\]

where \(x\) is an \(n\) element vector and \(A\) is an \(n\) by \(n\) unit, or non-unit, upper or lower triangular matrix.

The vector \(x\) is overwritten.

Parameters

- [in] handle: rocblas_handle handle to the rocblas library context queue.
- [in] uplo: rocblas_fill upper: \(A\) is an upper triangular matrix. rocblas_fill_lower: \(A\) is a lower triangular matrix.
- [in] transA: rocblas_operation
- [in] diag: rocblas_diagonal unit: \(A\) is assumed to be unit triangular. rocblas_diagonal_non_unit: \(A\) is not assumed to be unit triangular.
- [in] m: rocblas_int \(m\) specifies the number of rows of \(A\). \(m \geq 0\).
- [in] A: device pointer storing matrix \(A\), of dimension (lda, m )
- [in] lda: rocblas_int specifies the leading dimension of \(A\). lda = max( 1, m ).
- [in] x: device pointer storing vector \(x\).
- [in] incx: rocblas_int specifies the increment for the elements of \(x\).
2.12.1.9.2.47 rocblas_<type>trmv_batched()

rocblas_status rocblas_strmv_batched
(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *const *A, rocblas_int lda, float *const *x, rocblas_int incx, rocblas_int batch_count)

BLAS Level 2 API.

trmv_batched performs one of the matrix-vector operations

\[ x_i = A_i \times x_i \text{ or } x_i = A^T \times x_i, \quad 0 \leq i < \text{batch_count} \]

where \( x_i \) is an \( n \) element vector and \( A_i \) is an \( n \) by \( n \) (unit, or non-unit, upper or lower triangular matrix)

The vectors \( x_i \) are overwritten.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: \( A_i \) is an upper triangular matrix. rocblas_fill_lower: \( A_i \) is a lower triangular matrix.
- [in] transA: [rocblas_operation]
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: \( A_i \) is assumed to be unit triangular. rocblas_diagonal_non_unit: \( A_i \) is not assumed to be unit triangular.
- [in] m: [rocblas_int] \( m \) specifies the number of rows of matrices \( A_i \). \( m \geq 0 \).
- [in] A: device pointer storing pointer of matrices \( A_i \), of dimension ( \( lda, m \) )
- [in] lda: [rocblas_int] specifies the leading dimension of \( A_i \). \( lda \geq \max(1, m) \).
- [in] x: device pointer storing vectors \( x_i \).
- [in] incx: [rocblas_int] specifies the increment for the elements of vectors \( x_i \).

rocblas_status rocblas_dtrmv_batched
(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *const *A, rocblas_int lda, double *const *x, rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_ctrmv_batched
(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_float_complex *const *A, rocblas_int lda, rocblas_float_complex *const *x, rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_ztrmv_batched
(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_double_complex *const *A, rocblas_int lda, rocblas_double_complex *const *x, rocblas_int incx, rocblas_int batch_count)
2.12.1.9.2.48 rocblas_<type>trmv_strided_batched()

rocblas_status rocblas_strmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, rocblas_stride stridea, float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)

BLAS Level 2 API.

trmv_strided_batched performs one of the matrix-vector operations

\[ x_i = A_i x_i \text{ or } x_i = A_i^T x_i, ~ 0 \le i < \text{batch_count} \]

where \( x_i \) is an \( n \) element vector and \( A_i \) is an \( n \) by \( n \) (unit, or non-unit, upper or lower triangular matrix) with strides specifying how to retrieve \( x_{i-1} \) (resp. \( A_{i-1} \)) from \( x_i \) (resp. \( A_i \)).

The vectors \( x_i \) are overwritten.

Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** uplo: [rocblas_fill] rocblas_fill_upper: \( A_i \) is an upper triangular matrix. rocblas_fill_lower: \( A_i \) is a lower triangular matrix.
- **[in]** transA: [rocblas_operation]
- **[in]** diag: [rocblas_diagonal] rocblas_diagonal_unit: \( A_i \) is assumed to be unit triangular. rocblas_diagonal_non_unit: \( A_i \) is not assumed to be unit triangular.
- **[in]** m: [rocblas_int] \( m \) specifies the number of rows of matrices \( A_i \). \( m \geq 0 \).
- **[in]** A: device pointer of the matrix \( A_0 \), of dimension (lda, m)
- **[in]** lda: [rocblas_int] specifies the leading dimension of \( A_i \). \( \text{lda} \geq \max(1, m) \).
- **[in]** stride_a: [rocblas_stride] stride from the start of one \( A_i \) matrix to the next \( A_{i+1} \)
- **[in]** x: device pointer storing the vector \( x_0 \).
- **[in]** incx: [rocblas_int] specifies the increment for the elements of one vector \( x \).
- **[in]** stride_x: [rocblas_stride] stride from the start of one \( x_i \) vector to the next \( x_{i+1} \)
- **[in]** batch_count: [rocblas_int] The number of batched matrices/vectors.

rocblas_status rocblas_dtrmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, rocblas_stride stridea, double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)

rocblas_status rocblas_ctrmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stridea, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)
rocblas_status rocblas_ztrmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride strideA, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)

2.12.1.9.2.49 rocblas_<type>tbmv()

rocblas_status rocblas_stbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const float *A, rocblas_int lda, float *x, rocblas_int incx)

BLAS Level 2 API.
xTBMV performs one of the matrix-vector operations

\[
x := A \times x \quad \text{or} \\
x := A^{\top} \times x \quad \text{or} \\
x := A^{\ast \top} \times x,
\]

x is a vector and A is a banded m by m matrix (see description below).

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: A is an upper banded triangular matrix. rocblas_fill_lower: A is a lower banded triangular matrix.
- [in] trans: [rocblas_operation] indicates whether matrix A is transposed (conjugated) or not.
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: The main diagonal of A is assumed to consist of only 1’s and is not referenced. rocblas_diagonal_non_unit: No assumptions are made of A’s main diagonal.
- [in] m: [rocblas_int] the number of rows and columns of the matrix represented by A.
- [in] k: [rocblas_int] if uplo == rocblas_fill_upper, k specifies the number of super-diagonals of the matrix A. if uplo == rocblas_fill_lower, k specifies the number of sub-diagonals of the matrix A. k must satisfy k > 0 && k < lda.
- [in] A: device pointer storing banded triangular matrix A. if uplo == rocblas_fill_upper: The matrix represented is an upper banded triangular matrix with the main diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the k’th row, the first super diagonal resides on the RHS of the k-1’th row, etc, with the k’th diagonal on the RHS of the 0’th row. Ex: (rocblas_fill_upper; \(m = 5; k = 2\)) 1 6 9 0 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 0 3 8 7 -> 1 2 3 4 5 0 0 0 4 9 0 0 0 0 0 0 0 0 0 5 0 0 0 0 if uplo == rocblas_fill_lower: The matrix represented is a lower banded triangular matrix with the main diagonal and k sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the 0’th row, working up to the k’th diagonal residing on the LHS of the k’th row. Ex: (rocblas_fill_lower; \(m = 5; k = 2\)) 1 0 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 0 0 7 9 5 0 0 0 0 0 0
- [in] lda: [rocblas_int] specifies the leading dimension of A. lda must satisfy lda > k.
- [inout] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
rocblas_status rocblas_dtbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const double *A, rocblas_int lda, double *x, rocblas_int incx)

rocblas_status rocblas_ctbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_float_complex *A, rocblas_int lda, rocblas_float_complex *x, rocblas_int incx)

rocblas_status rocblas_ztbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_double_complex *A, rocblas_int lda, rocblas_double_complex *x, rocblas_int incx)

### 2.12.1.9.2.50 rocblas_<type>tbmv_batched()

rocblas_status rocblas_stbmv_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const float *A[], rocblas_int lda, float *x[], rocblas_int incx, rocblas_int batch_count)

BLAS Level 2 API.

xTBMV_BATCHED performs one of the matrix-vector operations

\[
x_i := A_i x_i \quad \text{or} \quad x_i := A_i^T x_i \quad \text{or} \quad x_i := A_i^H x_i,
\]

where \((A_i, x_i)\) is the \(i\)-th instance of the batch. \(x_i\) is a vector and \(A_i\) is an \(m\) by \(m\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

**Parameters**

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** uplo: [rocblas_fill] rocblas_fill_upper: each \(A_i\) is an upper banded triangular matrix. rocblas_fill_lower: each \(A_i\) is a lower banded triangular matrix.
- **[in]** trans: [rocblas_operation] indicates whether each matrix \(A_i\) is tranposed (conjugated) or not.
- **[in]** diag: [rocblas_diagonal] rocblas_diagonal_unit: The main diagonal of each \(A_i\) is assumed to consist of only 1’s and is not referenced. rocblas_diagonal_non_unit: No assumptions are made of each \(A_i\)’s main diagonal.
- **[in]** m: [rocblas_int] the number of rows and columns of the matrix represented by each \(A_i\).
- **[in]** k: [rocblas_int] if uplo == rocblas_fill_upper, \(k\) specifies the number of super-diagonals of each matrix \(A_i\). if uplo == rocblas_fill_lower, \(k\) specifies the number of sub-diagonals of each matrix \(A_i\). \(k\) must satisfy \(k > 0 \&\& k < lda\).
- **[in]** A: device array of device pointers storing each banded triangular matrix \(A_i\). if uplo == rocblas_fill_upper: The matrix represented is an upper banded triangular matrix with the main diagonal and \(k\) super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the \(k\)’th row, the first super diagonal resides on the RHS of the \(k\)-1’th row, etc. with the \(k\)’th diagonal on the RHS of the 0’th row. Ex: (rocblas_fill_upper; \(m = 5; \ k = 2\) ) 1 6 9 0 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 0 3 8 7 \(\rightarrow 1 2 3 4 5 0 0 0 4 9 0 0 0 0 0 0 5 0 0 0 0\) if uplo == rocblas_fill_lower: The matrix represented is a lower banded triangular matrix with the main diagonal and \(k\) sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so
that the main diagonal resides on the 0'th row, working up to the k'th diagonal residing on the LHS of the k'th row. Ex: (rocblas_fill_lower; m = 5; k = 2) 1 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 0 0 0 0 0 0 0 0 9 7 5 0 0 0 0

- [in] lda: [rocblas_int] specifies the leading dimension of each A_i. lda must satisfy lda > k.
- [inout] x: device array of device pointer storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
- [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_dtbmv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const double *const A[], rocblas_int lda, double *const x[], rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_ctbmv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_float_complex *const A[], rocblas_int da, rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count)
rocblas_status rocblas_ztbmv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_double_complex *const A[], rocblas_int lda, rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count)

2.12.1.9.2.51 rocblas_<type>tbmv_strided_batched()

rocblas_status rocblas_stbmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const float *const A, rocblas_int lda, rocblas_stride stride_A, float *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

BLAS Level 2 API.

xTBMV_STRIDED_BATCHED performs one of the matrix-vector operations

\[
\begin{align*}
x_{i, 1} & := A_{i, 1} \ast x_{i, 1} \\
x_{i, 2} & := A_{i, 1} \ast T \ast x_{i, 2} \\
x_{i, 3} & := A_{i, 1} \ast H \ast x_{i, 3}
\end{align*}
\]

where \((A_{i, 1}, x_{i, 1})\) is the \(i\)-th instance of the batch. \(x_{i, 1}\) is a vector and \(A_{i, 1}\) is an \(m\) by \(m\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

**Parameters**

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: each A_i is an upper banded triangular matrix. rocblas_fill_lower: each A_i is a lower banded triangular matrix.
- [in] trans: [rocblas_operation] indicates whether each matrix A_i is tranposed (conjugated) or not.
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: The main diagonal of each A_i is assumed to consist of only 1's and is not referenced. rocblas_diagonal_non_unit: No assumptions are made of each A_i's main diagonal.

• [in] m: [rocblas_int] the number of rows and columns of the matrix represented by each A_i.

• [in] k: [rocblas_int] if uplo == rocblas_fill_upper, k specifies the number of super-diagonals of each matrix A_i. if uplo == rocblas_fill_lower, k specifies the number of sub-diagonals of each matrix A_i. k must satisfy k > 0 && k < lda.

• [in] A: device array to the first matrix A_i of the batch. Stores each banded triangular matrix A_i. if uplo == rocblas_fill_upper: The matrix represented is an upper banded triangular matrix with the main diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the k’th row, the first super diagonal resides on the RHS of the k-1’th row, etc, with the k’th diagonal on the RHS of the 0’th row. Ex: (rocblas_fill_upper; m = 5; k = 2) 1 6 9 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 0 3 8 7 -> 1 2 3 4 5 0 0 0 4 9 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 0 7 9 5 0 0 0 0 0

• [in] lda: [rocblas_int] specifies the leading dimension of each A_i. lda must satisfy lda > k.

• [in] stride_A: [rocblas_stride] stride from the start of one A_i matrix to the next A_(i + 1).

• [inout] x: device array to the first vector x_i of the batch.

• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.

• [in] stride_x: [rocblas_stride] stride from the start of one x_i matrix to the next x_(i + 1).

• [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_dtbmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const double *A, rocblas_int lda, rocblas_stride stride_A, double *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_ctbmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stride_A, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_ztbmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stride_A, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)
2.12.1.9.2.52 rocblas_<type>trsv()

rocblas_status rocblas_dtrsv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, double *x, rocblas_int incx)

rocblas_status rocblas_strsv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, float *x, rocblas_int incx)

BLAS Level 2 API.

trsv solves

\[ A \cdot x = b \text{ or } A^{\top} \cdot x = b, \]

where x and b are vectors and A is a triangular matrix.

The vector x is overwritten on b.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
- [in] transA: [rocblas_operation]
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: A is assumed to be unit triangular. rocblas_diagonal_non_unit: A is not assumed to be unit triangular.
- [in] m: [rocblas_int] m specifies the number of rows of b. m >= 0.
- [in] A: device pointer storing matrix A, of dimension (lda, m)
- [in] lda: [rocblas_int] specifies the leading dimension of A. lda = max(1, m).
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.

2.12.1.9.2.53 rocblas_<type>trsv_batched()

rocblas_status rocblas_strsv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A[], rocblas_int lda, const float *x[], rocblas_int incx, rocblas_int batch_count)

BLAS Level 2 API.

trsv_batched solves

\[ A_i \cdot x_i = b_i \text{ or } A_i^{\top} \cdot x_i = b_i, \]

where (A_i, x_i, b_i) is the i-th instance of the batch. x_i and b_i are vectors and A_i is an m by m triangular matrix.

The vector x is overwritten on b.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] uplo: [rocblas_fill] rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
• [in] transA: [rocblas_operation]
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: A is assumed to be unit triangular. rocblas_diagonal_non_unit: A is not assumed to be unit triangular.
• [in] m: [rocblas_int] m specifies the number of rows of b. m >= 0.
• [in] A: device array of device pointers storing each matrix A_i.
• [in] lda: [rocblas_int] specifies the leading dimension of each A_i. lda = max(1, m)
• [in] x: device array of device pointers storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of x.
• [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dtrsv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *const A[][], rocblas_int lda, const double *const x[][], rocblas_int incx, rocblas_int batch_count)

2.12.1.9.2.54 rocblas_<type>trsv_strided_batched()

rocblas_status rocblas_strsv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, const float *x, rocblas_int incx, rocblas_int stride_A, rocblas_int stride_x, rocblas_int batch_count)

BLAS Level 2 API.

trsv_strided_batched solves

A_i*x_i = b_i or A_i**T*x_i = b_i,

where (A_i, x_i, b_i) is the i-th instance of the batch. x_i and b_i are vectors and A_i is an m by m triangular matrix, for i = 1, ..., batch_count.

The vector x is overwritten on b.

Parameters
• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] uplo: [rocblas_fill] rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
• [in] transA: [rocblas_operation]
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: A is assumed to be unit triangular. rocblas_diagonal_non_unit: A is not assumed to be unit triangular.
• [in] m: [rocblas_int] m specifies the number of rows of each b_i. m >= 0.
• [in] A: device pointer to the first matrix (A_1) in the batch, of dimension (lda, m)
• [in] stride_A: [rocblas_stride] stride from the start of one A_i matrix to the next A_(i + 1)
• [in] lda: [rocblas_int] specifies the leading dimension of each A_i. lda = max(1, m).
• [inout] x: device pointer to the first vector \((x_1)\) in the batch.

• [in] stride_x: \([\text{rocblas_stride}]\) stride from the start of one \(x_i\) vector to the next \(x_{(i + 1)}\)

• [in] incx: \([\text{rocblas_int}]\) specifies the increment for the elements of each \(x_i\).

• [in] batch_count: \([\text{rocblas_int}]\) number of instances in the batch

rocblas_status rocblas_dtrsv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, rocblas_stride stride_A, double *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

2.12.1.9.2.55 rocblas_<type>ger()

rocblas_status rocblas_dger (rocblas_handle handle, rocblas_int m, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, const double *y, rocblas_int incy, double *A, rocblas_int lda)

rocblas_status rocblas_sger (rocblas_handle handle, rocblas_int m, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, const float *y, rocblas_int incy, float *A, rocblas_int lda)

BLAS Level 2 API.

xGER performs the matrix-vector operations

\[
A := A + \alpha x y^T
\]

where alpha is a scalar, \(x\) and \(y\) are vectors, and \(A\) is an \(m \times n\) matrix.

Parameters

- [in] handle: \([\text{rocblas_handle}]\) handle to the rocblas library context queue.

- [in] m: \([\text{rocblas_int}]\) the number of rows of the matrix \(A\).

- [in] n: \([\text{rocblas_int}]\) the number of columns of the matrix \(A\).

- [in] alpha: device pointer or host pointer to scalar \(\alpha\).

- [in] x: device pointer storing vector \(x\).

- [in] incx: \([\text{rocblas_int}]\) specifies the increment for the elements of \(x\).

- [in] y: device pointer storing vector \(y\).

- [in] incy: \([\text{rocblas_int}]\) specifies the increment for the elements of \(y\).

- [inout] A: device pointer storing matrix \(A\).

- [in] lda: \([\text{rocblas_int}]\) specifies the leading dimension of \(A\).
2.12.1.9.2.56 rocblas_<type>ger_batched()

rocblas_status rocblas_sger_batched (rocblas_handle handle, rocblas_int m, rocblas_int n, const float *alpha, const float *const x[], rocblas_int incx, const float *const y[], rocblas_int incy, float *const A[], rocblas_int lda, rocblas_int batch_count)

BLAS Level 2 API.

xGER_BATCHED performs a batch of the matrix-vector operations

\[ A_i := A_i + \alpha x_i y_i^T \]

where \((A_i, x_i, y_i)\) is the \(i\)-th instance of the batch. \(\alpha\) is a scalar, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] m: [rocblas_int] the number of rows of each matrix \(A_i\).
- [in] n: [rocblas_int] the number of columns of each matrix \(A_i\).
- [in] alpha: device pointer or host pointer to scalar \(\alpha\).
- [in] x: device array of device pointers storing each vector \(x_i\).
- [in] incx: [rocblas_int] specifies the increment for the elements of each vector \(x_i\).
- [in] y: device array of device pointers storing each vector \(y_i\).
- [in] incy: [rocblas_int] specifies the increment for the elements of each vector \(y_i\).
- [inout] A: device array of device pointers storing each matrix \(A_i\).
- [in] lda: [rocblas_int] specifies the leading dimension of each \(A_i\).
- [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dger_batched (rocblas_handle handle, rocblas_int m, rocblas_int n, const double *alpha, const double *const x[], rocblas_int incx, const double *const y[], rocblas_int incy, double *const A[], rocblas_int lda, rocblas_int batch_count)

2.12.1.9.2.57 rocblas_<type>ger_strided_batched()

rocblas_status rocblas_sger_strided_batched (rocblas_handle handle, rocblas_int m, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, rocblas_stride stridex, const float *y, rocblas_int incy, rocblas_stride stridey, float *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)

BLAS Level 2 API.

xGER_STRIDED_BATCHED performs the matrix-vector operations

\[ A_i := A_i + \alpha x_i y_i^T \]

where \((A_i, x_i, y_i)\) is the \(i\)-th instance of the batch. \(\alpha\) is a scalar, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).
Parameters

- **handle**: [rocblas_handle] handle to the rocblas library context queue.
- **m**: [rocblas_int] the number of rows of each matrix \( A_i \).
- **n**: [rocblas_int] the number of columns of each matrix \( A_i \).
- **alpha**: device pointer or host pointer to scalar alpha.
- **x**: device pointer to the first vector \( (x_1) \) in the batch.
- **incx**: [rocblas_int] specifies the increments for the elements of each vector \( x_i \).
- **stridex**: [rocblas_stride] stride from the start of one vector \( (x_i) \) and the next one \( (x_{i+1}) \).
  - There are no restrictions placed on \( \text{stride}_x \), however the user should take care to ensure that \( \text{stride}_x \) is of appropriate size, for a typical case this means \( \text{stride}_x \geq m \times \text{incx} \).
- **y**: device pointer to the first vector \( (y_0) \) in the batch.
- **incy**: [rocblas_int] specifies the increment for the elements of each vector \( y_i \).
- **stridey**: [rocblas_stride] stride from the start of one vector \( (y_i) \) and the next one \( (y_{i+1}) \).
  - There are no restrictions placed on \( \text{stride}_y \), however the user should take care to ensure that \( \text{stride}_y \) is of appropriate size, for a typical case this means \( \text{stride}_y \geq n \times \text{incy} \).
- **A**: device pointer to the first matrix \( (A_1) \) in the batch.
- **lda**: [rocblas_int] specifies the leading dimension of each \( A_i \).
- **strideA**: [rocblas_stride] stride from the start of one matrix \( (A_i) \) and the next one \( (A_{i+1}) \).
- **batch_count**: [rocblas_int] number of instances in the batch.

```
rocblas_status rocblas_dger_strided_batched(rocblas_handle handle, rocblas_int m, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, rocblas_stride stridex, const double *y, rocblas_int incy, rocblas_stride stridey, double *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)
```

### 2.12.1.9.2.58 rocblas_<type>syr()

```
rocblas_status rocblas_dsyrr(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, rocblas_int lda)
rocblas_status rocblas_ssyr(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, rocblas_int lda)
```

BLAS Level 2 API.

\[ A := A + \alpha x x^T \]

where \( \alpha \) is a scalar, \( x \) is a vector, and \( A \) is an \( n \) by \( n \) symmetric matrix.

Parameters

- **handle**: [rocblas_handle] handle to the rocblas library context queue.
- **uplo**: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’ if rocblas_fill_upper, the lower part of \( A \) is not referenced if rocblas_fill_lower, the upper part of \( A \) is not referenced.
• [in] n: [rocblas_int] the number of rows and columns of matrix A.
• [in] alpha: device pointer or host pointer to scalar alpha.
• [in] x: device pointer storing vector x.
• [in] incx: [rocblas_int] specifies the increment for the elements of x.
• [inout] A: device pointer storing matrix A.
• [in] lda: [rocblas_int] specifies the leading dimension of A.

### 2.12.1.9.2.59 rocblas_<type>syr_batched()

rocblas_status rocblas_ssyr_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const
float *alpha, const float *const x[], rocblas_int incx, float
*const A[], rocblas_int lda, rocblas_int batch_count)

BLAS Level 2 API.

xSYR_batched performs a batch of matrix-vector operations

\[ A[i] := A[i] + \alpha \cdot x[i] \cdot x[i]^T \]

where alpha is a scalar, x is an array of vectors, and A is an array of n by n symmetric matrices, for \( i = 1, \ldots, \) batch_count

**Parameters**

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] uplo: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’ if rocblas_fill_upper, the lower part of A is not referenced if rocblas_fill_lower, the upper part of A is not referenced.
• [in] n: [rocblas_int] the number of rows and columns of matrix A.
• [in] alpha: device pointer or host pointer to scalar alpha.
• [in] x: device array of device pointers storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
• [inout] A: device array of device pointers storing each matrix A_i.
• [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
• [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dsyr_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const
double *alpha, const double *const x[], rocblas_int incx,
double *const A[], rocblas_int lda, rocblas_int batch_count)
2.12.1.9.2.60 rocblas_<type>syr_strided_batched()

rocblas_status rocblas_ssyr_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, rocblas_stride stridex, float *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)

BLAS Level 2 API.

xSYR_strided_batched performs the matrix-vector operations

\[ A[i] := A[i] + \alpha x[i] x[i]^T \]

where alpha is a scalar, vectors, and A is an array of n by n symmetric matrices, for \( i = 1, \ldots, \text{batch\_count} \)

Parameters

- \( \text{[in]} \) handle: [rocblas_handle] handle to the rocblas library context queue.
- \( \text{[in]} \) uplo: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’ if rocblas_fill_upper, the lower part of A is not referenced if rocblas_fill_lower, the upper part of A is not referenced
- \( \text{[in]} \) n: [rocblas_int] the number of rows and columns of each matrix A.
- \( \text{[in]} \) alpha: device pointer or host pointer to scalar alpha.
- \( \text{[in]} \) x: device pointer to the first vector x_1.
- \( \text{[in]} \) incx: [rocblas_int] specifies the increment for the elements of each x_i.
- \( \text{[in]} \) stridex: [rocblas_stride] specifies the pointer increment between vectors (x_i) and (x_i+1).
- \( \text{[inout]} \) A: device pointer to the first matrix A_1.
- \( \text{[in]} \) lda: [rocblas_int] specifies the leading dimension of each A_i.
- \( \text{[in]} \) strideA: [rocblas_stride] stride from the start of one matrix (A_i) and the next one (A_i+1)
- \( \text{[in]} \) batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dsyr_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, rocblas_stride stridex, double *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)

2.12.1.9.2.61 Level 3 BLAS

2.12.1.9.2.62 rocblas_<type>trtri()

rocblas_status rocblas_strtri (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const float *A, rocblas_int lda, float *invA, rocblas_int ldinvA)

BLAS Level 3 API.

trtri compute the inverse of a matrix A, namely, invA

and write the result into invA;
Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’ if rocblas_fill_upper, the lower part of A is not referenced if rocblas_fill_lower, the upper part of A is not referenced.
- [in] diag: [rocblas_diagonal] = ‘rocblas_diagonal_non_unit’, A is non-unit triangular; = ‘rocblas_diagonal_unit’, A is unit triangular;
- [in] n: [rocblas_int] size of matrix A and invA
- [in] A: device pointer storing matrix A.
- [in] lda: [rocblas_int] specifies the leading dimension of A.
- [out] invA: device pointer storing matrix invA.
- [in] ldinvA: [rocblas_int] specifies the leading dimension of invA.

rocblas_status rocblas_dtrtri (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const double *A, rocblas_int lda, double *invA, rocblas_int ldinvA)

rocblas_status rocblas_strtri_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const float *A[], rocblas_int lda, float *invA[], rocblas_int ldinvA, rocblas_int batch_count)

2.12.1.9.2.63 rocblas_<type>trtri_batched()

BLAS Level 3 API.

trtri_batched compute the inverse of A_i and write into invA_i where A_i and invA_i are the i-th matrices in the batch, for i = 1, ..., batch_count.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’
- [in] diag: [rocblas_diagonal] = ‘rocblas_diagonal_non_unit’, A is non-unit triangular; = ‘rocblas_diagonal_unit’, A is unit triangular;
- [in] n: [rocblas_int]
- [in] A: device array of device pointers storing each matrix A_i.
- [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
- [out] invA: device array of device pointers storing the inverse of each matrix A_i. Partial inplace operation is supported, see below. If UPLO = ‘U’, the leading N-by-N upper triangular part of the invA will store the inverse of the upper triangular matrix, and the strictly lower triangular part of invA is cleared. If UPLO = ‘L’, the leading N-by-N lower triangular part of the invA will store the inverse of the lower triangular matrix, and the strictly upper triangular part of invA is cleared.
- [in] ldinvA: [rocblas_int] specifies the leading dimension of each invA_i.
- [in] batch_count: [rocblas_int] numbers of matrices in the batch
rocblas_status rocblas_dtrtri_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const double *const A[], rocblas_int lda, double *invA[], rocblas_int ldinvA, rocblas_int batch_count)

2.12.1.9.2.64 rocblas_<type>trtri_strided_batched()

rocblas_status rocblas_strtri_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const float *A, rocblas_int lda, rocblas_stride stride_a, float *invA, rocblas_int ldinvA, rocblas_stride stride_invA, rocblas_int batch_count)

BLAS Level 3 API.

trtri_strided_batched compute the inverse of A_i and write into invA_i where A_i and invA_i are the i-th matrices in the batch, for i = 1, ..., batch_count

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’
- [in] diag: [rocblas_diagonal] = ‘rocblas_diagonal_non_unit’, A is non-unit triangular; = ‘rocblas_diagonal_unit’, A is unit triangular;
- [in] n: [rocblas_int]
- [in] A: device pointer pointing to address of first matrix A_1.
- [in] lda: [rocblas_int] specifies the leading dimension of each A.
- [in] stride_a: [rocblas_stride] “batch stride a”: stride from the start of one A_i matrix to the next A_(i + 1).
- [out] invA: device pointer storing the inverses of each matrix A_i. Partial inplace operation is supported, see below. If UPLO = ‘U’, the leading N-by-N upper triangular part of the invA will store the inverse of the upper triangular matrix, and the strictly lower triangular part of invA is cleared. If UPLO = ‘L’, the leading N-by-N lower triangular part of the invA will store the inverse of the lower triangular matrix, and the strictly upper triangular part of invA is cleared.
- [in] ldinvA: [rocblas_int] specifies the leading dimension of each invA_i.
- [in] stride_invA: [rocblas_stride] “batch stride invA”: stride from the start of one invA_i matrix to the next invA_(i + 1).
- [in] batch_count: [rocblas_int] numbers of matrices in the batch

rocblas_status rocblas_dtrtri_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const double *A, rocblas_int lda, rocblas_stride stride_a, double *invA, rocblas_int ldinvA, rocblas_stride stride_invA, rocblas_int batch_count)
2.12.1.9.2.65 rocblas_<type>trsm()

rocblas_status rocblas_dtrsm(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, double *B, rocblas_int ldb)

rocblas_status rocblas_strsm(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, float *B, rocblas_int ldb)

BLAS Level 3 API.

trsm solves

\[ \text{op}(A) \times X = \alpha B \quad \text{or} \quad X \times \text{op}(A) = \alpha B, \]

where alpha is a scalar, X and B are m by n matrices, A is triangular matrix and \( \text{op}(A) \) is one of

\[ \text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \quad \text{or} \quad \text{op}(A) = A^H. \]

The matrix X is overwritten on B.

Note about memory allocation: When trsm is launched with a k evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF_TRSM_B_CHNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF_TRSM_B_CHNK be set to the desired chunk of right hand sides to be used at a time.

(where k is m when rocblas_side_left and is n when rocblas_side_right)

Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** side: [rocblas_side] rocblas_side_left: \( \text{op}(A) \times X = \alpha B \). rocblas_side_right: \( X \times \text{op}(A) = \alpha B \).
- **[in]** uplo: [rocblas_fill] rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
- **[in]** transA: [rocblas_operation] transB: \( \text{op}(A) = A \). rocblas_operation_transpose: \( \text{op}(A) = A^T \). rocblas_operation_conjugate_transpose: \( \text{op}(A) = A^H \).
- **[in]** diag: [rocblas_diagonal] rocblas_diagonal_unit: A is assumed to be unit triangular. rocblas_diagonal_non_unit: A is not assumed to be unit triangular.
- **[in]** m: [rocblas_int] m specifies the number of rows of B. m >= 0.
- **[in]** n: [rocblas_int] n specifies the number of columns of B. n >= 0.
- **[in]** alpha: device pointer or host pointer specifying the scalar alpha. When alpha is &zero then A is not referenced and B need not be set before entry.
- **[in]** A: device pointer storing matrix A. of dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.
- **[in]** lda: [rocblas_int] lda specifies the first dimension of A. if side = rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).
• [inout] B: device pointer storing matrix B.
• [in] ldb: [rocblas_int] ldb specifies the first dimension of B. ldb >= max( 1, m ).

2.12.1.9.2.66 rocblas_<type>trsm_batched()

rocblas_status rocblas_strsm_batched (rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const float *alpha, const float *const A[], rocblas_int lda, float *const B[], rocblas_int ldb, rocblas_int batch_count)

BLAS Level 3 API.

trsm_batched performs the following batched operation:

\[
\text{op}(A_i) \times X_i = \alpha \times B_i \quad \text{or} \quad X_i \times \text{op}(A_i) = \alpha \times B_i,
\]

for \( i = 1, \ldots, \text{batch\_count} \).

where \( \alpha \) is a scalar, \( X \) and \( B \) are batched \( m \times n \) matrices, \( A \) is triangular batched matrix and \( \text{op}(A) \) is one of

\[
\text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \quad \text{or} \quad \text{op}(A) = A^H.
\]

Each matrix \( X_i \) is overwritten on \( B_i \) for \( i = 1, \ldots, \text{batch\_count} \).

Note about memory allocation: When trsm is launched with a \( k \) evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF_TRSM_B_CHNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF_TRSM_B_CHNK be set to the desired chunk of right hand sides to be used at a time. (where \( k \) is \( m \) when rocblas_side_left and is \( n \) when rocblas_side_right)

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] side: [rocblas_side] rocblas_side_left: \( \text{op}(A) \times X = \alpha \times B \). rocblas_side_right: \( X \times \text{op}(A) = \alpha \times B \).
• [in] uplo: [rocblas_fill] rocblas_fill_upper: each \( A_i \) is an upper triangular matrix. rocblas_fill_lower: each \( A_i \) is a lower triangular matrix.
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: each \( A_i \) is assumed to be unit triangular. rocblas_diagonal_non_unit: each \( A_i \) is not assumed to be unit triangular.
• [in] m: [rocblas_int] m specifies the number of rows of each \( B_i \). m >= 0.
• [in] n: [rocblas_int] n specifies the number of columns of each \( B_i \). n >= 0.
• [in] alpha: device pointer or host pointer specifying the scalar \( \alpha \). When alpha is &zero then \( A \) is not referenced and \( B \) need not be set before entry.
• [in] A: device array of device pointers storing each matrix \( A_i \) on the GPU. Matricies are of dimension (lda, k), where k is \( m \) when rocblas_side_left and is \( n \) when rocblas_side_right only the upper/lower triangular part is accessed.
• [in] lda: [rocblas_int] lda specifies the first dimension of each \( A_i \). if side = rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).
• [inout] B: device array of device pointers storing each matrix $B_i$ on the GPU.
• [in] ldb: [rocblas_int] ldb specifies the first dimension of each $B_i$. ldb $\geq$ $max(1, m)$.
• [in] batch_count: [rocblas_int] number of trsm operations in the batch.

rocblas_dtrsm_batched

rocblas_status \texttt{rocblas_dtrsm\_batched}(rocblas\_handle handle, \rocblas\_side side, \rocblas\_fill uplo, 
rocblas\_operation transA, rocblas\_diagonal diag, rocblas\_int m, rocblas\_int n, \textbf{const double} *alpha, const double *\textbf{const A}[], rocblas\_int lda, double *B[], rocblas\_int ldb, 
rocblas\_int batch\_count)

2.12.1.9.2.67 rocblas_<type>trsm_strided_batched()

rocblas_strsm_strided_batched

rocblas_status \texttt{rocblas\_strsm\_strided\_batched}(rocblas\_handle handle, \rocblas\_side side, \rocblas\_fill uplo, 
rocblas\_operation transA, rocblas\_diagonal diag, rocblas\_int m, rocblas\_int n, \textbf{const float} *alpha, const float *A, 
rocblas\_int lda, rocblas\_stride stride\_a, float *B, rocblas\_int ldb, rocblas\_stride stride\_b, 
rocblas\_int batch\_count)

BLAS Level 3 API.

trsm_strided_batched performs the following strided batched operation:

\[
\text{op}(A_i) \cdot X_i = \alpha \cdot B_i \quad \text{or} \quad X_i \cdot \text{op}(A_i) = \alpha \cdot B_i, \quad \text{for } i = 1, \ldots, \text{batch\_count}.
\]

where alpha is a scalar, X and B are strided batched m by n matrices, A is triangular strided batched matrix and 
\text{op}(A) is one of

\[
\text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \quad \text{or} \quad \text{op}(A) = A^H.
\]

Each matrix $X_i$ is overwritten on $B_i$ for $i = 1, \ldots, \text{batch\_count}$.

Note about memory allocation: When trsm is launched with a k evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF_TRSM_B_CHNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF_TRSM_B_CHNK be set to the desired chunk of right hand sides to be used at a time. (where k is m when rocblas\_side\_left and is n when rocblas\_side\_right)

Parameters

• [in] handle: [rocblas\_handle] handle to the rocblas library context queue.
• [in] side: [rocblas\_side] rocblas\_side\_left: \text{op}(A) \cdot X = \alpha \cdot B. \ \text{rocblas\_side\_right}: X \cdot \text{op}(A) = \alpha \cdot B.
• [in] uplo: [rocblas\_fill] rocblas\_fill\_upper: each $A_i$ is an upper triangular matrix. rocblas\_fill\_lower: each $A_i$ is a lower triangular matrix.
• [in] transA: [rocblas\_operation] transB: \text{op}(A) = A. rocblas\_operation\_transpose: \text{op}(A) = A^T. rocblas\_operation\_conjugate\_transpose: \text{op}(A) = A^H.
• [in] diag: [rocblas\_diagonal] rocblas\_diagonal\_unit: each $A_i$ is assumed to be unit triangular. rocblas\_diagonal\_non\_unit: each $A_i$ is not assumed to be unit triangular.
• [in] m: [rocblas\_int] m specifies the number of rows of each $B_i$. $m \geq 0$. 

2.12. ROCm Libraries 137
• [in] \( n \): [rocblas_int] \( n \) specifies the number of columns of each \( B_i \). \( n \geq 0 \).

• [in] \( \alpha \): device pointer or host pointer specifying the scalar \( \alpha \). When \( \alpha \) is \( \& \)zero then \( A \) is not referenced and \( B \) need not be set before entry.

• [in] \( A \): device pointer pointing to the first matrix \( A_1 \) of dimension \(( lda, k )\), where \( k \) is \( m \) when \( \text{rocblas} \_\text{side} = \text{rocblas} \_\text{side} \_\text{left} \) and is \( n \) when \( \text{rocblas} \_\text{side} = \text{rocblas} \_\text{side} \_\text{right} \) only the upper/lower triangular part is accessed.

• [in] \( lda \): [rocblas_int] \( lda \) specifies the first dimension of each \( A_1 \). if \( side = \text{rocblas} \_\text{side} \_\text{left} \), \( lda \geq \max( 1, m ) \), if \( side = \text{rocblas} \_\text{side} \_\text{right} \), \( lda \geq \max( 1, n ) \).

• [in] \( \text{stride} _a \): [rocblas_stride] \( \text{stride} \) from the start of one \( A_i \) matrix to the next \( A_{i+1} \).

• [inout] \( B \): device pointer pointing to the first matrix \( B_1 \).

• [in] \( ldb \): [rocblas_int] \( ldb \) specifies the first dimension of each \( B_1 \). \( ldb \geq \max( 1, m ) \).

• [in] \( \text{stride} _b \): [rocblas_stride] \( \text{stride} \) from the start of one \( B_i \) matrix to the next \( B_{i+1} \).

• [in] \( \text{batch} \_count \): [rocblas_int] number of \( \text{trsm} \) operations in the batch.

\[
\text{rocblas} \_\text{status} \ \text{rocblas} \_\text{dtrsm} \_\text{strided} \_\text{batched} \left( \begin{array}{c}
\text{rocblas} \_\text{handle} \ \text{handle} \\
\text{rocblas} \_\text{side} \ \text{side} \\
\text{rocblas} \_\text{fill} \ \text{uplo} \\
\text{rocblas} \_\text{operation} \ \text{transA} \\
\text{rocblas} \_\text{diagonal} \ \text{diag} \\
\text{rocblas} \_\text{int} \ m \\
\text{rocblas} \_\text{int} \ n \\
\text{const} \ \text{double} \ \* \alpha \\
\text{const} \ \text{double} \ \*A \\
\text{rocblas} \_\text{int} \ lda \\
\text{rocblas} \_\text{stride} _a \\
\text{double} \ \*B \\
\text{rocblas} \_\text{int} \ ldb \\
\text{rocblas} \_\text{stride} _b \\
\text{rocblas} \_\text{int} \ \text{batch} \_count
\end{array} \right)
\]

2.12.1.9.2.68 rocblas\_<type>trmm()

\[
\text{rocblas} \_\text{status} \ \text{rocblas} \_\text{strmm} \left( \begin{array}{c}
\text{rocblas} \_\text{handle} \ \text{handle} \\
\text{rocblas} \_\text{side} \ \text{side} \\
\text{rocblas} \_\text{fill} \ \text{uplo} \\
\text{rocblas} \_\text{operation} \ \text{transA} \\
\text{rocblas} \_\text{diagonal} \ \text{diag} \\
\text{rocblas} \_\text{int} \ m \\
\text{rocblas} \_\text{int} \ n \\
\text{const} \ \text{float} \ \* \alpha \\
\text{const} \ \text{float} \ \*A \\
\text{rocblas} \_\text{int} \ lda \\
\text{rocblas} \_\text{int} \ ldb
\end{array} \right)
\]

BLAS Level 3 API.

\( \text{trmm} \) performs one of the matrix-matrix operations

\[
B := \alpha \text{op}( A ) \text{B}, \text{or } B := \alpha \text{B} \text{op}( A )
\]

where \( \alpha \) is a scalar, \( B \) is an \( m \) by \( n \) matrix, \( A \) is a unit, or non-unit, upper or lower triangular matrix and \( \text{op}( A ) \) is one of

\[
\text{op}( A ) = A \quad \text{or} \quad \text{op}( A ) = A^T \quad \text{or} \quad \text{op}( A ) = A^H.
\]

Parameters

• [in] \( \text{handle} \): [rocblas\_handle] handle to the rocblas library context queue.

• [in] \( \text{side} \): [rocblas\_side] \( \text{side} \_\text{left} \): \( C := \alpha \text{op}( A ) \text{B} \). \( \text{side} \_\text{right} \): \( C := \alpha \text{B} \text{op}( A ) \).

• [in] \( \text{uplo} \): [rocblas\_fill] \( \text{uplo} \_\text{upper} \): \( A \) is an upper triangular matrix. \( \text{uplo} \_\text{lower} \): \( A \) is a lower triangular matrix.

• [in] \( \text{transA} \): [rocblas\_operation] \( \text{transB} \): \( \text{op}( A ) = A \). \( \text{transA} \_\text{transpose} \): \( \text{op}( A ) = A^T \). \( \text{transA} \_\text{conjugate} \_\text{transpose} \): \( \text{op}( A ) = A^H \).

• [in] \( \text{diag} \): [rocblas\_diagonal] \( \text{diag} \_\text{unit} \): \( A \) is assumed to be unit triangular. \( \text{diag} \_\text{non} \_\text{unit} \): \( A \) is not assumed to be unit triangular.
• [in] m: [rocblas_int] m specifies the number of rows of B. m >= 0.
• [in] n: [rocblas_int] n specifies the number of columns of B. n >= 0.
• [in] alpha: alpha specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry.
• [in] A: pointer storing matrix A on the GPU. of dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.
• [in] lda: [rocblas_int] lda specifies the first dimension of A. if side = rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).
• [in] B: pointer storing matrix B on the GPU.
• [in] ldb: [rocblas_int] ldb specifies the first dimension of B. ldb >= max( 1, m ).

rocblas_status rocblas_dtrmm(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, double *B, rocblas_int ldb)

2.12.1.9.2.69 rocblas_<type>gemm()  

rocblas_status rocblas_dgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *A, rocblas_int lda, const double *B, rocblas_int ldb, const double *C, rocblas_int ldc)
rocblas_status rocblas_sgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A, rocblas_int lda, const float *B, rocblas_int ldb, const float *C, rocblas_int ldc)

BLAS Level 3 API.
xGEMM performs one of the matrix-matrix operations

\[
C = \alpha \times \text{op}( A ) \times \text{op}( B ) + \beta \times C,
\]

where \( \text{op}( X ) \) is one of

- \( \text{op}( X ) = X \)  
or
- \( \text{op}( X ) = X^T \)  
or
- \( \text{op}( X ) = X^H \),

alpha and beta are scalars, and A, B and C are matrices, with \( \text{op}( A ) \) an m by k matrix, \( \text{op}( B ) \) a k by n matrix and C an m by n matrix.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] transA: [rocblas_operation] specifies the form of \( \text{op}( A ) \)
• [in] transB: [rocblas_operation] specifies the form of \( \text{op}( B ) \)
• [in] m: [rocblas_int] number or rows of matrices \( \text{op}( A ) \) and C
• [in] n: [rocblas_int] number of columns of matrices \( \text{op}( B ) \) and C
• [in] k: [rocblas_int] number of columns of matrix \( \text{op}( A ) \) and number of rows of matrix \( \text{op}( B ) \)
- [in] alpha: device pointer or host pointer specifying the scalar alpha.
- [in] A: device pointer storing matrix A.
- [in] lda: [rocblas_int] specifies the leading dimension of A.
- [in] B: device pointer storing matrix B.
- [in] ldb: [rocblas_int] specifies the leading dimension of B.
- [in] beta: device pointer or host pointer specifying the scalar beta.
- [inout] C: device pointer storing matrix C on the GPU.
- [in] ldc: [rocblas_int] specifies the leading dimension of C.

```
rocblas_status rocblas_hgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *A, rocblas_int lda, const rocblas_half *B, rocblas_int ldb, const rocblas_half *beta, rocblas_half *C, rocblas_int ldc)
```

```
rocblas_status rocblas_cgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *B, rocblas_int ldb, const rocblas_float_complex *beta, rocblas_float_complex *C, rocblas_int ldc)
```

```
rocblas_status rocblas_zgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, const rocblas_double_complex *B, rocblas_int ldb, const rocblas_double_complex *beta, rocblas_double_complex *C, rocblas_int ldc)
```

### 2.12.1.9.2.70 rocblas_<type>gemm_batched()

```
rocblas_status rocblas_sgemm_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A[], rocblas_int lda, const float *const B[], rocblas_int ldb, const float *beta, float *const C[], rocblas_int ldc, rocblas_int batch_count)
```

BLAS Level 3 API.

xGEMM_BATCHED performs one of the batched matrix-matrix operations $C_{i} = \alpha \cdot \text{op}(A_{i}) \cdot \text{op}(B_{i}) + \beta \cdot C_{i}$, for $i = 1, \ldots, \text{batch\_count}$. where $\text{op}(X)$ is one of $\text{op}(X) = X$, $\text{op}(X) = X^\text{T}$ or $\text{op}(X) = X^\text{H}$, $\alpha$ and $\beta$ are scalars, and $A$, $B$ and $C$ are strided batched matrices, with $\text{op}(A)$ an $m$ by $k$ by $\text{batch\_count}$ strided\_batched matrix, $\text{op}(B)$ an $k$ by $n$ by $\text{batch\_count}$ strided\_batched matrix and $C$ an $m$ by $n$ by $\text{batch\_count}$ strided\_batched matrix.

#### Parameters
- [in] handle: [rocblas_handle handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of $\text{op}(A)$
- [in] transB: [rocblas_operation] specifies the form of $\text{op}(B)$
- [in] m: [rocblas_int] matrix dimension $m$.

140 Chapter 2. Solid Compilation Foundation and Language Support
- **[in]** `k`: [rocblas_int] matrix dimension `k`.
- **[in]** `alpha`: device pointer or host pointer specifying the scalar `alpha`.
- **[in]** `A`: device array of device pointers storing each matrix `A_i`.
- **[in]** `lda`: [rocblas_int] specifies the leading dimension of each `A_i`.
- **[in]** `B`: device array of device pointers storing each matrix `B_i`.
- **[in]** `ldb`: [rocblas_int] specifies the leading dimension of each `B_i`.
- **[in]** `beta`: device pointer or host pointer specifying the scalar `beta`.
- **[inout]** `C`: device array of device pointers storing each matrix `C_i`.
- **[in]** `ldc`: [rocblas_int] specifies the leading dimension of each `C_i`.
- **[in]** `batch_count`: [rocblas_int] number of gemm operations in the batch

```
rocblas_status rocblas_dgemm_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *const A[], rocblas_int lda, const double *const B[], rocblas_int ldb, const double *beta, double *const C[], rocblas_int ldc, rocblas_int batch_count)
```

```
rocblas_status rocblas_hgemm_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *const A[], rocblas_int lda, const rocblas_half *const B[], rocblas_int ldb, const rocblas_half *beta, rocblas_half *const C[], rocblas_int ldc, rocblas_int batch_count)
```

```
rocblas_status rocblas_cgemm_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_float_complex *alpha, const rocblas_float_complex *const A[], rocblas_int lda, const rocblas_float_complex *const B[], rocblas_int ldb, const rocblas_float_complex *beta, rocblas_float_complex *const C[], rocblas_int ldc, rocblas_int batch_count)
```

```
rocblas_status rocblas_zgemm_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_double_complex *alpha, const rocblas_double_complex *const A[], rocblas_int lda, const rocblas_double_complex *const B[], rocblas_int ldb, const rocblas_double_complex *beta, rocblas_double_complex *const C[], rocblas_int ldc, rocblas_int batch_count)
```
2.12.1.9.2.71 rocblas_<type>gemm_strided_batched()

rocblas_status rocblas_dgemm_strided_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *A, rocblas_int lda, rocblas_stride stride_a, const double *B, rocblas_int ldb, rocblas_stride stride_b, const double *beta, double *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_sgemm_strided_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A, rocblas_int lda, rocblas_stride stride_a, const float *B, rocblas_int ldb, rocblas_stride stride_b, const float *beta, float *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

BLAS Level 3 API.

xGEMM_STRIDED_BATCHED performs one of the strided batched matrix-matrix operations

\[ C_i = \alpha \cdot \text{op}(A_i) \cdot \text{op}(B_i) + \beta \cdot C_i, \text{ for } i = 1, \ldots, \text{batch\_count} \]

where \( \text{op}(X) \) is one of

\[
\begin{align*}
\text{op}(X) &= X \quad \text{or} \\
\text{op}(X) &= X^T \quad \text{or} \\
\text{op}(X) &= X^H,
\end{align*}
\]

alpha and beta are scalars, and A, B and C are strided batched matrices, with \( \text{op}(A) \) an \( m \) by \( k \) by \( \text{batch\_count} \) strided_batched matrix, \( \text{op}(B) \) an \( k \) by \( n \) by \( \text{batch\_count} \) strided_batched matrix and C an \( m \) by \( n \) by \( \text{batch\_count} \) strided_batched matrix.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of \( \text{op}(A) \)
- [in] transB: [rocblas_operation] specifies the form of \( \text{op}(B) \)
- [in] m: [rocblas_int] matrix dimension m.
- [in] k: [rocblas_int] matrix dimension k.
- [in] alpha: device pointer or host pointer specifying the scalar alpha.
- [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
- [in] stride_a: [rocblas_stride] stride from the start of one A_i matrix to the next A_(i + 1).
- [in] B: device pointer pointing to the first matrix B_1.
- [in] ldb: [rocblas_int] specifies the leading dimension of each B_i.
- [in] stride_b: [rocblas_stride] stride from the start of one B_i matrix to the next B_(i + 1).
• [in] beta: device pointer or host pointer specifying the scalar beta.
• [inout] C: device pointer pointing to the first matrix C_1.
• [in] ldc: [rocblas_int] specifies the leading dimension of each C_i.
• [in] stride_c: [rocblas_stride] stride from the start of one C_i matrix to the next C_(i + 1).
• [in] batch_count: [rocblas_int] number of gemm operators in the batch

rocblas_status rocblas_hgemm_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_half *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_half *beta, rocblas_half *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_cgemm_strided_batched (rocblas_handle handle, rocblas_operation transa, rocblas_operation transb, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_float_complex *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_float_complex *beta, rocblas_float_complex *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_zgemm_strided_batched (rocblas_handle handle, rocblas_operation transa, rocblas_operation transb, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_double_complex *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_double_complex *beta, rocblas_double_complex *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

2.12.1.9.2.72 rocblas_<type>gemm_kernel_name()

rocblas_status rocblas_dgemm_kernel_name (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *A, rocblas_int lda, rocblas_stride stride_a, const double *B, rocblas_int ldb, rocblas_stride stride_b, const double *beta, double *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)
rocblas_status rocblas_sgemm_kernel_name(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A, rocblas_int lda, rocblas_stride stride_a, const float *B, rocblas_int ldb, rocblas_stride stride_b, const float *beta, float *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_hgemm_kernel_name(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_half *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_half *beta, rocblas_half *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

2.12.1.9.2.73 rocblas_<type>geam()

rocblas_status rocblas_dgeam(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, const double *beta, const double *B, rocblas_int ldb, double *C, rocblas_int ldc)

rocblas_status rocblas_sgeam(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, const float *beta, const float *B, rocblas_int ldb, float *C, rocblas_int ldc)

BLAS Level 3 API.

xGEAM performs one of the matrix-matrix operations

\[ C = \alpha \text{op}( A ) + \beta \text{op}( B ), \]

where \text{op}( X ) is one of

\[
\begin{align*}
\text{op}( X ) &= X &\text{or} \\
\text{op}( X ) &= X^{\ast}\text{T} &\text{or} \\
\text{op}( X ) &= X^{\ast}\text{H}
\end{align*}
\]

alpha and beta are scalars, and A, B and C are matrices, with \text{op}( A ) an m by n matrix, \text{op}( B ) an m by n matrix, and C an m by n matrix.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of \text{op}( A )
- [in] transB: [rocblas_operation] specifies the form of \text{op}( B )
- [in] m: [rocblas_int] matrix dimension m.
- [in] alpha: device pointer or host pointer specifying the scalar alpha.
- [in] A: device pointer storing matrix A.
- [in] lda: [rocblas_int] specifies the leading dimension of A.
• [in] beta: device pointer or host pointer specifying the scalar beta.
• [in] B: device pointer storing matrix B.
• [in] ldb: [rocblas_int] specifies the leading dimension of B.
• [inout] C: device pointer storing matrix C.
• [in] ldc: [rocblas_int] specifies the leading dimension of C.

2.12.1.9.2.74 BLAS Extensions

2.12.1.9.2.75 rocblas_gemm_ex()

rocblas_status rocblas_gemm_ex (rocblas_handle handle, rocblas_operation transA, rocblas_operation
transB, rocblas_int m, rocblas_int n, rocblas_int k, const void *alpha, const void *a,
rocblas_datatype a_type, rocblas_int lda, const void *b, rocblas_datatype b_type, rocblas_int ldb,
const void *beta, const void *c, rocblas_datatype c_type, rocblas_int ldc, void *
d, rocblas_datatype d_type, rocblas_int ldd, rocblas_datatype compute_type,
rocblas_gemm_algo algo, int32_t solution_index, uint32_t flags)

BLAS EX API.

GEMM_EX performs one of the matrix-matrix operations

\[ D = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C, \]

where \( \text{op}(X) \) is one of

\[ \text{op}(X) = X \text{ or } X^T \text{ or } X^H, \]

alpha and beta are scalars, and A, B, C, and D are matrices, with \( \text{op}(A) \) an m by k matrix, \( \text{op}(B) \) a k by n
matrix and C and D are m by n matrices.

Supported types are as follows:

- rocblas_datatype_f64_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f32_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f16_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f16_r = a_type = b_type = c_type = d_type; rocblas_datatype_f32_r = compute_type
- rocblas_datatype_bf16_r = a_type = b_type = c_type = d_type; rocblas_datatype_f32_r = compute_type
- rocblas_datatype_i8_r = a_type = b_type; rocblas_datatype_i32_r = c_type = d_type = compute_type
- rocblas_datatype_f32_c = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f64_c = a_type = b_type = c_type = d_type = compute_type

Below are restrictions for rocblas_datatype_i8_r = a_type = b_type; rocblas_datatype_i32_r = c_type = d_type
= compute_type:

- k must be a multiple of 4
- lda must be a multiple of 4 if transA == rocblas_operation_transpose
- ldb must be a multiple of 4 if transB == rocblas_operation_none
for transA == rocblas_operation_transpose or transB == rocblas_operation_none the matrices A and B must have each 4 consecutive values in the k dimension packed. This packing can be achieved with the following pseudo-code. The code assumes the original matrices are in A and B, and the packed matrices are A_packed and B_packed. The size of the A_packed matrix is the same as the size of the A matrix, and the size of the B_packed matrix is the same as the size of the B matrix.

```c
if (transA == rocblas_operation_none)
{
    int nb = 4;
    for(int i_m = 0; i_m < m; i_m++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            A_packed[i_k % nb + (i_m + (i_k / nb) * lda) * nb] = A[i_m + i_k * lda];
        }
    }
}
else
{
    A_packed = A;
}
if (transB == rocblas_operation_transpose)
{
    int nb = 4;
    for(int i_n = 0; i_n < m; i_n++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            B_packed[i_k % nb + (i_n + (i_k / nb) * lda) * nb] = B[i_n + i_k * lda];
        }
    }
}
else
{
    B_packed = B;
}
```

Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** transA: [rocblas_operation] specifies the form of op( A ).
- **[in]** transB: [rocblas_operation] specifies the form of op( B ).
- **[in]** m: [rocblas_int] matrix dimension m.
- **[in]** n: [rocblas_int] matrix dimension n.
- **[in]** k: [rocblas_int] matrix dimension k.
- **[in]** alpha: [const void *] device pointer or host pointer specifying the scalar alpha. Same datatype as compute_type.
- **[in]** a: [void *] device pointer storing matrix A.
- **[in]** a_type: [rocblas_datatype] specifies the datatype of matrix A.
- **[in]** lda: [rocblas_int] specifies the leading dimension of A.
• [in] b: [void *] device pointer storing matrix B.
• [in] b_type: [rocblas_datatype] specifies the datatype of matrix B.
• [in] ldb: [rocblas_int] specifies the leading dimension of B.
• [in] beta: [const void *] device pointer or host pointer specifying the scalar beta. Same datatype as compute_type.
• [in] c: [void *] device pointer storing matrix C.
• [in] c_type: [rocblas_datatype] specifies the datatype of matrix C.
• [in] ldc: [rocblas_int] specifies the leading dimension of C.
• [out] d: [void *] device pointer storing matrix D.
• [in] d_type: [rocblas_datatype] specifies the datatype of matrix D.
• [in] ldd: [rocblas_int] specifies the leading dimension of D.
• [in] compute_type: [rocblas_datatype] specifies the datatype of computation.
• [in] algo: [rocblas_gemm_algo] enumerant specifying the algorithm type.
• [in] solution_index: [int32_t] reserved for future use.

2.12.1.9.2.76 rocblas_gemm_strided_batched_ex()

rocblas_status rocblas_gemm_strided_batched_ex (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const void *alpha, const void *a, rocblas_datatype a_type, rocblas_int lda, rocblas_stride stride_a, const void *b, rocblas_datatype b_type, rocblas_int ldb, rocblas_stride stride_b, const void *beta, const void *c, rocblas_datatype c_type, rocblas_int ldc, rocblas_stride stride_c, void *d, rocblas_datatype d_type, rocblas_int ldd, rocblas_stride stride_d, rocblas_int batch_count, rocblas_datatype compute_type, rocblas_gemm_algo algo, int32_t solution_index, uint32_t flags)

BLAS EX API.

GEMM_STRIDED_BATCHED_EX performs one of the strided_batched matrix-matrix operations

\[ D_i = \alpha \cdot \text{op}(A_i) \cdot \text{op}(B_i) + \beta \cdot C_i, \quad \text{for} \ i = 1, \ldots, \ \text{batch\_count} \]

where \( \text{op}(X) \) is one of

- \( \text{op}(X) = X \)
- \( \text{op}(X) = X^T \)
- \( \text{op}(X) = X^H \)

alpha and beta are scalars, and A, B, C, and D are strided_batched matrices, with \( \text{op}(A) \) an m by k by batch_count strided_batched matrix, \( \text{op}(B) \) a k by n by batch_count strided_batched matrix and C and D are m by n by batch_count strided_batched matrices.
The strided_batched matrices are multiple matrices separated by a constant stride. The number of matrices is \texttt{batch\_count}.

Supported types are as follows:

- \texttt{rocblas\_datatype\_f64\_r = a\_type = b\_type = c\_type = d\_type = compute\_type}
- \texttt{rocblas\_datatype\_f32\_r = a\_type = b\_type = c\_type = d\_type = compute\_type}
- \texttt{rocblas\_datatype\_f16\_r = a\_type = b\_type = c\_type = d\_type = compute\_type}
- \texttt{rocblas\_datatype\_bf16\_r = a\_type = b\_type = c\_type = d\_type; rocblas\_datatype\_f32\_r = compute\_type}
- \texttt{rocblas\_datatype\_i8\_r = a\_type = b\_type; rocblas\_datatype\_i32\_r = c\_type = d\_type = compute\_type}
- \texttt{rocblas\_datatype\_f32\_c = a\_type = b\_type = c\_type = d\_type = compute\_type}
- \texttt{rocblas\_datatype\_f64\_c = a\_type = b\_type = c\_type = d\_type = compute\_type}

Below are restrictions for \texttt{rocblas\_datatype\_i8\_r = a\_type = b\_type; rocblas\_datatype\_i32\_r = c\_type = d\_type = compute\_type}:

- \texttt{k} must be a multiple of 4
- \texttt{lda} must be a multiple of 4 if \texttt{transA == rocblas\_operation\_transpose}
- \texttt{ldb} must be a multiple of 4 if \texttt{transB == rocblas\_operation\_none}
- for \texttt{transA == rocblas\_operation\_transpose} or \texttt{transB == rocblas\_operation\_none} the matrices A and B must have each 4 consecutive values in the \texttt{k} dimension packed. This packing can be achieved with the following pseudo-code. The code assumes the original matrices are in A and B, and the packed matrices are A\_packed and B\_packed. The size of the A\_packed matrix is the same as the size of the A matrix, and the size of the B\_packed matrix is the same as the size of the B matrix.

```c
if(transA == rocblas\_operation\_none)
{
    int nb = 4;
    for(int i_m = 0; i_m < m; i_m++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            A\_packed[i_k % nb + (i_m + (i_k / nb) * lda) * nb] = A[i_m + i_k * lda];
        }
    }
}
else
{
    A\_packed = A;
}
if(transB == rocblas\_operation\_transpose)
{
    int nb = 4;
    for(int i_n = 0; i_n < m; i_n++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            B\_packed[i_k % nb + (i_n + (i_k / nb) * lda) * nb] = B[i_n + i_k * lda];
        }
    }
}
```

(continues on next page)
else
{
    B_packed = B;
}

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of \( \text{op}(A) \).
- [in] transB: [rocblas_operation] specifies the form of \( \text{op}(B) \).
- [in] m: [rocblas_int] matrix dimension \( m \).
- [in] n: [rocblas_int] matrix dimension \( n \).
- [in] k: [rocblas_int] matrix dimension \( k \).
- [in] alpha: [const void *] device pointer or host pointer specifying the scalar \( \alpha \). Same datatype as compute_type.
- [in] a: [void *] device pointer pointing to first matrix \( A_1 \).
- [in] a_type: [rocblas_datatype] specifies the datatype of each matrix \( A_i \).
- [in] lda: [rocblas_int] specifies the leading dimension of each \( A_i \).
- [in] stride_a: [rocblas_stride] specifies stride from start of one \( A_i \) matrix to the next \( A_{(i+1)} \).
- [in] b: [void *] device pointer pointing to first matrix \( B_1 \).
- [in] b_type: [rocblas_datatype] specifies the datatype of each matrix \( B_i \).
- [in] ldb: [rocblas_int] specifies the leading dimension of each \( B_i \).
- [in] stride_b: [rocblas_stride] specifies stride from start of one \( B_i \) matrix to the next \( B_{(i+1)} \).
- [in] beta: [const void *] device pointer or host pointer specifying the scalar \( \beta \). Same datatype as compute_type.
- [in] c: [void *] device pointer pointing to first matrix \( C_1 \).
- [in] c_type: [rocblas_datatype] specifies the datatype of each matrix \( C_i \).
- [in] ldc: [rocblas_int] specifies the leading dimension of each \( C_i \).
- [in] stride_c: [rocblas_stride] specifies stride from start of one \( C_i \) matrix to the next \( C_{(i+1)} \).
- [out] d: [void *] device pointer storing each matrix \( D_i \).
- [in] d_type: [rocblas_datatype] specifies the datatype of each matrix \( D_i \).
- [in] ldd: [rocblas_int] specifies the leading dimension of each \( D_i \).
- [in] stride_d: [rocblas_stride] specifies stride from start of one \( D_i \) matrix to the next \( D_{(i+1)} \).
- [in] batch_count: [rocblas_int] number of gemm operations in the batch.
- [in] compute_type: [rocblas_datatype] specifies the datatype of computation.
• [in] algo: [rocblas_gemm_algo] enumerant specifying the algorithm type.
• [in] solution_index: [int32_t] reserved for future use.

2.12.1.9.2.77 rocblas_trsm_ex()

rocblas_status rocblas_trsm_ex(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const void *alpha, const void *A, rocblas_int lda, void *B, rocblas_int ldb, const void *invA, rocblas_int invA_size, rocblas_datatype compute_type)

BLAS EX API
TRSM_EX solves

\[
\text{op}(A) \times X = \alpha B \quad \text{or} \quad X \times \text{op}(A) = \alpha B,
\]

where \( \alpha \) is a scalar, \( X \) and \( B \) are \( m \) by \( n \) matrices, \( A \) is triangular matrix and \( \text{op}(A) \) is one of

\[
\text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \quad \text{or} \quad \text{op}(A) = A^H.
\]

The matrix \( X \) is overwritten on \( B \).

TRSM_EX gives the user the ability to reuse the invA matrix between runs. If invA == NULL, rocblas_trsm_ex will automatically calculate invA on every run.

Setting up invA: The accepted invA matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix \( A \), followed by any smaller diagonal block that remains. To set up invA it is recommended that rocblas_trtri_batched be used with matrix \( A \) as the input.

Device memory of size 128 x \( k \) should be allocated for invA ahead of time, where \( k \) is \( m \) when rocblas_side_left and is \( n \) when rocblas_side_right. The actual number of elements in invA should be passed as invA_size.

To begin, rocblas_trtri_batched must be called on the full 128x128 sized diagonal blocks of matrix \( A \). Below are the restricted parameters:

- \( n = 128 \)
- \( \text{ldinvA} = 128 \)
- \( \text{stride}_\text{invA} = 128 \times 128 \)
- \( \text{batch}_\text{count} = k / 128 \)

Then any remaining block may be added:

- \( n = k \% 128 \)
- \( \text{invA} = \text{invA} + \text{stride}_\text{invA} \times \text{previous}_\text{batch}_\text{count} \)
- \( \text{ldinvA} = 128 \)
- \( \text{batch}_\text{count} = 1 \)

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] side: [rocblas_side] rocblas_side_left: \( \text{op}(A) \times X = \alpha B \). rocblas_side_right: \( X \times \text{op}(A) = \alpha B \).
ReadTheDocs-Breathe Documentation, Release 1.0.0

- **[in]** `uplo`: [rocblas_fill] rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.

- **[in]** `transA`: [rocblas_operation] transB: op(A) = A. rocblas_operation_transpose: op(A) = A^T. rocblas_operation_conjugate_transpose: op(A) = A^H.

- **[in]** `diag`: [rocblas_diagonal] rocblas_diagonal_unit: A is assumed to be unit triangular. rocblas_diagonal_non_unit: A is not assumed to be unit triangular.

- **[in]** `m`: [rocblas_int] `m` specifies the number of rows of B. m >= 0.

- **[in]** `n`: [rocblas_int] `n` specifies the number of columns of B. n >= 0.

- **[in]** `alpha`: [void *] device pointer or host pointer specifying the scalar alpha. When alpha is &zero then A is not referenced, and B need not be set before entry.

- **[in]** `A`: [void *] device pointer storing matrix A. of dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.

- **[in]** `lda`: [rocblas_int] `lda` specifies the first dimension of A. if side = rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).

- **[inout]** `B`: [void *] device pointer storing matrix B. B is of dimension ( ldb, n ). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

- **[in]** `ldb`: [rocblas_int] `ldb` specifies the first dimension of B. ldb >= max( 1, m ).

- **[in]** `invA`: [void *] device pointer storing the inverse diagonal blocks of A. invA is of dimension ( ld_invA, k ), where k is m when rocblas_side_left and is n when rocblas_side_right. ld_invA must be equal to 128.

- **[in]** `invA_size`: [rocblas_int] `invA_size` specifies the number of elements of device memory in invA.

- **[in]** `compute_type`: [rocblas_datatype] specifies the datatype of computation

### 2.12.1.9.2.78 rocblas_trsm_batched_ex()

- **rocblas_status rocblas_trsm_batched_ex** (rocblas_handle `handle`, rocblas_side `side`, rocblas_fill `uplo`, rocblas_operation `transA`, rocblas_diagonal `diag`, rocblas_int `m`, rocblas_int `n`, const void *`alpha`, const void *`A`, rocblas_int `lda`, void *`B`, rocblas_int `ldb`, rocblas_int `batch_count`, const void *`invA`, rocblas_int `invA_size`, rocblas_datatype `compute_type`)

**BLAS EX API**

**TRSM_BATCHED_EX** solves

\[ \text{op}(A_i) \times X_i = \alpha \times B_i \quad \text{or} \quad X_i \times \text{op}(A_i) = \alpha \times B_i, \]

for i = 1, ..., batch_count; and where alpha is a scalar, X and B are arrays of m by n matrices, A is an array of triangular matrix and each op(A_i) is one of

\[ \text{op}( A_i ) = A_i \quad \text{or} \quad \text{op}( A_i ) = A_i^T \quad \text{or} \quad \text{op}( A_i ) = A_i^H. \]

Each matrix X_i is overwritten on B_i.

**TRSM_EX** gives the user the ability to reuse the invA matrix between runs. If invA == NULL, rocblas_trsm_batched_ex will automatically calculate each invA_i on every run.

2.12. ROCm Libraries 151
Setting up invA: Each accepted invA_i matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix A_i, followed by any smaller diagonal block that remains. To set up each invA_i it is recommended that rocblas_trtri_batched be used with matrix A_i as the input. invA is an array of pointers of batch_count length holding each invA_i.

Device memory of size 128 x k should be allocated for each invA_i ahead of time, where k is m when rocblas_side_left and is n when rocblas_side_right. The actual number of elements in each invA_i should be passed as invA_size.

To begin, rocblas_trtri_batched must be called on the full 128x128 sized diagonal blocks of each matrix A_i. Below are the restricted parameters:

- \( n = 128 \)
- \( \text{ldinvA} = 128 \)
- \( \text{stride_invA} = 128 \times 128 \)
- \( \text{batch_count} = k / 128 \)

Then any remaining block may be added:

- \( n = k \mod 128 \)
- \( \text{invA} = \text{invA} + \text{stride_invA} \times \text{previous_batch_count} \)
- \( \text{ldinvA} = 128 \)
- \( \text{batch_count} = 1 \)

Parameters

- \([\text{in}]\, \text{handle} : \text{rocblas_handle}\) handle to the rocblas library context queue.
- \([\text{in}]\, \text{side} : \text{rocblas_side}\) rocblas_side_left: \( \text{op(A)} \times X = \alpha \times \text{B} \). rocblas_side_right: \( X \times \text{op(A)} = \alpha \times \text{B} \).
- \([\text{in}]\, \text{uplo} : \text{rocblas_fill}\) rocblas_fill_upper: each A_i is an upper triangular matrix. rocblas_fill_lower: each A_i is a lower triangular matrix.
- \([\text{in}]\, \text{transA} : \text{rocblas_operation}\) transB: \( \text{op(A)} = A \). rocblas_operation_transpose: \( \text{op(A)} = A^\text{T} \). rocblas_operation_conjugate_transpose: \( \text{op(A)} = A^\text{H} \).
- \([\text{in}]\, \text{diag} : \text{rocblas_diagonal}\) rocblas_diagonal_unit: each A_i is assumed to be unit triangular. rocblas_diagonal_non_unit: each A_i is not assumed to be unit triangular.
- \([\text{in}]\, m : \text{rocblas_int}\) m specifies the number of rows of each B_i. \( m \geq 0 \).
- \([\text{in}]\, n : \text{rocblas_int}\) n specifies the number of columns of each B_i. \( n \geq 0 \).
- \([\text{in}]\, \alpha : \text{void *}\) device pointer or host pointer alpha specifying the scalar alpha. When alpha is &zero then A is not referenced, and B need not be set before entry.
- \([\text{in}]\, A : \text{void *}\) device array of device pointers storing each matrix A_i. each A_i is of dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.
- \([\text{in}]\, \text{lda} : \text{rocblas_int}\) lda specifies the first dimension of each A_i. if side = rocblas_side_left, \( \text{lda} \geq \max( 1, m ) \), if side = rocblas_side_right, \( \text{lda} \geq \max( 1, n ) \).
- \([\text{inout}]\, B : \text{void *}\) device array of device pointers storing each matrix B_i. each B_i is of dimension ( ldb, n ). Before entry, the leading m by n part of the array B_i must contain the right-hand side matrix B_i, and on exit is overwritten by the solution matrix X_i
- \([\text{in}]\, \text{ldb} : \text{rocblas_int}\) ldb specifies the first dimension of each B_i. \( \text{ldb} \geq \max( 1, m ) \).

• [in] invA: [void *] device array of device pointers storing the inverse diagonal blocks of each A_i. Each invA_i is of dimension (ld_invA, k), where k is m when rocblas_side_left and is n when rocblas_side_right. ld_invA must be equal to 128.

• [in] invA_size: [rocblas_int] invA_size specifies the number of elements of device memory in each invA_i.

• [in] compute_type: [rocblas_datatype] specifies the datatype of computation

2.12.1.9.2.79 rocblas_trsm_strided_batched_ex()

rocblas_status rocblas_trsm_strided_batched_ex (rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const void *alpha, const void *A, rocblas_int lda, rocblas_stride stride_A, void *B, rocblas_int ldb, rocblas_stride stride_B, rocblas_int batch_count, const void *invA, rocblas_int invA_size, rocblas_stride stride_invA, rocblas_datatype compute_type)

BLAS EX API

TRSM_STRIDED_BATCHED_EX solves

\[
op(A_i)X_i = \alpha B_i \text{ or } X_i\circ \op(A_i) = \alpha B_i,\]

for \( i = 1, \ldots, \text{batch_count}; \) and where \( \alpha \) is a scalar, \( X \) and \( B \) are strided batched \( m \) by \( n \) matrices, \( A \) is a strided batched triangular matrix and \( \op(A_i) \) is one of

\[
\op(A_i) = A_i \text{ or } \op(A_i) = A_i^\top \text{ or } \op(A_i) = A_i^\text{H}.
\]

Each matrix \( X_i \) is overwritten on \( B_i \).

TRSM_EX gives the user the ability to reuse each invA_i matrix between runs. If invA == NULL, rocblas_trsm_batched_ex will automatically calculate each invA_i on every run.

Setting up invA: Each accepted invA_i matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix A_i, followed by any smaller diagonal block that remains. To set up invA_i it is recommended that rocblas_trtri_batched be used with matrix A_i as the input. invA is a contiguous piece of memory holding each invA_i.

Device memory of size 128 x k should be allocated for each invA_i ahead of time, where k is m when rocblas_side_left and is n when rocblas_side_right. The actual number of elements in each invA_i should be passed as invA_size.

To begin, rocblas_trtri_batched must be called on the full 128x128 sized diagonal blocks of each matrix A_i. Below are the restricted parameters:

• \( n = 128 \)
• \( \text{ld}_\text{invA} = 128 \)
• \( \text{stride}_\text{invA} = 128x128 \)
• \( \text{batch_count} = k / 128, \)

Then any remaining block may be added:

• \( n = k \% 128 \)
- \( \text{invA} = \text{invA} + \text{stride_invA} \times \text{previous_batch_count} \)
- \( \text{ldinvA} = 128 \)
- \( \text{batch_count} = 1 \)

**Parameters**

- **[in]** `handle`: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** `side`: [rocblas_side] rocblas_side_left: \( \text{op}(A) \times X = \alpha B \). rocblas_side_right: \( X \times \text{op}(A) = \alpha B \).
- **[in]** `uplo`: [rocblas_fill] rocblas_fill_upper: each \( A_i \) is an upper triangular matrix.
  rocblas_fill_lower: each \( A_i \) is a lower triangular matrix.
- **[in]** `transA`: [rocblas_operation] transB: \( \text{op}(A) = A \). rocblas_operation_transpose: \( \text{op}(A) = A^T \).
  rocblas_operation_conjugate_transpose: \( \text{op}(A) = A^H \).
- **[in]** `diag`: [rocblas_diagonal] rocblas_diagonal_unit: each \( A_i \) is assumed to be unit triangular.
  rocblas_diagonal_non_unit: each \( A_i \) is not assumed to be unit triangular.
- **[in]** `m`: [rocblas_int] \( m \) specifies the number of rows of each \( B_i \). \( m \geq 0 \).
- **[in]** `n`: [rocblas_int] \( n \) specifies the number of columns of each \( B_i \). \( n \geq 0 \).
- **[in]** `alpha`: [void *] device pointer or host pointer specifying the scalar \( \alpha \). When \( \alpha \) is 
  &zero then \( A \) is not referenced, and \( B \) need not be set before entry.
- **[in]** `A`: [void *] device pointer storing matrix \( A \) of dimension ( \( lda, k \) ), where \( k \) is \( m \) when
  rocblas_side_left and is \( n \) when rocblas_side_right only the upper/lower triangular part is accessed.
- **[in]** `lda`: [rocblas_int] lda specifies the first dimension of \( A \). if \( side = \text{rocblas_side_left} \), lda >= 
  \( \max(1, m) \), if \( side = \text{rocblas_side_right} \), lda >= \( \max(1, n) \).
- **[in]** `stride_A`: [rocblas_stride] The stride between each \( A \) matrix.
- **[inout]** `B`: [void *] device pointer pointing to first matrix \( B_i \). each \( B_i \) is of dimension ( \( ldb, n \) ). Before entry, the leading \( m \) by \( n \) part of each array \( B_i \) must contain the right-hand side of matrix 
  \( B_i \) and on exit is overwritten by the solution matrix \( X_i \).
- **[in]** `ldb`: [rocblas_int] ldb specifies the first dimension of each \( B_i \). ldb >= \( \max(1, m) \).
- **[in]** `stride_B`: [rocblas_stride] The stride between each \( B_i \) matrix.
- **[in]** `batch_count`: [rocblas_int] specifies how many batches.
- **[in]** `invA`: [void *] device pointer storing the inverse diagonal blocks of each \( A_i \). invA points to 
  the first invA_1. each invA_i is of dimension ( \( ld_invA, k \) ), where \( k \) is \( m \) when rocblas_side_left and
  is \( n \) when rocblas_side_right. ld_invA must be equal to 128.
- **[in]** `invA_size`: [rocblas_int] invA_size specifies the number of elements of device memory in 
  each invA_i.
- **[in]** `stride_invA`: [rocblas_stride] The stride between each invA matrix.
- **[in]** `compute_type`: [rocblas_datatype] specifies the datatype of computation
2.12.1.9.2.80 Build Information

2.12.1.9.2.81 rocblas_get_version_string()

rocblas_status rocblas_get_version_string(char *buf, size_t len)
loads char* buf with the rocblas library version. size_t len is the maximum length of char* buf.

Parameters

- [inout] buf: pointer to buffer for version string
- [in] len: length of buf

2.12.1.9.2.82 Auxiliary

2.12.1.9.2.83 rocblas_pointer_to_mode()

rocblas_pointer_mode rocblas_pointer_to_mode(void *ptr)
Indicates whether the pointer is on the host or device.

2.12.1.9.2.84 rocblas_create_handle()

rocblas_status rocblas_create_handle(rocblas_handle *handle)
create handle

2.12.1.9.2.85 rocblas_destroy_handle()

rocblas_status rocblas_destroy_handle(rocblas_handle handle)
destroy handle

2.12.1.9.2.86 rocblas_add_stream()

rocblas_status rocblas_add_stream(rocblas_handle handle, hipStream_t stream)
add stream to handle

2.12.1.9.2.87 rocblas_set_stream()

rocblas_status rocblas_set_stream(rocblas_handle handle, hipStream_t stream)
remove any streams from handle, and add one
2.12.1.9.2.88 rocblas_get_stream()

rocblas_status rocblas_get_stream(rocblas_handle handle, hipStream_t *stream)
get stream [0] from handle

2.12.1.9.2.89 rocblas_set_pointer_mode()

rocblas_status rocblas_set_pointer_mode(rocblas_handle handle, rocblas_pointer_mode pointer_mode)
set rocblas_pointer_mode

2.12.1.9.2.90 rocblas_get_pointer_mode()

rocblas_status rocblas_get_pointer_mode(rocblas_handle handle, rocblas_pointer_mode *pointer_mode)
get rocblas_pointer_mode

2.12.1.9.2.91 rocblas_set_vector()

rocblas_status rocblas_set_vector(rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy)
copy vector from host to device

2.12.1.9.2.92 rocblas_set_vector_async()

rocblas_status rocblas_set_vector_async(rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy, hipStream_t stream)
asynchronously copy vector from host to device
rocblas_set_vector_async copies a vector from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters
• [in] n: [rocblas_int] number of elements in the vector
• [in] x: pointer to vector on the host
• [in] incx: [rocblas_int] specifies the increment for the elements of the vector
• [out] y: pointer to vector on the device
• [in] incy: [rocblas_int] specifies the increment for the elements of the vector
• [in] stream: specifies the stream into which this transfer request is queued
2.12.1.9.2.93 rocblas_get_vector()

rocblas_status rocblas_get_vector(rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy)

copy vector from device to host

2.12.1.9.2.94 rocblas_get_vector_async()

rocblas_status rocblas_get_vector_async(rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy, hipStream_t stream)
asynchronously copy vector from device to host

rocblas_get_vector_async copies a vector from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters

• [in] n: [rocblas_int] number of elements in the vector
• [in] x: pointer to vector on the device
• [in] incx: [rocblas_int] specifies the increment for the elements of the vector
• [out] y: pointer to vector on the host
• [in] incy: [rocblas_int] specifies the increment for the elements of the vector
• [in] stream: specifies the stream into which this transfer request is queued

2.12.1.9.2.95 rocblas_set_matrix()

rocblas_status rocblas_set_matrix(rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb)
copy matrix from host to device

2.12.1.9.2.96 rocblas_get_matrix()

rocblas_status rocblas_get_matrix(rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb)
copy matrix from device to host

2.12.1.9.2.97 rocblas_get_matrix_async()

rocblas_status rocblas_get_matrix_async(rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb, hipStream_t stream)
asynchronously copy matrix from device to host

rocblas_get_matrix_async copies a matrix from device memory to pinned host memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters

• [in] rows: [rocblas_int] number of rows in matrices
• [in] cols: [rocblas_int] number of columns in matrices
• [in] `elem_size`: [rocblas_int] number of bytes per element in the matrix
• [in] `a`: pointer to matrix on the GPU
• [in] `lda`: [rocblas_int] specifies the leading dimension of A
• [out] `b`: pointer to matrix on the host
• [in] `ldb`: [rocblas_int] specifies the leading dimension of B
• [in] `stream`: specifies the stream into which this transfer request is queued

2.12.1.9.2.98 rocblas_start_device_memory_size_query()

rocblas_status rocblas_start_device_memory_size_query (rocblas_handle handle)

Indicates that subsequent rocBLAS kernel calls should collect the optimal device memory size in bytes for their
given kernel arguments, and keep track of the maximum. Each kernel call can reuse temporary device memory
on the same stream, so the maximum is collected. Returns rocblas_status_size_query_mismatch if another size
query is already in progress; returns rocblas_status_success otherwise.

Parameters

• [in] `handle`: rocblas handle

2.12.1.9.2.99 rocblas_stop_device_memory_size_query()

rocblas_status rocblas_stop_device_memory_size_query (rocblas_handle handle, size_t *size)

Stops collecting optimal device memory size information Returns rocblas_status_size_query_mismatch if a col-
lection is not underway; rocblas_status_invalid_handle if handle is nullptr; rocblas_status_invalid_pointer if size
is nullptr; rocblas_status_success otherwise

Parameters

• [in] `handle`: rocblas handle
• [out] `size`: maximum of the optimal sizes collected

2.12.1.9.2.100 rocblas_get_device_memory_size()

rocblas_status rocblas_get_device_memory_size (rocblas_handle handle, size_t *size)

Gets the current device memory size for the handle Returns rocblas_status_invalid_handle if handle is nullptr;
rocblas_status_invalid_pointer if size is nullptr; rocblas_status_success otherwise

Parameters

• [in] `handle`: rocblas handle
• [out] `size`: current device memory size for the handle
### 2.12.1.9.2.101 rocblas_set_device_memory_size()

**rocblas_status rocblas_set_device_memory_size (rocblas_handle handle, size_t size)**

Changes the size of allocated device memory at runtime.

Any previously allocated device memory is freed.

If size > 0 sets the device memory size to the specified size (in bytes) If size == 0 frees the memory allocated so far, and lets rocBLAS manage device memory in the future, expanding it when necessary

Returns rocblas_status_invalid_handle if handle is nullptr; rocblas_status_invalid_pointer if size is nullptr; rocblas_status_success otherwise

**Parameters**

- [in] handle: rocblas handle
- [in] size: size of allocated device memory

### 2.12.1.9.2.102 rocblas_is_managing_device_memory()

**bool rocblas_is_managing_device_memory (rocblas_handle handle)**

Returns true when device memory in handle is managed by rocBLAS

**Parameters**

- [in] handle: rocblas handle

### 2.12.1.10 All API

**struct rocblas_half**

```
#include <rocblas-types.h>
```

Represents a 16 bit floating point number.

**Public Members**

- uint16_t data

**namespace rocblas**

**Functions**

- void reinit_logs()

**file rocblas-auxiliary.h**

```
#include "rocblas-export.h"
#include "rocblas-types.h"
```

rocblas-auxiliary.h provides auxiliary functions in rocblas
Functions

rocblas_status rocblas_create_handle (rocblas_handle *handle)
create handle

rocblas_status rocblas_destroy_handle (rocblas_handle handle)
destroy handle

rocblas_status rocblas_add_stream (rocblas_handle handle, hipStream_t stream)
add stream to handle

rocblas_status rocblas_set_stream (rocblas_handle handle, hipStream_t stream)
remove any streams from handle, and add one

rocblas_status rocblas_get_stream (rocblas_handle handle, hipStream_t *stream)
get stream [0] from handle

rocblas_status rocblas_set_pointer_mode (rocblas_handle handle, rocblas_pointer_mode pointer_mode)
set rocblas_pointer_mode

rocblas_status rocblas_get_pointer_mode (rocblas_handle handle, rocblas_pointer_mode *pointer_mode)
get rocblas_pointer_mode

rocblas_pointer_mode rocblas_pointer_to_mode (void *ptr)
Indicates whether the pointer is on the host or device.

rocblas_status rocblas_set_vector (rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy)
copy vector from host to device

rocblas_status rocblas_get_vector (rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy)
copy vector from device to host

rocblas_status rocblas_set_matrix (rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb)
copy matrix from host to device

rocblas_status rocblas_get_matrix (rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb)
copy matrix from device to host

rocblas_status rocblas_set_vector_async (rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy, hipStream_t stream)
asynchronously copy vector from host to device

rocblas_set_vector_async copies a vector from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters

- [in] n: [rocblas_int] number of elements in the vector
- [in] x: pointer to vector on the host
- [in] incx: [rocblas_int] specifies the increment for the elements of the vector
- [out] y: pointer to vector on the device
- [in] incy: [rocblas_int] specifies the increment for the elements of the vector
- [in] stream: specifies the stream into which this transfer request is queued
rocblas_status rocblas_get_vector_async(rocblas_int n, rocblas_int elem_size, const void *x, rocblas_int incx, void *y, rocblas_int incy, hipStream_t stream)

asynchronously copy vector from device to host
rocblas_get_vector_async copies a vector from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters
- [in] n: [rocblas_int] number of elements in the vector
- [in] x: pointer to vector on the device
- [in] incx: [rocblas_int] specifies the increment for the elements of the vector
- [out] y: pointer to vector on the host
- [in] incy: [rocblas_int] specifies the increment for the elements of the vector
- [in] stream: specifies the stream into which this transfer request is queued

rocblas_status rocblas_set_matrix_async(rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb, hipStream_t stream)

asynchronously copy matrix from host to device
rocblas_set_matrix_async copies a matrix from pinned host memory to device memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters
- [in] rows: [rocblas_int] number of rows in matrices
- [in] cols: [rocblas_int] number of columns in matrices
- [in] elem_size: [rocblas_int] number of bytes per element in the matrix
- [in] a: pointer to matrix on the host
- [in] lda: [rocblas_int] specifies the leading dimension of A
- [out] b: pointer to matrix on the GPU
- [in] ldb: [rocblas_int] specifies the leading dimension of B
- [in] stream: specifies the stream into which this transfer request is queued

rocblas_status rocblas_get_matrix_async(rocblas_int rows, rocblas_int cols, rocblas_int elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb, hipStream_t stream)

asynchronously copy matrix from device to host
rocblas_get_matrix_async copies a matrix from device memory to pinned host memory asynchronously. Memory on the host must be allocated with hipHostMalloc or the transfer will be synchronous.

Parameters
- [in] rows: [rocblas_int] number of rows in matrices
- [in] cols: [rocblas_int] number of columns in matrices
- [in] elem_size: [rocblas_int] number of bytes per element in the matrix
- [in] a: pointer to matrix on the GPU
- [in] lda: [rocblas_int] specifies the leading dimension of A
• [out] b: pointer to matrix on the host
• [in] ldb: [rocblas_int] specifies the leading dimension of B
• [in] stream: specifies the stream into which this transfer request is queued

file rocblas-functions.h
#include “rocblas-export.h”
#include “rocblas-types.h”
rocblas_functions.h provides Basic Linear Algebra Subprograms of Level 1, 2 and 3, using HIP optimized for AMD HCC-based GPU hardware. This library can also run on CUDA-based NVIDIA GPUs. This file exposes C89 BLAS interface

Defines

ROCBLAS_VA_OPT_3RD_ARG (_1, _2, _3, ...)
ROCBLAS_VA_OPT_SUPPORTED (...)
ROCBLAS_VA_OPT_COUNT_IMPL (X, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...)
ROCBLAS_VA_OPT_COUNT (...)
ROCBLAS_VA_OPT_PRAGMA_SELECT0 (...)
ROCBLAS_VA_OPT_PRAGMA_SELECTN (pragma, ...) 
ROCBLAS_VA_OPT_PRAGMA_IMPL2 (pragma, count)
ROCBLAS_VA_OPT_PRAGMA_IMPL (pragma, count)
ROCBLAS_VA_OPT_PRAGMA (pragma, ...)
rocblas_gemm_ex (handle, transA, transB, m, n, k, alpha, a, a_type, lda, b, b_type, ldb, beta, c, c_type, ldc, d, d_type, ldd, compute_type, algo, solution_index, flags, ...)
rocblas_gemm_strided_batched_ex (handle, transA, transB, m, n, k, alpha, a, a_type, lda, stride_a, b, b_type, ldb, stride_b, beta, c, c_type, ldc, stride_c, d, d_type, ldd, stride_d, batch_count, compute_type, algo, solution_index, flags, ...)
rocblas_trsm_ex (handle, side, uplo, transA, diag, m, n, alpha, A, lda, B, ldb, invA, invA_size, compute_type, ...) 

Functions

rocblas_status rocblas_sscal (rocblas_handle handle, rocblas_int n, const float *alpha, float *x, rocblas_int incx)

BLAS Level 1 API.
scal scales each element of vector x with scalar alpha.

\[ x := \alpha \times x \]

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] the number of elements in x.
• [in] alpha: device pointer or host pointer for the scalar alpha.
• [inout] x: device pointer storing vector x.
• [in] incx: [rocblas_int] specifies the increment for the elements of x.
rocblas_status rocblas_dscal(rocblas_handle handle, rocblas_int n, const double *alpha, double *x, rocblas_int incx)

rocblas_status rocblas_cscal(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, rocblas_float_complex *x, rocblas_int incx)

rocblas_status rocblas_zscal(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, rocblas_double_complex *x, rocblas_int incx)

rocblas_status rocblas_csscal(rocblas_handle handle, rocblas_int n, const float *alpha, rocblas_float_complex *x, rocblas_int incx)

rocblas_status rocblas_zdscal(rocblas_handle handle, rocblas_int n, const double *alpha, rocblas_double_complex *x, rocblas_int incx)

rocblas_status rocblas_sscal_batched(rocblas_handle handle, rocblas_int n, const float *alpha, float *const x[], rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_dscal_batched(rocblas_handle handle, rocblas_int n, const double *alpha, double *const x[], rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_cscal_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_zscal_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_csscal_batched(rocblas_handle handle, rocblas_int n, const float *alpha, rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_zdscal_batched(rocblas_handle handle, rocblas_int n, const double *alpha, rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count)

BLAS Level 1 API.

scal_batched scales each element of vector x_i with scalar alpha, for i = 1, . . . , batch_count.

\[ x_i := \alpha \times x_i \]

where (x_i) is the i-th instance of the batch.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each x_i.
- [in] alpha: host pointer or device pointer for the scalar alpha.
- [inout] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
- [in] batch_count: [rocblas_int] specifies the number of batches in x.
rocblas_status rocblas_sscal_strided_batched(rocblas_handle handle, rocblas_int n, const float *alpha, float *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

BLAS Level 1 API.

scal_strided_batched scales each element of vector x_i with scalar alpha, for i = 1, ..., batch_count.

\[ x_i := \alpha \cdot x_i, \]

where \((x_i)\) is the i-th instance of the batch.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each \(x_i\).
- [in] alpha: host pointer or device pointer for the scalar alpha.
- [inout] x: device pointer to the first vector \((x_1)\) in the batch.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
- [in] stride_x: [rocblas_stride] stride from the start of one vector \((x_i)\) and the next one \((x_{i+1})\). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size, for a typical case this means stride_x >= n * incx.
- [in] batch_count: [rocblas_int] specifies the number of batches in x.

rocblas_status rocblas_dscal_strided_batched(rocblas_handle handle, rocblas_int n, const double *alpha, double *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_cscal_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_zscal_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_csscal_strided_batched(rocblas_handle handle, rocblas_int n, const float *alpha, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_zdscal_strided_batched(rocblas_handle handle, rocblas_int n, const double *alpha, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_scopy(rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, float *y, rocblas_int incy)

BLAS Level 1 API.

copy copies each element \(x[i]\) into \(y[i]\), for \(i = 1, \ldots, n\)
y := x,

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x to be copied to y.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
- [out] y: device pointer storing vector y.
- [in] incy: [rocblas_int] specifies the increment for the elements of y.

rocblas_status rocblas_dcopy (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, double *y, rocblas_int incy)

rocblas_status rocblas_ccopy (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy)

rocblas_status rocblas_zcopy (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy)

rocblas_status rocblas_scopy_batched (rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, float *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_dcopy_batched (rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, double *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_ccopy_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)

BLAS Level 1 API.

copy_batched copies each element x_i[j] into y_i[j], for j = 1, \ldots, n; i = 1, \ldots, \text{batch_count}

\[ y_{i,j} := x_{i,j}, \]

where \((x_i, y_i)\) is the i-th instance of the batch. \(x_i\) and \(y_i\) are vectors.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each \(x_i\) to be copied to \(y_i\).
- [in] x: device array of device pointers storing each vector \(x_i\).
- [in] incx: [rocblas_int] specifies the increment for the elements of each vector \(x_i\).
- [out] y: device array of device pointers storing each vector \(y_i\).
- [in] incy: [rocblas_int] specifies the increment for the elements of each vector \(y_i\).
- [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dcopy_batched (rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, double *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_ccopy_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_zcopy_batched
(rocblas_handle handle, rocblas_int n, const
rocblas_double_complex *const x[], rocblas_int incx, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_scopy_strided_batched
(rocblas_handle handle, rocblas_int n, const
float *x, rocblas_int incx, rocblas_stride stridex, float *y, rocblas_int incy, rocblas_stride stridey, rocblas_int
batch_count)

BLAS Level 1 API.

copy_strided_batched copies each element \( x_i[j] \) into \( y_i[j] \), for \( j = 1, \ldots, n; i = 1, \ldots, \text{batch_count} \)

\[
y_i := x_i,
\]

where \((x_i, y_i)\) is the \(i\)-th instance of the batch. \(x_i\) and \(y_i\) are vectors.

Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** n: [rocblas_int] the number of elements in each \(x_i\) to be copied to \(y_i\).
- **[in]** x: device pointer to the first vector \((x_1)\) in the batch.
- **[in]** incx: [rocblas_int] specifies the increments for the elements of vectors \(x_i\).
- **[in]** stridex: [rocblas_stride] stride from the start of one vector \((x_i)\) and the next one \((x_{i+1})\). There are no restrictions placed on \(\text{stride}_x\), however the user should take care to ensure that \(\text{stride}_x\) is of appropriate size, for a typical case this means \(\text{stride}_x \geq n \times \text{incx}\).
- **[out]** y: device pointer to the first vector \((y_1)\) in the batch.
- **[in]** incy: [rocblas_int] specifies the increment for the elements of vectors \(y_i\).
- **[in]** stridey: [rocblas_stride] stride from the start of one vector \((y_i)\) and the next one \((y_{i+1})\). There are no restrictions placed on \(\text{stride}_y\), however the user should take care to ensure that \(\text{stride}_y\) is of appropriate size, for a typical case this means \(\text{stride}_y \geq n \times \text{incy}\). \(\text{stridey}\) should be non zero.
- **[in]** incy: [rocblas_int] specifies the increment for the elements of \(y\).
- **[in]** batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dcopy_strided_batched
(rocblas_handle handle, rocblas_int n, const
double *x, rocblas_int incx, rocblas_stride stridex, double *y, rocblas_int incy, rocblas_stride stridey, rocblas_int
batch_count)

rocblas_status rocblas_ccopy_strided_batched
(rocblas_handle handle, rocblas_int n, const
rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int
batch_count)

Chapter 2. Solid Compilation Foundation and Language Support
rocblas_status rocblas_zcopy_strided_batched(rocblas_handle handle, rocblas_int n, 
const rocblas_double_complex *x, 
rocblas_int incx, rocblas_stride stridex, 
rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_sdot (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, 
const float *y, rocblas_int incy, float *result)

BLAS Level 1 API.

dot(u) performs the dot product of vectors x and y

\[
\text{result} = x \times y
\]

dotc performs the dot product of the conjugate of complex vector x and complex vector y

\[
\text{result} = \text{conjugate}(x) \times y
\]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x and y.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of y.
- [in] y: device pointer storing vector y.
- [in] incy: [rocblas_int] specifies the increment for the elements of y.
- [inout] result: device pointer or host pointer to store the dot product. return is 0.0 if n <= 0.

rocblas_status rocblas_ddot (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, 
const double *y, rocblas_int incy, double *result)

rocblas_status rocblas_hdot (rocblas_handle handle, rocblas_int n, const rocblas_half *x, 
rocblas_int incx, const rocblas_half *y, rocblas_int incy, 
rocblas_half *result)

rocblas_status rocblas_bfdot (rocblas_handle handle, rocblas_int n, const rocblas_bfloat16 *x, 
rocblas_int incx, const rocblas_bfloat16 *y, rocblas_int incy, 
rocblas_bfloat16 *result)

rocblas_status rocblas_cdotu (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *, 
rocblas_int incx, const rocblas_float_complex *y, rocblas_int incy, 
rocblas_float_complex *result)

rocblas_status rocblas_zdotu (rocblas_handle handle, rocblas_int n, const 
rocblas_double_complex *x, rocblas_int incx, const 
rocblas_double_complex *y, rocblas_int incy, 
rocblas_double_complex *result)

rocblas_status rocblas_cdotc (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *, 
rocblas_int incx, const rocblas_float_complex *y, rocblas_int incy, 
rocblas_float_complex *result)
rocblas_status rocblas_zdotc (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, const rocblas_double_complex *y, rocblas_int incy, rocblas_double_complex *result)

rocblas_status rocblas_zdotc_batched (rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, const float *const y[], rocblas_int incy, rocblas_int batch_count, float *result)

BLAS Level 1 API.

dot_batched(u) performs a batch of dot products of vectors x and y

\[
\text{result}_i = x_i \times y_i;
\]

dotc_batched performs a batch of dot products of the conjugate of complex vector x and complex vector y

\[
\text{result}_i = \text{conjugate} (x_i) \times y_i;
\]

where \((x_i, y_i)\) is the \(i\)-th instance of the batch. \(x_i\) and \(y_i\) are vectors, for \(i = 1, \ldots, \text{batch\_count}\)

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in each \(x_i\) and \(y_i\).
- [in] x: device array of device pointers storing each vector \(x_i\).
- [in] incx: [rocblas_int] specifies the increment for the elements of each \(x_i\).
- [in] y: device array of device pointers storing each vector \(y_i\).
- [in] incy: [rocblas_int] specifies the increment for the elements of each \(y_i\).
- [in] batch_count: [rocblas_int] number of instances in the batch
- [inout] result: device array or host array of batch_count size to store the dot products of each batch. return 0.0 for each element if \(n \leq 0\).
**rocblas_status rocblas_zdotu_batched** (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count, rocblas_double_complex *result)

**rocblas_status rocblas_cdotc_batched** (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, const rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count, rocblas_float_complex *result)

**rocblas_status rocblas_zdotc_batched** (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count, rocblas_double_complex *result)

**rocblas_status rocblas_sdot_strided_batched** (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, const float *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, float *result)

BLAS Level 1 API.

`dot_strided_batched(u)` performs a batch of dot products of vectors `x` and `y`

\[
result_i = x_i \times y_i;
\]

`dotc_strided_batched` performs a batch of dot products of the conjugate of complex vector `x` and complex vector `y`

\[
result_i = \text{conjugate} \ (x_i) \times y_i;
\]

where \((x_i, y_i)\) is the \(i\)-th instance of the batch. `x_i` and `y_i` are vectors, for \(i = 1, \ldots, \text{batch\_count}\)

**Parameters**

- **[in] handle:** [rocblas_handle] handle to the rocblas library context queue.
- **[in] n:** [rocblas_int] the number of elements in each `x_i` and `y_i`.
- **[in] x:** device pointer to the first vector (`x_1`) in the batch.
- **[in] incx:** [rocblas_int] specifies the increment for the elements of each `x_i`.
- **[in] stridex:** [rocblas_stride] stride from the start of one vector (`x_i`) and the next one (`x_i+1`)
- **[in] y:** device pointer to the first vector (`y_1`) in the batch.
- **[in] incy:** [rocblas_int] specifies the increment for the elements of each `y_i`.
- **[in] stridey:** [rocblas_stride] stride from the start of one vector (`y_i`) and the next one (`y_i+1`)
- **[in] batch_count:** [rocblas_int] number of instances in the batch
- **[inout] result:** device array or host array of `batch\_count` size to store the dot products of each batch. return 0.0 for each element if `n <= 0.`
rocblas_status rocblas_ddot_strided_batched(rocblas_handle handle, rocblas_int n, const
double *x, rocblas_int incx, rocblas_stride stridx, const double *y, rocblas_int
incy, rocblas_stride stridey, rocblas_int batch_count, double *result)

rocblas_status rocblas_ddot_strided_batched(rocblas_handle handle, rocblas_int n,
const rocblas_half *x, rocblas_int incx, rocblas_stride stridx, const rocblas_half
*y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_half *result)

rocblas_status rocblas_bfdot_strided_batched(rocblas_handle handle, rocblas_int n,
const rocblas_bfloat16 *x, rocblas_int incx, rocblas_stride stridx, const rocblas_bfloat16
*y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_bfloat16 *result)

rocblas_status rocblas_cdotu_strided_batched(rocblas_handle handle, rocblas_int n, const
rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridx, const rocblas_float_complex
*y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_float_complex *result)

rocblas_status rocblas_cdotc_strided_batched(rocblas_handle handle, rocblas_int n, const
rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridx, const rocblas_float_complex
*y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_float_complex *result)

rocblas_status rocblas_zdotu_strided_batched(rocblas_handle handle, rocblas_int n, const
rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridx, const rocblas_double_complex
*y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_double_complex
*result)

rocblas_status rocblas_zdotc_strided_batched(rocblas_handle handle, rocblas_int n, const
rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridx, const rocblas_double_complex
*y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count, rocblas_double_complex
*result)

rocblas_status rocblas_sswap(rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, float
*y, rocblas_int incy)

BLAS Level 1 API.
swap interchanges vectors x and y.

\[
y := x; \quad x := y
\]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] \( n \): [rocblas_int] the number of elements in \( x \) and \( y \).
• [inout] \( x \): device pointer storing vector \( x \).
• [in] \( \text{incx} \): [rocblas_int] specifies the increment for the elements of \( x \).
• [inout] \( y \): device pointer storing vector \( y \).
• [in] \( \text{incy} \): [rocblas_int] specifies the increment for the elements of \( y \).

rocblas_status rocblas_dswap(rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, double *y, rocblas_int incy)
rocblas_status rocblas_cswap(rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy)
rocblas_status rocblas_zswap(rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy)
rocblas_status rocblas_sswap_batched(rocblas_handle handle, rocblas_int n, float *x[], rocblas_int incx, float *y[], rocblas_int incy, rocblas_int batch_count)

BLAS Level 1 API.

swap_batched interchanges vectors \( x_i \) and \( y_i \), for \( i = 1, \ldots, \text{batch\_count} \)

\[
y_i := x_i; x_i := y_i
\]

Parameters
• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] \( n \): [rocblas_int] the number of elements in each \( x_i \) and \( y_i \).
• [inout] \( x \): device array of device pointers storing each vector \( x_i \).
• [in] \( \text{incx} \): [rocblas_int] specifies the increment for the elements of each \( x_i \).
• [inout] \( y \): device array of device pointers storing each vector \( y_i \).
• [in] \( \text{incy} \): [rocblas_int] specifies the increment for the elements of each \( y_i \).
• [in] \( \text{batch\_count} \): [rocblas_int] number of instances in the batch.

rocblas_status rocblas_dswap_batched(rocblas_handle handle, rocblas_int n, double *x[], rocblas_int incx, double *y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_cswap_batched(rocblas_handle handle, rocblas_int n, rocblas_float_complex *x[], rocblas_int incx, rocblas_float_complex *y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_zswap_batched(rocblas_handle handle, rocblas_int n, rocblas_double_complex *x[], rocblas_int incx, rocblas_double_complex *y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_sswap_strided_batched(rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, rocblas_int incy, rocblas_int batch_count, float *x[], rocblas_int incx, rocblas_int incy, rocblas_int batch_count)

BLAS Level 1 API.

swap_strided_batched interchanges vectors \( x_i \) and \( y_i \), for \( i = 1, \ldots, \text{batch\_count} \)
\[ y_i := x_i; \quad x_i := y_i \]

**Parameters**

- **[in]** `handle`: `[rocblas_handle]` handle to the rocblas library context queue.
- **[in]** `n`: `[rocblas_int]` the number of elements in each `x_i` and `y_i`.
- **[inout]** `x`: device pointer to the first vector `x_1`.
- **[in]** `incx`: `[rocblas_int]` specifies the increment for the elements of `x`.
- **[in]** `stridex`: `[rocblas_stride]` stride from the start of one vector (`x_i`) and the next one (`x_i+1`). There are no restrictions placed on `stridex`, however the user should take care to ensure that `stridex` is of appropriate size, for a typical case this means `stridex >= n * incx`.
- **[inout]** `y`: device pointer to the first vector `y_1`.
- **[in]** `incy`: `[rocblas_int]` specifies the increment for the elements of `y`.
- **[in]** `stridey`: `[rocblas_stride]` stride from the start of one vector (`y_i`) and the next one (`y_i+1`). There are no restrictions placed on `stridey`, however the user should take care to ensure that `stridey` is of appropriate size, for a typical case this means `stridey >= n * incy`. `stridey` should be non zero.
- **[in]** `batch_count`: `[rocblas_int]` number of instances in the batch.

```cpp
rocblas_status rocblas_dswap_strided_batched(rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, rocblas_stride stridex, double *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```

```cpp
rocblas_status rocblas_cswap_strided_batched(rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```

```cpp
rocblas_status rocblas_zswap_strided_batched(rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```

```cpp
rocblas_status rocblas_saxpy(rocblas_handle handle, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, float *y, rocblas_int incy)
```

BLAS Level 1 API.

`axpy` computes constant `alpha` multiplied by vector `x`, plus vector `y`.

\[ y := \alpha * x + y \]

**Parameters**

- **[in]** `handle`: `[rocblas_handle]` handle to the rocblas library context queue.
- **[in]** `n`: `[rocblas_int]` the number of elements in `x` and `y`.
- **[in]** `alpha`: device pointer or host pointer to specify the scalar `alpha`.
- **[in]** `x`: device pointer storing vector `x`. 
rocblas_status rocblas_daxpy (rocblas_handle handle, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, double *y, rocblas_int incy)
rocblas_status rocblas_haxpy (rocblas_handle handle, rocblas_int n, const rocblas_half *alpha, const rocblas_half *x, rocblas_int incx, rocblas_half *y, rocblas_int incy)
rocblas_status rocblas_caxpy (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy)
rocblas_status rocblas_zaxpy (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy)
rocblas_status rocblas_haxpy_batched (rocblas_handle handle, rocblas_int n, const rocblas_half *alpha, const rocblas_half *const x[], rocblas_int incx, rocblas_half *const y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_saxpy_batched (rocblas_handle handle, rocblas_int n, const float *alpha, const float *const x[], rocblas_int incx, float *const y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_daxpy_batched (rocblas_handle handle, rocblas_int n, const double *alpha, const double *const x[], rocblas_int incx, double *const y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_caxpy_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)

BLAS Level 1 API.

axpy_batched compute $y := \alpha \times x + y$ over a set of batched vectors.

Parameters

- [in] handle: rocblas_handle handle to the rocblas library context queue.
- [in] n: rocblas_int
- [in] alpha: specifies the scalar $\alpha$.
- [in] x: pointer storing vector $x$ on the GPU.
- [in] incx: rocblas_int specifies the increment for the elements of $x$.
- [out] y: pointer storing vector $y$ on the GPU.
- [inout] incy: rocblas_int specifies the increment for the elements of $y$.
- [in] batch_count: rocblas_int number of instances in the batch
**rocblas_status rocblas_zaxpy_batched**

```c
rocblas_status rocblas_zaxpy_batched (rocblas_handle handle, rocblas_int n, const
rocblas_double_complex *alpha, const
rocblas_double_complex *const x[], rocblas_int incx, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)
```

**rocblas_status rocblas_haxpy_strided_batched**

```c
rocblas_status rocblas_haxpy_strided_batched (rocblas_handle handle, rocblas_int n, const
rocblas_half *alpha, const rocblas_half *
rocblas_int incx, rocblas_stride stridex, rocblas_half *
rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```

BLAS Level 1 API.

axpy_batched compute \( y := \alpha \cdot \mathbf{x} + y \) over a set of strided batched vectors.

**Parameters**

- `[in]` `handle`: rocblas_handle handle to the rocblas library context queue.
- `[in]` `n`: rocblas_int
- `[in]` `alpha`: specifies the scalar \( \alpha \).
- `[in]` `x`: pointer storing vector \( \mathbf{x} \) on the GPU.
- `[in]` `incx`: rocblas_int specifies the increment for the elements of \( \mathbf{x} \).
- `[in]` `stridex`: rocblas_stride specifies the increment between vectors of \( \mathbf{x} \).
- `[out]` `y`: pointer storing vector \( \mathbf{y} \) on the GPU.
- `[inout]` `incy`: rocblas_int specifies the increment for the elements of \( \mathbf{y} \).
- `[in]` `stridey`: rocblas_stride specifies the increment between vectors of \( \mathbf{y} \).
- `[in]` `batch_count`: rocblas_int number of instances in the batch.

**rocblas_status rocblas_saxpy_strided_batched**

```c
rocblas_status rocblas_saxpy_strided_batched (rocblas_handle handle, rocblas_int n, const
float *alpha, const float *
rocblas_int incx, rocblas_stride stridex, float *
rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```

**rocblas_status rocblas_daxpy_strided_batched**

```c
rocblas_status rocblas_daxpy_strided_batched (rocblas_handle handle, rocblas_int n, const
double *alpha, const double *
x, rocblas_int incx, rocblas_stride stridex, double *
y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```

**rocblas_status rocblas_caxpy_strided_batched**

```c
rocblas_status rocblas_caxpy_strided_batched (rocblas_handle handle, rocblas_int n, const
rocblas_float_complex *alpha, const
rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_float_complex *
y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)
```
rocblas_status rocblas_zaxpy_strided_batched (rocblas_handle handle, rocblas_int n,
const rocblas_double_complex *alpha, const rocblas_double_complex *x,
rocblas_int incx, rocblas_stride stridx,
rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_sasum (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, float *result)

BLAS Level 1 API.

asum computes the sum of the magnitudes of elements of a real vector x, or the sum of magnitudes of the real and imaginary parts of elements if x is a complex vector.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x and y.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x. incx must be > 0.
- [inout] result: device pointer or host pointer to store the asum product. return is 0.0 if n <= 0.

rocblas_status rocblas_dasum (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, double *result)

rocblas_status rocblas_scasum (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, float *result)

rocblas_status rocblas_dzasum (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, double *result)

rocblas_status rocblas_sasum_batched (rocblas_handle handle, rocblas_int n, const float *x[], rocblas_int incx, float *results)

BLAS Level 1 API.

asum_batched computes the sum of the magnitudes of the elements in a batch of real vectors x_i, or the sum of magnitudes of the real and imaginary parts of elements if x_i is a complex vector, for i = 1, ..., batch_count.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each vector x_i.
- [in] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- [out] results: device array or host array of batch_count size for results. return is 0.0 if n, incx<=0.
- [in] batch_count: [rocblas_int] number of instances in the batch.
BLAS Level 1 API.

asum_strided_batched computes the sum of the magnitudes of elements of a real vectors $x_i$, or the sum of magnitudes of the real and imaginary parts of elements if $x_i$ is a complex vector, for $i = 1, \ldots, \text{batch\_count}$

Parameters

- `[in] handle`: [rocblas_handle] handle to the rocblas library context queue.
- `[in] n`: [rocblas_int] number of elements in each vector $x_i$
- `[in] x`: device pointer to the first vector $x_1$.
- `[in] incx`: [rocblas_int] specifies the increment for the elements of each $x_i$. incx must be $> 0$.
- `[in] stridex`: [rocblas_stride] stride from the start of one vector ($x_i$) and the next one ($x_{i+1}$). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size, for a typical case this means stride_x $\geq n \times \text{incx}$.
- `[out] results`: device pointer or host pointer to array for storing contiguous batch_count results. return is 0.0 if n, incx<=$0$.
- `[in] batch\_count`: [rocblas_int] number of instances in the batch

BLAS Level 1 API.

nrm2 computes the euclidean norm of a real or complex vector
result := sqrt( x'*x ) for real vectors
result := sqrt( x**H*x ) for complex vectors

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of y.
- [inout] result: device pointer or host pointer to store the nrm2 product. return is 0.0 if n, incx<=0.

rocblas_status rocblas_dnrm2 (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, double *result)
rocblas_status rocblas_scnrm2 (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, float *result)
rocblas_status rocblas_dznrm2 (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, double *result)
rocblas_status rocblas_snrm2_batched (rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, rocblas_int batch_count, float *results)

BLAS Level 1 API.

nrm2_batched computes the euclidean norm over a batch of real or complex vectors

result := sqrt( x_i'*x_i ) for real vectors x, for i = 1, ..., batch_count
result := sqrt( x_i**H*x_i ) for complex vectors x, for i = 1, ..., batch_count

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each x_i.
- [in] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- [in] batch_count: [rocblas_int] number of instances in the batch
- [out] results: device pointer or host pointer to array of batch_count size for nrm2 results. return is 0.0 for each element if n <= 0, incx<=0.

rocblas_status rocblas_dnrm2_batched (rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, rocblas_int batch_count, double *results)
rocblas_status rocblas_scnrm2_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count, float *results)
rocblas_status rocblas_dznrm2_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count, double *results)

rocblas_status rocblas_snrm2_strided_batched (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)

rocblas_status rocblas_dnrm2_strided_batched (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

rocblas_status rocblas_scnrm2_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)

rocblas_status rocblas_dznrm2_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

rocblas_status rocblas_isamax (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_int *result)

BLAS Level 1 API.

nrm2_strided_batched computes the euclidean norm over a batch of real or complex vectors

\[ \text{nrm2}_\text{strided}_\text{batched} \text{ computes } \sqrt{x_i'x_i} \text{ for real vectors } x, \text{ for } i = 1, \ldots, \text{batch_count} \]

\[ \text{nrm2}_\text{strided}_\text{batched} \text{ computes } \sqrt{x_i^*x_i} \text{ for complex vectors, for } i = 1, \ldots, \text{batch_count} \]

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each x_i.
- [in] x: device pointer to the first vector x_1.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- [in] stridex: [rocblas_stride] stride from the start of one vector (x_i) and the next one (x_i+1). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size, for a typical case this means stride_x \( \geq n \times \text{incx} \).
- [in] batch_count: [rocblas_int] number of instances in the batch
- [out] results: device pointer or host pointer to array for storing contiguous batch_count results. return is 0.0 for each element if n <= 0, incx<=0.

rocblas_status rocblas_dnrm2_strided_batched (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

rocblas_status rocblas_scnrm2_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, float *results)

rocblas_status rocblas_dznrm2_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, double *results)

rocblas_status rocblas_isamax (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_int *result)

BLAS Level 1 API.

amax finds the first index of the element of maximum magnitude of a vector x. vector

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] the number of elements in x.
- [in] x: device pointer storing vector x.
• [in] iny: [rocblas_int] specifies the increment for the elements of y.
• [inout] result: device pointer or host pointer to store the amax index. return is 0.0 if n, inx<=0.

rocblas_status rocblas_idamax (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int inx, rocblas_int *result)
rocblas_status rocblas_icamax (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int inx, rocblas_int *result)
rocblas_status rocblas_izamax (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int inx, rocblas_int *result)
rocblas_status rocblas_isamax_batched (rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int inx, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.

amax_batched finds the first index of the element of maximum magnitude of each vector x_i in a batch, for i = 1, ..., batch_count.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in each vector x_i
• [in] x: device array of device pointers storing each vector x_i.
• [in] inx: [rocblas_int] specifies the increment for the elements of each x_i. inx must be > 0.
• [in] batch_count: [rocblas_int] number of instances in the batch, must be > 0.
• [out] result: device or host array of pointers of batch_count size for results. return is 0 if n, inx<=0.

rocblas_status rocblas_idamax_batched (rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int inx, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_icamax_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int inx, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_izamax_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int inx, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_isamax_strided_batched (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int inx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.

amax_strided_batched finds the first index of the element of maximum magnitude of each vector x_i in a batch, for i = 1, ..., batch_count.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in each vector x_i
• [in] x: device pointer to the first vector x_1.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
• [in] stridex: [rocblas_stride] specifies the pointer increment between one x_i and the next x_(i + 1).
• [in] batch_count: [rocblas_int] number of instances in the batch
• [out] result: device or host pointer for storing contiguous batch_count results. return is 0 if n <= 0, incx<=0.

rocblas_status rocblas_idamax_strided_batched (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_icamax_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_izamax_strided_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_isamin (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_int *result)

BLAS Level 1 API.

amin finds the first index of the element of minimum magnitude of a vector x.

vector

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] the number of elements in x.
• [in] x: device pointer storing vector x.
• [in] incx: [rocblas_int] specifies the increment for the elements of y.
• [inout] result: device pointer or host pointer to store the amin index. return is 0.0 if n, incx<=0.

rocblas_status rocblas_idamin (rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_int *result)

rocblas_status rocblas_icamin (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_int *result)

rocblas_status rocblas_izamin (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_int *result)
rocblas_status rocblas_isamin_batched (rocblas_handle handle, rocblas_int n, const float *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.
amin_batched finds the first index of the element of minimum magnitude of each vector x_i in a batch, for i = 1, \ldots, batch_count.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each vector x_i.
- [in] x: device array of device pointers storing each vector x_i.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- [in] batch_count: [rocblas_int] number of instances in the batch, must be > 0.
- [out] result: device or host pointers to array of batch_count size for results. return is 0 if n, incx<=0.

rocblas_status rocblas_idamin_batched (rocblas_handle handle, rocblas_int n, const double *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_icamin_batched (rocblas_handle handle, rocblas_int n, const rocblas_float_complex *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_izamin_batched (rocblas_handle handle, rocblas_int n, const rocblas_double_complex *const x[], rocblas_int incx, rocblas_int batch_count, rocblas_int *result)
rocblas_status rocblas_isamin_strided_batched (rocblas_handle handle, rocblas_int n, const float *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

BLAS Level 1 API.
amin_strided_batched finds the first index of the element of minimum magnitude of each vector x_i in a batch, for i = 1, \ldots, batch_count.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in each vector x_i.
- [in] x: device pointer to the first vector x_1.
- [in] incx: [rocblas_int] specifies the increment for the elements of each x_i. incx must be > 0.
- [in] stridex: [rocblas_stride] specifies the pointer increment between one x_i and the next x_(i + 1)
- [in] batch_count: [rocblas_int] number of instances in the batch
- [out] result: device or host pointer to array for storing contiguous batch_count results. return is 0 if n <= 0, incx<=0.
rocblas_status rocblas_idamin_strided_batched(rocblas_handle handle, rocblas_int n, const double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_icamin_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_izamin_strided_batched(rocblas_handle handle, rocblas_int n, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count, rocblas_int *result)

rocblas_status rocblas_srot (rocblas_handle handle, rocblas_int n, float *x, rocblas_int incx, float *y, rocblas_int incy, const float *c, const float *s)

rocblas_status rocblas_drot (rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, double *y, rocblas_int incy, const double *c, const double *s)

rocblas_status rocblas_crot (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy, const rocblas_float_complex *c, const rocblas_float_complex *s)

rocblas_status rocblas_csrot (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx, rocblas_float_complex *y, rocblas_int incy, const float *c, const float *s)

rocblas_status rocblas_zrot (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy, const double *c, const double *s)

rocblas_status rocblas_zdrot (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_double_complex *y, rocblas_int incy, const double *c, const double *s)

BLAS Level 1 API.

rot applies the Givens rotation matrix defined by c=cos(alpha) and s=sin(alpha) to vectors x and y. Scalars c and s may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] n: [rocblas_int] number of elements in the x and y vectors.
- [inout] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment between elements of x.
- [inout] y: device pointer storing vector y.
- [in] incy: [rocblas_int] specifies the increment between elements of y.
- [in] c: device pointer or host pointer storing scalar cosine component of the rotation matrix.
- [in] s: device pointer or host pointer storing scalar sine component of the rotation matrix.
**rocblas_status rocblas_srot_batched**

ROCBLAS API.

**rocblas_srot_batched**

Rotates the vectors **x** and **y** by the Givens rotation matrix defined by \( c = \cos(\alpha) \) and \( s = \sin(\alpha) \) for \( i = 1, \ldots, \text{batch\_count} \). Scalars \( c \) and \( s \) may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

**Parameters**

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** \( n \): [rocblas_int] number of elements in each \( x \_i \) and \( y \_i \) vectors.
- **[inout]** \( x \): device array of device pointers storing each vector \( x \_i \).
- **[in]** \( \text{incx} \): [rocblas_int] specifies the increment between elements of each \( x \_i \).
- **[inout]** \( y \): device array of device pointers storing each vector \( y \_i \).
- **[in]** \( \text{incy} \): [rocblas_int] specifies the increment between elements of each \( y \_i \).
- **[in]** \( c \): device pointer or host pointer to scalar cosine component of the rotation matrix.
- **[in]** \( s \): device pointer or host pointer to scalar sine component of the rotation matrix.
- **[in]** \( \text{batch\_count} \): [rocblas_int] the number of \( x \) and \( y \) arrays, i.e. the number of batches.

**rocblas_status rocblas_drot_batched**

**rocblas_status rocblas_crot_batched**

**rocblas_status rocblas_csrot_batched**

**rocblas_status rocblas_zrot_batched**

**rocblas_status rocblas_zdrot_batched**

**rocblas_status rocblas_srot_strided_batched**
BLAS Level 1 API.

rot_strided_batched applies the Givens rotation matrix defined by $c=\cos(\alpha)$ and $s=\sin(\alpha)$ to strided batched vectors $x_i$ and $y_i$, for $i = 1, \ldots, \text{batch\_count}$. Scalars $c$ and $s$ may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] $n$: [rocblas_int] number of elements in each $x_i$ and $y_i$ vectors.
- [inout] $x$: device pointer to the first vector $x_1$.
- [in] $\text{incx}$: [rocblas_int] specifies the increment between elements of each $x_i$.
- [in] stride_x: [rocblas_stride] specifies the increment from the beginning of $x_i$ to the beginning of $x_{(i+1)}$.
- [inout] $y$: device pointer to the first vector $y_1$.
- [in] $\text{incy}$: [rocblas_int] specifies the increment between elements of each $y_i$.
- [in] stride_y: [rocblas_stride] specifies the increment from the beginning of $y_i$ to the beginning of $y_{(i+1)}$.
- [in] $c$: device pointer or host pointer to scalar cosine component of the rotation matrix.
- [in] $s$: device pointer or host pointer to scalar sine component of the rotation matrix.
- [in] batch_count: [rocblas_int] the number of $x$ and $y$ arrays, i.e. the number of batches.

rocblas_status rocblas_drot_strided_batched
    (rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, rocblas_stride stride_x,
     double *y, rocblas_int incy, rocblas_stride stride_y, const double *c, const double *s,
     rocblas_int batch_count)

rocblas_status rocblas_crot_strided_batched
    (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx,
     rocblas_stride stride_x, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stride_y,
     const float *c, const rocblas_float_complex *s, rocblas_int batch_count)

rocblas_status rocblas_csrot_strided_batched
    (rocblas_handle handle, rocblas_int n, rocblas_float_complex *x, rocblas_int incx,
     rocblas_stride stride_x, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stride_y,
     const float *c, const rocblas_float_complex *s, rocblas_int batch_count)

rocblas_status rocblas_zrot_strided_batched
    (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx,
     rocblas_stride stride_x, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stride_y,
     const double *c, const rocblas_double_complex *s, rocblas_int batch_count)
rocblas_status rocblas_zdrot_strided_batched (rocblas_handle handle, rocblas_int n, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stride_y, const double *c, const double *s, rocblas_int batch_count)

rocblas_status rocblas_srotg (rocblas_handle handle, float *a, float *b, float *c, float *s)

rocblas_status rocblas_drotg (rocblas_handle handle, double *a, double *b, double *c, double *s)

rocblas_status rocblas_crotg (rocblas_handle handle, rocblas_float_complex *a, rocblas_float_complex *b, float *c, rocblas_float_complex *s)

rocblas_status rocblas_zrotg (rocblas_handle handle, rocblas_double_complex *a, rocblas_double_complex *b, double *c, rocblas_double_complex *s)

rocblas_status rocblas_srotg_batched (rocblas_handle handle, float *const a[], float *const b[], float *const c[], float *const s[], rocblas_int batch_count)

rocblas_status rocblas_drotg_batched (rocblas_handle handle, float *const a[], float *const b[], float *const c[], float *const s[], rocblas_int batch_count)

rocblas_status rocblas_crotg_batched (rocblas_handle handle, rocblas_float_complex *a[], rocblas_float_complex *b[], float *const c[], float *const s[], rocblas_int batch_count)

rocblas_status rocblas_zrotg_batched (rocblas_handle handle, rocblas_double_complex *a[], rocblas_double_complex *b[], double *c[], rocblas_double_complex *s[], rocblas_int batch_count)

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [inout] a: device pointer or host pointer to input vector element, overwritten with r.
- [inout] b: device pointer or host pointer to input vector element, overwritten with z.
- [inout] c: device pointer or host pointer to cosine element of Givens rotation.
- [inout] s: device pointer or host pointer sine element of Givens rotation.

BLAS Level 1 API.

rotg creates the Givens rotation matrix for the vector (a b). Scalars c and s and arrays a and b may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [inout] a: device pointer or host pointer to input vector element, overwritten with r.
- [inout] b: device pointer or host pointer to input vector element, overwritten with z.
- [inout] c: device pointer or host pointer to cosine element of Givens rotation.
- [inout] s: device pointer or host pointer sine element of Givens rotation.

BLAS Level 1 API.

rotg_batched creates the Givens rotation matrix for the batched vectors (a_i b_i), for i = 1, ..., batch_count. a, b, c, and s may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [inout] a: device array of device pointers storing each single input vector element a_i, overwritten with r_i.
- [inout] b: device array of device pointers storing each single input vector element b_i, overwritten with z_i.
- [inout] c: device array of device pointers storing each cosine element of Givens rotation for the batch.
• [inout] \( s \): device array of device pointers storing each sine element of Givens rotation for the batch.

• [in] \( \text{batch\_count} \): [rocblas\_int] number of batches (length of arrays \( a, b, c, \) and \( s \)).

\[
\text{rocblas\_status}\ rocblas\_drotg\_batched (rocblas\_handle handle, double *\text{const} a[], double *\text{const} b[], double *\text{const} c[], double *\text{const} s[], \text{rocblas\_int batch\_count})
\]

\[
\text{rocblas\_status}\ rocblas\_crotg\_batched (rocblas\_handle handle, rocblas\_float\_complex *\text{const} a[], rocblas\_float\_complex *\text{const} b[], float *\text{const} c[], rocblas\_float\_complex *\text{const} s[], \text{rocblas\_int batch\_count})
\]

\[
\text{rocblas\_status}\ rocblas\_zrotg\_batched (rocblas\_handle handle, rocblas\_double\_complex *\text{const} a[], rocblas\_double\_complex *\text{const} b[], double *\text{const} c[], rocblas\_double\_complex *\text{const} s[], \text{rocblas\_int batch\_count})
\]

\[
\text{rocblas\_status}\ rocblas\_srotg\_strided\_batched (rocblas\_handle \text{handle}, float *\text{const} a, float *\text{const} b, float *\text{const} c, float *\text{const} s, \text{rocblas\_int batch\_count})
\]

BLAS Level 1 API.

rotg\_strided\_batched creates the Givens rotation matrix for the strided batched vectors \((a_i, b_i)\), for \( i = 1, \ldots, \text{batch\_count} \). \( a, b, c, \) and \( s \) may be stored in either host or device memory, location is specified by calling rocblas\_set\_pointer\_mode. If the pointer mode is set to rocblas\_pointer\_mode\_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas\_pointer\_mode\_device, this function returns immediately and synchronization is required to read the results.

Parameters

• [in] \text{handle}: [rocblas\_handle] handle to the rocblas library context queue.

• [inout] \text{a}: device strided\_batched pointer or host strided\_batched pointer to first single input vector element \( a_1 \), overwritten with \( r \).

• [in] \text{stride\_a}: [rocblas\_stride] distance between elements of \( a \) in batch (distance between \( a_i \) and \( a_{i+1} \))

• [inout] \text{b}: device strided\_batched pointer or host strided\_batched pointer to first single input vector element \( b_1 \), overwritten with \( z \).

• [in] \text{stride\_b}: [rocblas\_stride] distance between elements of \( b \) in batch (distance between \( b_i \) and \( b_{i+1} \))

• [inout] \text{c}: device strided\_batched pointer or host strided\_batched pointer to first cosine element of Givens rotations \( c_1 \).

• [in] \text{stride\_c}: [rocblas\_stride] distance between elements of \( c \) in batch (distance between \( c_i \) and \( c_{i+1} \))

• [inout] \text{s}: device strided\_batched pointer or host strided\_batched pointer to sine element of Givens rotations \( s_1 \).

• [in] \text{stride\_s}: [rocblas\_stride] distance between elements of \( s \) in batch (distance between \( s_i \) and \( s_{i+1} \))
• [in] batch_count: [rocblas_int] number of batches (length of arrays a, b, c, and s).

rocblas_status rocblas_drotg_strided_batched (rocblas_handle handle,
rocblas_stride stride_a, double *a,
rocblas_stride stride_b, double *b,
rocblas_stride stride_c, double *c,
rocblas_stride stride_s, rocblas_int batch_count)

rocblas_status rocblas_crotg_strided_batched (rocblas_handle handle,
rocblas_float_complex *a, rocblas_stride stride_a,
rocblas_float_complex *b, rocblas_stride stride_b,
float *c, rocblas_stride stride_c,
rocblas_float_complex *s, rocblas_stride stride_s, rocblas_int batch_count)

rocblas_status rocblas_zrotg_strided_batched (rocblas_handle handle,
rocblas_double_complex *a, rocblas_stride stride_a,
rocblas_double_complex *b, rocblas_stride stride_b,
double *c, rocblas_stride stride_c,
rocblas_double_complex *s, rocblas_stride stride_s, rocblas_int batch_count)

rocblas_status rocblas_srotm (rocblas_handle handle, rocblas_int n,
float *x, rocblas_int incx, float *y, rocblas_int incy, const float *param)

BLAS Level 1 API.

rotm applies the modified Givens rotation matrix defined by param to vectors x and y.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] n: [rocblas_int] number of elements in the x and y vectors.
• [inout] x: device pointer storing vector x.
• [in] incx: [rocblas_int] specifies the increment between elements of x.
• [inout] y: device pointer storing vector y.
• [in] incy: [rocblas_int] specifies the increment between elements of y.
• [in] param: device vector or host vector of 5 elements defining the rotation. param[0] = flag
  form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1
  => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may be stored in either
  host or device memory, location is specified by calling rocblas_set_pointer_mode.

rocblas_status rocblas_drotm (rocblas_handle handle, rocblas_int n,
double *x, rocblas_int incx, double *y, rocblas_int incy, const double *param)

rocblas_status rocblas_srotm_batched (rocblas_handle handle, rocblas_int n,
float *const x[], rocblas_int incx, float *const y[], rocblas_int incy,
const float *const param[], rocblas_int batch_count)

BLAS Level 1 API.

rotm_batched applies the modified Givens rotation matrix defined by param_i to batched vectors x_i and
y_i, for i = 1, ..., batch_count.
Parameters

- **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.
- **[in] n**: [rocblas_int] number of elements in the x and y vectors.
- **[inout] x**: device array of device pointers storing each vector x_i.
- **[in] incx**: [rocblas_int] specifies the increment between elements of each x_i.
- **[inout] y**: device array of device pointers storing each vector y_1.
- **[in] incy**: [rocblas_int] specifies the increment between elements of each y_i.
- **[in] param**: device array of device vectors of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may ONLY be stored on the device for the batched version of this function.
- **[in] batch_count**: [rocblas_int] the number of x and y arrays, i.e. the number of batches.

\[
\text{rocblas_status rocblas_drotm_batched} (\text{rocblas_handle \ handle, rocblas_int n, double *const x[], rocblas_int incx, double *const y[], rocblas_int incy, const double *const param[], rocblas_int batch_count})
\]

\[
\text{rocblas_status rocblas_srotm_strided_batched} (\text{rocblas_handle \ handle, rocblas_int n, float *x, rocblas_int incx, rocblas_stride stride_x, float *y, rocblas_int incy, rocblas_stride stride_y, const float *param, rocblas_stride stride_param, rocblas_int batch_count})
\]

BLAS Level 1 API.

rotn_strided_batched applies the modified Givens rotation matrix defined by param_i to strided batched vectors x_i and y_i, for i = 1, ..., batch_count

Parameters

- **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.
- **[in] n**: [rocblas_int] number of elements in the x and y vectors.
- **[inout] x**: device pointer pointing to first strided batched vector x_1.
- **[in] incx**: [rocblas_int] specifies the increment between elements of each x_i.
- **[in] stride_x**: [rocblas_stride] specifies the increment between the beginning of x_i and x_(i+1)
- **[inout] y**: device pointer pointing to first strided batched vector y_1.
- **[in] incy**: [rocblas_int] specifies the increment between elements of each y_i.
- **[in] stride_y**: [rocblas_stride] specifies the increment between the beginning of y_i and y_(i+1)
- **[in] param**: device pointer pointing to first array of 5 elements defining the rotation (param_1). param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = ( H11 H12 H21 H22 ) flag = 0 => H = ( 1.0 H12 H21 1.0 ) flag = 1 => H = ( H11 1.0 -1.0 H22 ) flag = -2 => H = ( 1.0 0.0 0.0 1.0 ) param may ONLY be stored on the device for the strided_batched version of this function.
• [in] stride_param: [rocblas_stride] specifies the increment between the beginning of param_i and param_(i + 1)
• [in] batch_count: [rocblas_int] the number of x and y arrays, i.e. the number of batches.

rocblas_status rocblas_drotm_strided_batched (rocblas_handle handle, rocblas_int n, double *x, rocblas_int incx, rocblas_stride stride_x, double *y, rocblas_int incy, rocblas_stride stride_y, const double *param, rocblas_stride stride_param, rocblas_int batch_count)

rocblas_status rocblas_srotmg (rocblas_handle handle, float *d1, float *d2, float *x1, const float *y1, float *param)

BLAS Level 1 API.

rotmg creates the modified Givens rotation matrix for the vector (d1 * x1, d2 * y1). Parameters may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters
• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [inout] d1: device pointer or host pointer to input scalar that is overwritten.
• [inout] d2: device pointer or host pointer to input scalar that is overwritten.
• [in] y1: device pointer or host pointer to input scalar.
• [out] param: device vector or host vector of 5 elements defining the rotation. param[0] = flag param[1] = H11 param[2] = H21 param[3] = H12 param[4] = H22 The flag parameter defines the form of H: flag = -1 => H = \begin{pmatrix} H11 & H12 \\ H21 & H22 \end{pmatrix} flag = 0 => H = \begin{pmatrix} 1 & H12 \\ H21 & 1 \end{pmatrix} flag = 1 => H = \begin{pmatrix} H11 & 1 \\ -1 & H22 \end{pmatrix} flag = -2 => H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} param may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

rocblas_status rocblas_drotmg (rocblas_handle handle, double *d1, double *d2, double *x1, const double *y1, double *param)

rocblas_status rocblas_srotmg_batched (rocblas_handle handle, float *const d1[], float *const d2[], float *const x1[], const float *const y1[], float *const param[], rocblas_int batch_count)

BLAS Level 1 API.

rotmg_batched creates the modified Givens rotation matrix for the batched vectors (d1_i * x1_i, d2_i * y1_i), for i = 1, ..., batch_count. Parameters may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters
• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [inout] d1: device batched array or host batched array of input scalars that is overwritten.
• [inout] d2: device batched array or host batched array of input scalars that is overwritten.
- [inout] \( x_1 \): device batched array or host batched array of input scalars that is overwritten.
- [in] \( y_1 \): device batched array or host batched array of input scalars.
- [out] \( \text{param} \): device batched array or host batched array of vectors of 5 elements defining the rotation. \( \text{param}[0] = \text{flag} \) \( \text{param}[1] = H_{11} \) \( \text{param}[2] = H_{21} \) \( \text{param}[3] = H_{12} \) \( \text{param}[4] = H_{22} \)

The flag parameter defines the form of \( H \): 

\( \text{flag} = -1 \Rightarrow H = ( H_{11} \ H_{12} \ H_{21} \ H_{22} ) \) 
\( \text{flag} = 0 \Rightarrow H = ( 1.0 \ H_{12} \ H_{21} \ 1.0 ) \) 
\( \text{flag} = 1 \Rightarrow H = ( H_{11} \ 1.0 \ -1.0 \ H_{22} ) \) 
\( \text{flag} = -2 \Rightarrow H = ( 1.0 \ 0.0 \ 0.0 \ 1.0 ) \)

\( \text{param} \) may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

- [in] \( \text{batch_count} \): [rocblas_int] the number of instances in the batch.

\[
\text{rocblas_status \ rocblas_drotmg_batched}(\text{rocblas_handle \ handle}, \ \text{double *d1[]}, \ \text{double *const d2[]}, \ \text{double *const x1[]}, \ \text{const double *const y1[]}, \ \text{double *const param[]}, \ \text{rocblas_int batch_count})
\]

\[
\text{rocblas_status \ rocblas_srotmg_strided_batched}(\text{rocblas_handle \ handle}, \ \text{float *d1}, \ \text{rocblas_stride stride_d1}, \ \text{float *d2}, \ \text{rocblas_stride stride_d2}, \ \text{float *x1}, \ \text{rocblas_stride stride_x1}, \ \text{const float *y1}, \ \text{rocblas_stride stride_y1}, \ \text{float *param}, \ \text{rocblas_stride stride_param}, \ \text{rocblas_int batch_count})
\]

BLAS Level 1 API.

rotmg_strided_batched creates the modified Givens rotation matrix for the strided batched vectors \( (d1_i * x1_i, d2_i * y1_i) \), for \( i = 1, \ldots, \text{batch_count} \). Parameters may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode. If the pointer mode is set to rocblas_pointer_mode_host, this function blocks the CPU until the GPU has finished and the results are available in host memory. If the pointer mode is set to rocblas_pointer_mode_device, this function returns immediately and synchronization is required to read the results.

Parameters

- [in] \( \text{handle} \): [rocblas_handle] handle to the rocblas library context queue.
- [inout] \( d1 \): device strided_batched array or host strided_batched array of input scalars that is overwritten.
- [in] \( \text{stride}_d1 \): [rocblas_stride] specifies the increment between the beginning of \( d1_i \) and \( d1_{(i+1)} \)
- [inout] \( d2 \): device strided_batched array or host strided_batched array of input scalars that is overwritten.
- [in] \( \text{stride}_d2 \): [rocblas_stride] specifies the increment between the beginning of \( d2_i \) and \( d2_{(i+1)} \)
- [inout] \( x1 \): device strided_batched array or host strided_batched array of input scalars that is overwritten.
- [in] \( \text{stride}_x1 \): [rocblas_stride] specifies the increment between the beginning of \( x1_i \) and \( x1_{(i+1)} \)
- [in] \( y1 \): device strided_batched array or host strided_batched array of input scalars.
- [in] \( \text{stride}_y1 \): [rocblas_stride] specifies the increment between the beginning of \( y1_i \) and \( y1_{(i+1)} \)
  - flag = -1 => H = ( H11 H12 H21 H22 )
  - flag = 0 => H = ( 1.0 H12 H21 1.0 )
  - flag = 1 => H = ( H11 1.0 0.0 -1.0 )
  - flag = -2 => H = ( 1.0 0.0 0.0 1.0 )
param may be stored in either host or device memory, location is specified by calling rocblas_set_pointer_mode.

• **[in] stride_param**: [rocblas_stride] specifies the increment between the beginning of param_i and param_(i + 1)

• **[in] batch_count**: [rocblas_int] the number of instances in the batch.

```c
rocblas_status rocblas_drotmg_strided_batched(rocblas_handle handle, double *d1,
                                             rocblas_stride stride_d1, double *d2,
                                             rocblas_stride stride_d2, double *x1,
                                             rocblas_stride stride_x1, const double *y1,
                                             rocblas_stride stride_y1, double *param,
                                             rocblas_stride stride_param,
                                             rocblas_int batch_count)
```

```c
rocblas_status rocblas_sgemv(rocblas_handle handle, rocblas_operation trans, rocblas_int m,
                              rocblas_int n, const float *alpha, const float *A, rocblas_int lda,
                              const float *x, rocblas_int incx, const float *beta, float *y,
                              rocblas_int incy)
```

BLAS Level 2 API.

xGEMV performs one of the matrix-vector operations

\[
\begin{align*}
\text{y} & := \alpha A x + \beta y, \quad \text{or} \\
\text{y} & := \alpha A^T x + \beta y, \quad \text{or} \\
\text{y} & := \alpha A^H x + \beta y,
\end{align*}
\]

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

**Parameters**

• **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.

• **[in] trans**: [rocblas_operation] indicates whether matrix A is tranposed (conjugated) or not

• **[in] m**: [rocblas_int] number of rows of matrix A

• **[in] n**: [rocblas_int] number of columns of matrix A

• **[in] alpha**: device pointer or host pointer to scalar alpha.

• **[in] A**: device pointer storing matrix A.

• **[in] lda**: [rocblas_int] specifies the leading dimension of A.

• **[in] x**: device pointer storing vector x.

• **[in] incx**: [rocblas_int] specifies the increment for the elements of x.

• **[in] beta**: device pointer or host pointer to scalar beta.

• **[inout] y**: device pointer storing vector y.

• **[in] incy**: [rocblas_int] specifies the increment for the elements of y.
rocblas_status rocblas_dgemv(rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, const double *x, rocblas_int incx, const double *beta, double *y, rocblas_int incy)

rocblas_status rocblas_cgemv(rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *x, rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy)

rocblas_status rocblas_zgemv(rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, const rocblas_double_complex *x, rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy)

rocblas_status rocblas_chemv(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *x, rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy)

BLAS Level 2 API.

xHEMV performs one of the matrix-vector operations

\[ y := \alpha A x + \beta y \]

where alpha and beta are scalars, x and y are n element vectors and A is an n by n hermitian matrix.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: A is an upper banded triangular matrix. rocblas_fill_lower: A is a lower banded triangular matrix.
- [in] n: [rocblas_int] the order of the matrix A.
- [in] alpha: device pointer or host pointer to scalar alpha.
- [in] A: device pointer storing matrix A. Of dimension (lda, n), if uplo == rocblas_fill_upper: The upper triangular part of A must contain the upper triangular part of a hermitian matrix. The lower triangular part of A will not be referenced. if uplo == rocblas_fill_lower: The lower triangular part of A must contain the lower triangular part of a hermitian matrix. The upper triangular part of A will not be referenced. As a hermitian matrix, the imaginary part of the main diagonal of A will not be referenced and is assumed to be == 0.
- [in] lda: [rocblas_int] specifies the leading dimension of A. must be >= max(1, n)
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
- [in] beta: device pointer or host pointer to scalar beta.
- [inout] y: device pointer storing vector y.
- [in] incy: [rocblas_int] specifies the increment for the elements of y.
rocblas_status rocblas_zhemv(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, const rocblas_double_complex *x, rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy)

rocblas_status rocblas_chemv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *const A[], rocblas_int lda, const rocblas_float_complex *const x[], rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)

BLAS Level 2 API.

xHEMV_BATCHED performs one of the matrix-vector operations

\[ y_i := \alpha A_i x_i + \beta y_i \]

where alpha and beta are scalars, \( x_i \) and \( y_i \) are \( n \) element vectors and \( A_i \) is an \( n \) by \( n \) hermitian matrix, for each batch in \( i = [1, \text{batch\_count}] \).

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: \( A \) is an upper banded triangular matrix. rocblas_fill_lower: \( A \) is a lower banded triangular matrix.
- [in] n: [rocblas_int] the order of each matrix \( A_i \).
- [in] alpha: device pointer or host pointer to scalar alpha.
- [in] A: device array of device pointers storing each matrix \( A_i \) of dimension (lda, n). if uplo == rocblas_fill_upper: The upper triangular part of each \( A_i \) must contain the upper triangular part of a hermitian matrix. The lower triangular part of each \( A_i \) will not be referenced. if uplo == rocblas_fill_lower: The lower triangular part of each \( A_i \) must contain the lower triangular part of a hermitian matrix. The upper triangular part of each \( A_i \) will not be referenced. As a hermitian matrix, the imaginary part of the main diagonal of each \( A_i \) will not be referenced and is assumed to be == 0.
- [in] lda: [rocblas_int] specifies the leading dimension of each \( A_i \). must be >= max(1, n)
- [in] x: device array of device pointers storing each vector \( x_i \).
- [in] incx: [rocblas_int] specifies the increment for the elements of each \( x_i \).
- [in] beta: device pointer or host pointer to scalar beta.
- [inout] y: device array of device pointers storing each vector \( y_i \).
- [in] incy: [rocblas_int] specifies the increment for the elements of \( y \).
- [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_zhemv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *const A[], rocblas_int lda, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_chemv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stride_A, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stride_y, rocblas_int batch_count)

BLAS Level 2 API.

xHEMV_STRIDED_BATCHED performs one of the matrix-vector operations

\[ y_i := \alpha A_i x_i + \beta y_i \]

where alpha and beta are scalars, \( x_i \) and \( y_i \) are \( n \) element vectors and \( A_i \) is an \( n \) by \( n \) hermitian matrix, for each batch in \( i = [1, \text{batch\_count}] \).

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: \( A \) is an upper banded triangular matrix. rocblas_fill_lower: \( A \) is a lower banded triangular matrix.
- [in] n: [rocblas_int] the order of each matrix \( A_i \).
- [in] alpha: device pointer or host pointer to scalar alpha.
- [in] A: device array of device pointers storing each matrix \( A_i \) of dimension (lda, n). if uplo == rocblas_fill_upper: The upper triangular part of each \( A_i \) must contain the upper triangular part of a hermitian matrix. The lower triangular part of each \( A_i \) will not be referenced. if uplo == rocblas_fill_lower: The lower triangular part of each \( A_i \) must contain the lower triangular part of a hermitian matrix. The upper triangular part of each \( A_i \) will not be referenced. As a hermitian matrix, the imaginary part of the main diagonal of each \( A_i \) will not be referenced and is assumed to be == 0.
- [in] lda: [rocblas_int] specifies the leading dimension of each \( A_i \). must be >= max(1, n)
- [in] x: device array of device pointers storing each vector \( x_i \).
- [in] incx: [rocblas_int] specifies the increment for the elements of each \( x_i \).
- [in] beta: device pointer or host pointer to scalar beta.
- [inout] y: device array of device pointers storing each vector \( y_i \).
- [in] incy: [rocblas_int] specifies the increment for the elements of \( y \).
- [in] batch_count: [rocblas_int] number of instances in the batch.
### rocblas_zhemv_strided_batched

- **rocblas_status rocblas_zhemv_strided_batched**
  - (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stride_A, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stride_y, rocblas_int batch_count)

### rocblas_sgemv_batched

- **rocblas_status rocblas_sgemv_batched**
  - (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const float *alpha, const float *const A[], rocblas_int lda, const float *const x[], rocblas_int incx, const float *beta, float *const y[], rocblas_int incy, rocblas_int batch_count)

#### BLAS Level 2 API.

**xGEMV_BATCHED** performs a batch of matrix-vector operations

\[
y_{i} := \alpha A_{i} x_{i} + \beta y_{i}, \quad \text{or} \quad y_{i} := \alpha A_{i}^{T} x_{i} + \beta y_{i}, \quad \text{or} \quad y_{i} := \alpha A_{i}^{H} x_{i} + \beta y_{i},
\]

where \((A_{i}, x_{i}, y_{i})\) is the \(i\)-th instance of the batch. \(\alpha\) and \(\beta\) are scalars, \(x_{i}\) and \(y_{i}\) are vectors and \(A_{i}\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch}\_\text{count}\).

#### Parameters

- **[in]** handle: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** trans: [rocblas_operation] indicates whether matrices \(A_{i}\) are tranposed (conjugated) or not
- **[in]** m: [rocblas_int] number of rows of each matrix \(A_{i}\)
- **[in]** n: [rocblas_int] number of columns of each matrix \(A_{i}\)
- **[in]** alpha: device pointer or host pointer to scalar \(\alpha\).
- **[in]** A: device array of device pointers storing each matrix \(A_{i}\).
- **[in]** lda: [rocblas_int] specifies the leading dimension of each matrix \(A_{i}\).
- **[in]** x: device array of device pointers storing each vector \(x_{i}\).
- **[in]** incx: [rocblas_int] specifies the increment for the elements of each vector \(x_{i}\).
- **[in]** beta: device pointer or host pointer to scalar \(\beta\).
- **[inout]** y: device array of device pointers storing each vector \(y_{i}\).
- **[in]** incy: [rocblas_int] specifies the increment for the elements of each vector \(y_{i}\).
- **[in]** batch_count: [rocblas_int] number of instances in the batch

### rocblas_dgemv_batched

- **rocblas_status rocblas_dgemv_batched**
  - (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const double *alpha, const double *const A[], rocblas_int lda, const double *const x[], rocblas_int incx, const double *beta, double *const y[], rocblas_int incy, rocblas_int batch_count)
rocblas_status rocblas_cgemv_batched (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *const A[], rocblas_int lda, const rocblas_float_complex *const x[], rocblas_int incx, const rocblas_float_complex *beta, rocblas_float_complex *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_zgemv_batched (rocblas_handle handle, rocblas_operation trans, rocblas_int m, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *const A[], rocblas_int lda, const rocblas_double_complex *const x[], rocblas_int incx, const rocblas_double_complex *beta, rocblas_double_complex *const y[], rocblas_int incy, rocblas_int batch_count)

rocblas_status rocblas_sgemv_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, rocblas_stride strideA, const float *x, rocblas_int incx, rocblas_stride stridesx, const float *beta, float *y, rocblas_int incy, rocblas_stride stridesy, rocblas_int batch_count)

BLAS Level 2 API.

xGEMV_STRIDED_BATCHED performs a batch of matrix-vector operations

\[
\begin{align*}
y_i & := \alpha A_{i} x_i + \beta y_i, \\
y_i & := \alpha A_{i}^T x_i + \beta y_i, \\
y_i & := \alpha A_{i}^H x_i + \beta y_i,
\end{align*}
\]

where \((A_i, x_i, y_i)\) is the i-th instance of the batch. \(\alpha\) and \(\beta\) are scalars, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] indicates whether matrices \(A_i\) are tranposed (conjugated) or not
- [in] m: [rocblas_int] number of rows of matrices \(A_i\)
- [in] n: [rocblas_int] number of columns of matrices \(A_i\)
- [in] alpha: device pointer or host pointer to scalar \(\alpha\).
- [in] A: device pointer to the first matrix \((A_1)\) in the batch.
- [in] lda: [rocblas_int] specifies the leading dimension of matrices \(A_i\).
- [in] strideA: [rocblas_stride] stride from the start of one matrix \((A_i)\) and the next one \((A_i+1)\)
- [in] x: device pointer to the first vector \((x_1)\) in the batch.
- [in] incx: [rocblas_int] specifies the increment for the elements of vectors \(x_i\).
• [in] stridex: [rocblas_stride] stride from the start of one vector (x_i) and the next one (x_i+1). There are no restrictions placed on stride_x, however the user should take care to ensure that stride_x is of appropriate size. When trans equals rocblas_operation_none this typically means stride_x >= n * incx, otherwise stride_x >= m * incx.

• [in] beta: device pointer or host pointer to scalar beta.

• [inout] y: device pointer to the first vector (y_1) in the batch.

• [in] incy: [rocblas_int] specifies the increment for the elements of vectors y_i.

• [in] stridy: [rocblas_stride] stride from the start of one vector (y_i) and the next one (y_i+1). There are no restrictions placed on stride_y, however the user should take care to ensure that stride_y is of appropriate size. When trans equals rocblas_operation_none this typically means stride_y >= m * incy, otherwise stride_y >= n * incy. stridy should be non zero.

• [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dgemv_strided_batched
(rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, rocblas_stride strideA, const double *x, rocblas_int incx, rocblas_stride stridex, const double *beta, double *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_cgemv_strided_batched
(rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride strideA, const rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_float_complex *beta, rocblas_float_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_zgemv_strided_batched
(rocblas_handle handle, rocblas_operation transA, rocblas_int m, rocblas_int n, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride strideA, const rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, const rocblas_double_complex *beta, rocblas_double_complex *y, rocblas_int incy, rocblas_stride stridey, rocblas_int batch_count)

rocblas_status rocblas_strmv
(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, float *x, rocblas_int incx)

BLAS Level 2 API.

trmv performs one of the matrix-vector operations
\[ x = A \cdot x \text{ or } x = A^* \cdot x, \]

where \( x \) is an \( n \) element vector and \( A \) is an \( n \) by \( n \) unit, or non-unit, upper or lower triangular matrix.

The vector \( x \) is overwritten.

**Parameters**

- \([\text{in}]\) handle: [rocblas_handle] handle to the rocblas library context queue.
- \([\text{in}]\) uplo: [rocblas_fill] rocblas_fill_upper: \( A \) is an upper triangular matrix. rocblas_fill_lower: \( A \) is a lower triangular matrix.
- \([\text{in}]\) transA: [rocblas_operation]
- \([\text{in}]\) diag: [rocblas_diagonal] rocblas_diagonal_unit: \( A \) is assumed to be unit triangular. rocblas_diagonal_non_unit: \( A \) is not assumed to be unit triangular.
- \([\text{in}]\) m: [rocblas_int] \( m \) specifies the number of rows of \( A \). \( m \geq 0 \).
- \([\text{in}]\) A: device pointer storing matrix \( A \), of dimension ( \( lda, m \) )
- \([\text{in}]\) lda: [rocblas_int] specifies the leading dimension of \( A \). \( lda = \max(1, m) \).
- \([\text{in}]\) x: device pointer storing vector \( x \).
- \([\text{in}]\) incx: [rocblas_int] specifies the increment for the elements of \( x \).

```c
rocblas_status rocblas_dtrmv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, double *x, rocblas_int incx)

rocblas_status rocblas_ctrmv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_float_complex *A, rocblas_int lda, rocblas_float_complex *x, rocblas_int incx)

rocblas_status rocblas_ztrmv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_double_complex *A, rocblas_int lda, rocblas_double_complex *x, rocblas_int incx)

rocblas_status rocblas_strmv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, float *x, rocblas_int incx, rocblas_int batch_count)
```

**BLAS Level 2 API.**

trmv_batched performs one of the matrix-vector operations

\[ x_i = A_i \cdot x_i \text{ or } x_i = A_i^* \cdot x_i, \quad 0 \leq i < \text{batch_count} \]

where \( x_i \) is an \( n \) element vector and \( A_i \) is an \( n \) by \( n \) (unit, or non-unit, upper or lower triangular matrix)

The vectors \( x_i \) are overwritten.

**Parameters**

- \([\text{in}]\) handle: [rocblas_handle] handle to the rocblas library context queue.
- \([\text{in}]\) uplo: [rocblas_fill] rocblas_fill_upper: \( A_i \) is an upper triangular matrix. rocblas_fill_lower: \( A_i \) is a lower triangular matrix.
• [in] transA: [rocblas_operation]
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: A_i is assumed to be unit triangular. rocblas_diagonal_non_unit: A_i is not assumed to be unit triangular.
• [in] m: [rocblas_int] m specifies the number of rows of matrices A_i. m >= 0.
• [in] A: device pointer storing pointer of matrices A_i, of dimension ( lda, m )
• [in] lda: [rocblas_int] specifies the leading dimension of A_i. lda >= max( 1, m ).
• [in] x: device pointer storing vectors x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of vectors x_i.

rocblas_status rocblas_dtrmv_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *const *A, rocblas_int lda, double *const *x, rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_ctrmv_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_float_complex *const *A, rocblas_int lda, rocblas_float_complex *const *x, rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_ztrmv_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_double_complex *const *A, rocblas_int lda, rocblas_double_complex *const *x, rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_strmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *const *A, rocblas_int lda, rocblas_int strideA, float *x, rocblas_int incx, rocblas_int strideX, rocblas_int batch_count)

BLAS Level 2 API.

trmv_strided_batched performs one of the matrix-vector operations

\[
x_i = A_i * x_i \text{ or } x_i = A^T * x_i, \quad 0 \le i < \text{batch_count}
\]

where x_i is an n element vector and A_i is an n by n (unit, or non-unit, upper or lower triangular matrix) with strides specifying how to retrieve $x_i$ (resp. $A_i$) from $x_{i-1} S$ (resp. $A_{i-1} S$).

The vectors x_i are overwritten.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] uplo: [rocblas_fill] rocblas_fill_upper: A_i is an upper triangular matrix. rocblas_fill_lower: A_i is a lower triangular matrix.
• [in] transA: [rocblas_operation]
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: A_i is assumed to be unit triangular. rocblas_diagonal_non_unit: A_i is not assumed to be unit triangular.
• [in] `m`: [rocblas_int] `m` specifies the number of rows of matrices A_i. m >= 0.
• [in] `A`: device pointer of the matrix A_0, of dimension ( lda, m )
• [in] `lda`: [rocblas_int] specifies the leading dimension of A_i. lda >= max( 1, m ).
• [in] `stride_a`: [rocblas_stride] stride from the start of one A_i matrix to the next A_{i + 1}
• [in] `x`: device pointer storing the vector x_0.
• [in] `incx`: [rocblas_int] specifies the increment for the elements of one vector x.
• [in] `stride_x`: [rocblas_stride] stride from the start of one x_i vector to the next x_{i + 1}

```
rocblas_status rocblas_dtrmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, rocblas_stride stridea, double *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)
```

```
rocblas_status rocblas_ctrmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stridea, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)
```

```
rocblas_status rocblas_ztrmv_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stridea, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stridex, rocblas_int batch_count)
```

```
rocblas_status rocblas_stbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const float *A, rocblas_int lda, float *x, rocblas_int incx)
```

BLAS Level 2 API.

xTBMV performs one of the matrix-vector operations

\[
x := A \times x \quad \text{or} \\
x := A^T \times x \quad \text{or} \\
x := A^H \times x
\]

x is a vectors and A is a banded m by m matrix (see description below).

Parameters

• [in] `handle`: [rocblas_handle] handle to the rocblas library context queue.
• [in] `uplo`: [rocblas_fill] rocblas_fill_upper: A is an upper banded triangular matrix. rocblas_fill_lower: A is a lower banded triangular matrix.
• [in] `trans`: [rocblas_operation] indicates whether matrix A is tranposed (conjugated) or not.
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: The main diagonal of A is assumed to consist of only 1’s and is not referenced. rocblas_diagonal_non_unit: No assumptions are made of A’s main diagonal.

• [in] m: [rocblas_int] the number of rows and columns of the matrix represented by A.

• [in] k: [rocblas_int] if uplo == rocblas_fill_upper, k specifies the number of super-diagonals of the matrix A. if uplo == rocblas_fill_lower, k specifies the number of sub-diagonals of the matrix A. k must satisfy k > 0 && k < lda.

• [in] A: device pointer storing banded triangular matrix A. if uplo == rocblas_fill_upper: The matrix represented is an upper banded triangular matrix with the main diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the k’th row, the first super diagonal resides on the RHS of the k-1’th row, etc, with the k’th diagonal on the RHS of the 0’th row. Ex: (rocblas_fill_upper; m = 5; k = 2) 1 6 9 0 0 0 9 8 7 0 2 7 8 0 0 0 6 7 8 9 0 3 8 7 -> 1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 if uplo == rocblas_fill_lower: The matrix represented is a lower banded triangular matrix with the main diagonal and k sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the 0’th row, working up to the k’th diagonal residing on the LHS of the k’th row. Ex: (rocblas_fill_lower; m = 5; k = 2) 1 0 0 0 0 1 2 3 4 5 6 2 0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 0 0 0 0 0 7 9 5 0 0 0 0

• [in] lda: [rocblas_int] specifies the leading dimension of A. Lda must satisfy lda > k.

• [inout] x: device pointer storing vector x.

• [in] incx: [rocblas_int] specifies the increment for the elements of x.

rocblas_status rocblas_dtbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const double *A, rocblas_int lda, double *x, rocblas_int incx)

rocblas_status rocblas_ctbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_float_complex *A, rocblas_int lda, rocblas_float_complex *x, rocblas_int incx)

rocblas_status rocblas_ztbmv (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_double_complex *A, rocblas_int lda, rocblas_double_complex *x, rocblas_int incx)

rocblas_status rocblas_stbmv_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_double_complex *A[], rocblas_int lda, rocblas_double_complex *x[], rocblas_int incx, rocblas_int batch_count)

BLAS Level 2 API.

xTBMV_BATCHED performs one of the matrix-vector operations

\[
x_i := A_i*x_i \quad \text{or} \quad x_i := A_i^{**T}*x_i \quad \text{or} \quad x_i := A_i^{**H}*x_i,
\]

where \((A_i, x_i)\) is the i-th instance of the batch. \(x_i\) is a vector and \(A_i\) is an m by m matrix, for i = 1, ..., batch_count.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.

2.12. ROCm Libraries

201
• [in] uplo: [rocblas_fill] rocblas_fill_upper: each A_i is an upper banded triangular matrix. 
  rocblas_fill_lower: each A_i is a lower banded triangular matrix.
• [in] trans: [rocblas_operation] indicates whether each matrix A_i is tranposed (conjugated) 
  or not.
• [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: The main diagonal of each A_i is as- 
  sumed to consist of only 1’s and is not referenced. rocblas_diagonal_non_unit: No assumptions 
  are made of each A_i’s main diagonal.
• [in] m: [rocblas_int] the number of rows and columns of the matrix represented by each A_i.
• [in] k: [rocblas_int] if uplo == rocblas_fill_upper, k specifies the number of super-diagonals 
  of each matrix A_i. if uplo == rocblas_fill_lower, k specifies the number of sub-diagonals of each 
  matrix A_i. k must satisfy k > 0 & & k < lda.
• [in] A: device array of device pointers storing each banded triangular matrix A_i. if uplo == 
  rocblas_fill_upper: The matrix represented is an upper banded triangular matrix with the main 
  diagonal and k super-diagonals, everything else can be assumed to be 0. The matrix is compacted 
  so that the main diagonal resides on the k’th row, the first super diagonal resides on the RHS of 
  the k-1’th row, etc, with the k’th diagonal on the RHS of the 0’th row. Ex: (rocblas_fill_upper; m 
  = 5; k = 2) 1 6 0 0 0 0 9 8 7 0 2 7 8 0 0 6 7 8 9 0 3 8 7 -> 1 2 3 4 5 0 0 4 9 0 0 0 0 0 0 0 5 
  0 0 0 0 if uplo == rocblas_fill_lower: The matrix represented is a lower banded triangular matrix 
  with the main diagonal and k sub-diagonals, everything else can be assumed to be 0. The matrix 
  is compacted so that the main diagonal resides on the 0’th row, working up to the k’th diagonal 
  residing on the LHS of the k’th row. Ex: (rocblas_fill_lower; m = 5; k = 2) 1 0 0 0 0 1 2 3 4 5 6 2 
  0 0 0 6 7 8 9 0 9 7 3 0 0 -> 9 8 7 0 0 0 8 8 4 0 0 0 0 0 0 7 9 5 0 0 0 0 
• [in] lda: [rocblas_int] specifies the leading dimension of each A_i. lda must satisfy lda > k.
• [inout] x: device array of device pointer storing each vector x_i.
• [in] incx: [rocblas_int] specifies the increment for the elements of each x_i.
• [in] batch_count: [rocblas_int] number of instances in the batch.

rocblas_status rocblas_dtbmv_batched (rocblas_handle handle, 
rocblas_fill uplo, 
rocblas_operation trans, 
rocblas_diagonal diag, 
rocblas_int m, 
rocblas_int k, 
const double *const A[], 
rocblas_int lda, 
const double *const x[], 
rocblas_int incx, 
rocblas_int batch_count)

rocblas_status rocblas_ctbmv_batched (rocblas_handle handle, 
rocblas_fill uplo, 
rocblas_operation trans, 
rocblas_diagonal diag, 
rocblas_int m, 
rocblas_int k, 
const 
rocblas_float_complex *const A[], 
rocblas_int da, 
rocblas_float_complex *const x[], 
rocblas_int incx, 
rocblas_int batch_count)

rocblas_status rocblas_ztbmv_batched (rocblas_handle handle, 
rocblas_fill uplo, 
rocblas_operation trans, 
rocblas_diagonal diag, 
rocblas_int m, 
rocblas_int k, 
const 
rocblas_double_complex *const A[], 
rocblas_int lda, 
rocblas_double_complex *const x[], 
rocblas_int incx, 
rocblas_int batch_count)
rocblas_status rocblas_stbmv_strided_batched(
    rocblas_handle handle,
    rocblas_fill uplo,
    rocblas_operation trans,
    rocblas_diagonal diag,
    rocblas_int m, rocblas_int k, const
    float *A, rocblas_int lda, rocblas_stride stride_A,
    float *x, rocblas_int incx,
    rocblas_stride stride_x,
    rocblas_int
    batch_count)

BLAS Level 2 API.

xTBMV_STRIDED_BATCHED performs one of the matrix-vector operations

\[ x_i := A_i \times x_i \quad \text{or} \quad x_i := A_i^T \times x_i \quad \text{or} \quad x_i := A_i^H \times x_i, \]

where \((A_i, x_i)\) is the \(i\)-th instance of the batch. \(x_i\) is a vector and \(A_i\) is an \(m \times m\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

Parameters

- \([\text{in}]\) handle: [rocblas_handle] handle to the rocblas library context queue.
- \([\text{in}]\) uplo: [rocblas_fill] rocblas_fill_upper: each \(A_i\) is an upper banded triangular matrix. rocblas_fill_lower: each \(A_i\) is a lower banded triangular matrix.
- \([\text{in}]\) trans: [rocblas_operation] indicates whether each matrix \(A_i\) is tranposed (conjugated) or not.
- \([\text{in}]\) diag: [rocblas_diagonal] rocblas_diagonal_unit: The main diagonal of each \(A_i\) is assumed to consist of only 1’s and is not referenced. rocblas_diagonal_non_unit: No assumptions are made of each \(A_i\)’s main diagonal.
- \([\text{in}]\) m: [rocblas_int] the number of rows and columns of the matrix represented by each \(A_i\).
- \([\text{in}]\) k: [rocblas_int] if uplo == rocblas_fill_upper, \(k\) specifies the number of super-diagonals of each matrix \(A_i\). if uplo == rocblas_fill_lower, \(k\) specifies the number of sub-diagonals of each matrix \(A_i\). \(k\) must satisfy \(k > 0 && k < \text{lda}\).
- \([\text{in}]\) A: device array to the first matrix \(A_i\) of the batch. Stores each banded triangular matrix \(A_i\), if uplo == rocblas_fill_upper: The matrix represented is an upper banded triangular matrix with the main diagonal and \(k\) super-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the \(k\)’th row, the first super diagonal resides on the RHS of the \(k-1\)’th row, etc, with the \(k\)’th diagonal on the RHS of the 0’th row. Ex: (rocblas_fill_upper; \(m = 5; k = 2\)) \(1\ 6\ 9\ 0\ 0\ 0\ 9\ 8\ 7\ 0\ 2\ 7\ 8\ 0\ 0\ 6\ 7\ 8\ 9\ 0\ 3\ 8\ 7\) \(->\) \(1\ 2\ 3\ 4\ 5\ 0\ 0\ 4\ 9\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 5\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\)
  if uplo == rocblas_fill_lower: The matrix represented is a lower banded triangular matrix with the main diagonal and \(k\) sub-diagonals, everything else can be assumed to be 0. The matrix is compacted so that the main diagonal resides on the 0’th row, working up to the \(k\)’th diagonal residing on the LHS of the \(k\)’th row. Ex: (rocblas_fill_lower; \(m = 5; k = 2\)) \(1\ 0\ 0\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 2\ 0\ 0\ 0\ 6\ 7\ 8\ 9\ 0\ 9\ 7\ 3\ 0\ 0\) \(->\) \(9\ 8\ 7\ 0\ 0\ 8\ 8\ 4\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 7\ 9\ 5\ 0\ 0\ 0\ 0\ 0\ 0\ 0\)
- \([\text{in}]\) lda: [rocblas_int] specifies the leading dimension of each \(A_i\). lda must satisfy lda > \(k\).
- \([\text{in}]\) stride_A: [rocblas_stride] stride from the start of one \(A_i\) matrix to the next \(A_i(i + 1)\).
- \([\text{inout}]\) x: device array to the first vector \(x_i\) of the batch.
- \([\text{in}]\) incx: [rocblas_int] specifies the increment for the elements of each \(x_i\).
- \([\text{in}]\) stride_x: [rocblas_stride] stride from the start of one \(x_i\) matrix to the next \(x_i(i + 1)\).
- \([\text{in}]\) batch_count: [rocblas_int] number of instances in the batch.
rocblas_status rocblas_dbmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const double *A, rocblas_int lda, rocblas_stride stride_A, double *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_ctbmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stride_A, rocblas_float_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_ztbmv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation trans, rocblas_diagonal diag, rocblas_int m, rocblas_int k, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stride_A, rocblas_double_complex *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

rocblas_status rocblas_strsv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, float *x, rocblas_int incx)

BLAS Level 2 API.

trsv solves

A\*x = b \text{ or } A^{*}\text{trans}A\times x = b,

where x and b are vectors and A is a triangular matrix.

The vector x is overwritten on b.

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
- [in] transA: [rocblas_operation]
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: A is assumed to be unit triangular. rocblas_diagonal_non_unit: A is not assumed to be unit triangular.
- [in] m: [rocblas_int] m specifies the number of rows of b. m >= 0.
- [in] A: device pointer storing matrix A, of dimension ( lda, m )
- [in] lda: [rocblas_int] specifies the leading dimension of A. lda = max( 1, m ).
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
rocblas_status rocblas_dtrsv(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *A, rocblas_int lda, double *x, rocblas_int incx)

rocblas_status rocblas_strsv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *const A[], rocblas_int lda, float *const x[], rocblas_int incx, rocblas_int batch_count)

BLAS Level 2 API.

trsv_batched solves

\[ A_{i}x_{i} = b_{i} \text{ or } A_{i}^{T}x_{i} = b_{i}, \]

where \((A_{i}, x_{i}, b_{i})\) is the \(i\)-th instance of the batch. \(x_{i}\) and \(b_{i}\) are vectors and \(A_{i}\) is an \(m\) by \(m\) triangular matrix.

The vector \(x\) is overwritten on \(b\).

**Parameters**

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: \(A\) is an upper triangular matrix. rocblas_fill_lower: \(A\) is a lower triangular matrix.
- [in] transA: [rocblas_operation]
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: \(A\) is assumed to be unit triangular. rocblas_diagonal_non_unit: \(A\) is not assumed to be unit triangular.
- [in] m: [rocblas_int] \(m\) specifies the number of rows of \(b\). \(m \geq 0\).
- [in] A: device array of device pointers storing each matrix \(A_{i}\).
- [in] lda: [rocblas_int] specifies the leading dimension of each \(A_{i}\). \(lda = \text{max}(1, m)\)
- [in] x: device array of device pointers storing each vector \(x_{i}\).
- [in] incx: [rocblas_int] specifies the increment for the elements of \(x\).
- [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dtrsv_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const double *const A[], rocblas_int lda, double *const x[], rocblas_int incx, rocblas_int batch_count)

rocblas_status rocblas_strsv_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, const float *A, rocblas_int lda, rocblas_stride stride_A, float *x, rocblas_int incx, rocblas_stride stride_x, rocblas_int batch_count)

BLAS Level 2 API.

trsv_strided_batched solves

\[ A_{i}x_{i} = b_{i} \text{ or } A_{i}^{T}x_{i} = b_{i}, \]
where \((A_i, x_i, b_i)\) is the \(i\)-th instance of the batch. \(x_i\) and \(b_i\) are vectors and \(A_i\) is an \(m\) by \(m\) triangular matrix, for \(i = 1, \ldots, \text{batch\_count}\).

The vector \(x\) is overwritten on \(b\).

**Parameters**

- \([\text{in}]\) \(\text{handle}\): \([\text{rocblas\_handle}]\) handle to the rocblas library context queue.
- \([\text{in}]\) \(\text{uplo}\): \([\text{rocblas\_fill}]\) \(\text{rocblas\_fill\_upper}: A\) is an upper triangular matrix. \(\text{rocblas\_fill\_lower}: A\) is a lower triangular matrix.
- \([\text{in}]\) \(\text{transA}\): \([\text{rocblas\_operation}]\)
- \([\text{in}]\) \(\text{diag}\): \([\text{rocblas\_diagonal}]\) \(\text{rocblas\_diagonal\_unit}: A\) is assumed to be unit triangular. \(\text{rocblas\_diagonal\_non\_unit}: A\) is not assumed to be unit triangular.
- \([\text{in}]\) \(m\): \([\text{rocblas\_int}]\) \(m\) specifies the number of rows of each \(b_i\). \(m \geq 0\).
- \([\text{in}]\) \(A\): device pointer to the first matrix \((A_1)\) in the batch, of dimension \((\text{lda, } m)\)
- \([\text{in}]\) \(\text{stride\_A}\): \([\text{rocblas\_stride}]\) stride from the start of one \(A_i\) matrix to the next \(A_{(i + 1)}\)
- \([\text{in}]\) \(\text{lda}\): \([\text{rocblas\_int}]\) specifies the leading dimension of each \(A_i\). \(\text{lda} = \max(1, m)\).
- \([\text{inout}]\) \(x\): device pointer to the first vector \((x_1)\) in the batch.
- \([\text{in}]\) \(\text{stride\_x}\): \([\text{rocblas\_stride}]\) stride from the start of one \(x_i\) vector to the next \(x_{(i + 1)}\)
- \([\text{in}]\) \(\text{incx}\): \([\text{rocblas\_int}]\) specifies the increment for the elements of each \(x_i\).
- \([\text{in}]\) \(\text{batch\_count}\): \([\text{rocblas\_int}]\) number of instances in the batch

\[
\text{rocblas\_status rocblas\_dtrsv\_strided\_batched}(\text{rocblas\_handle handle, rocblas\_fill uplo, rocblas\_operation transA, rocblas\_diagonal diag, rocblas\_int m, const double *A, rocblas\_int lda, rocblas\_stride stride\_A, double *x, rocblas\_int incx, rocblas\_stride stride\_x, rocblas\_int batch\_count)
\]

\[
\text{rocblas\_status rocblas\_sger}(\text{rocblas\_handle handle, rocblas\_int m, rocblas\_int n, const float *alpha, const float *x, rocblas\_int incx, const float *y, rocblas\_int incy, float *A, rocblas\_int lda)}
\]

**BLAS Level 2 API.**

\(\text{xGER}\) performs the matrix-vector operations

\[
A := A + \alpha x y^{\ast} T
\]

where \(\alpha\) is a scalar, \(x\) and \(y\) are vectors, and \(A\) is an \(m\) by \(n\) matrix.

**Parameters**

- \([\text{in}]\) \(\text{handle}\): \([\text{rocblas\_handle}]\) handle to the rocblas library context queue.
- \([\text{in}]\) \(m\): \([\text{rocblas\_int}]\) the number of rows of the matrix \(A\).
- \([\text{in}]\) \(n\): \([\text{rocblas\_int}]\) the number of columns of the matrix \(A\).
- \([\text{in}]\) \(\alpha\): device pointer or host pointer to scalar \(\alpha\).
- \([\text{in}]\) \(x\): device pointer storing vector \(x\).
- \([\text{in}]\) \(\text{incx}\): \([\text{rocblas\_int}]\) specifies the increment for the elements of \(x\).
- \([\text{in}]\) \(y\): device pointer storing vector \(y\).
• [in] incy: [rocblas_int] specifies the increment for the elements of y.
• [inout] A: device pointer storing matrix A.
• [in] lda: [rocblas_int] specifies the leading dimension of A.

rocblas_status rocblas_dger (rocblas_handle handle, rocblas_int m, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, const double *y, rocblas_int incy, double *A, rocblas_int lda)

rocblas_status rocblas_sger_batched (rocblas_handle handle, rocblas_int m, rocblas_int n, const float *alpha, const float *x[], rocblas_int incx, const float *const y[], rocblas_int incy, float *const A[], rocblas_int lda, rocblas_int batch_count)

BLAS Level 2 API.

xGER_BATCHED performs a batch of the matrix-vector operations

\[ A_i := A_i + \alpha x_i y_i^T \]

where \((A_i, x_i, y_i)\) is the \(i\)-th instance of the batch. \(\alpha\) is a scalar, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] m: [rocblas_int] the number of rows of each matrix \(A_i\).
• [in] n: [rocblas_int] the number of columns of each matrix \(A_i\).
• [in] alpha: device pointer or host pointer to scalar \(\alpha\).
• [in] x: device array of device pointers storing each vector \(x_i\).
• [in] incx: [rocblas_int] specifies the increment for the elements of each vector \(x_i\).
• [in] y: device array of device pointers storing each vector \(y_i\).
• [in] incy: [rocblas_int] specifies the increment for the elements of each vector \(y_i\).
• [inout] A: device array of device pointers storing each matrix \(A_i\).
• [in] lda: [rocblas_int] specifies the leading dimension of each \(A_i\).
• [in] batch_count: [rocblas_int] number of instances in the batch

rocblas_status rocblas_dger_batched (rocblas_handle handle, rocblas_int m, rocblas_int n, const double *alpha, const double *const x[], rocblas_int incx, const double *const y[], rocblas_int incy, double *const A[], rocblas_int lda, rocblas_int batch_count)

rocblas_status rocblas_sger_strided_batched (rocblas_handle handle, rocblas_int m, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, rocblas_stride stridex, const float *y, rocblas_int incy, rocblas_stride stridey, float *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)

BLAS Level 2 API.

xGER_STRIDED_BATCHED performs the matrix-vector operations

\[ A_i := A_i + \alpha x_i y_i^T \]
\[ A_i := A_i + \alpha x_i y_i^T \]

where \((A_i, x_i, y_i)\) is the i-th instance of the batch. \(\alpha\) is a scalar, \(x_i\) and \(y_i\) are vectors and \(A_i\) is an \(m\) by \(n\) matrix, for \(i = 1, \ldots, \text{batch\_count}\).

### Parameters

- [in] `handle`: [rocblas_handle] handle to the rocblas library context queue.
- [in] `m`: [rocblas_int] the number of rows of each matrix \(A_i\).
- [in] `n`: [rocblas_int] the number of columns of each matrix \(A_i\).
- [in] `alpha`: device pointer or host pointer to scalar \(\alpha\).
- [in] `x`: device pointer to the first vector \((x_1)\) in the batch.
- [in] `incx`: [rocblas_int] specifies the increments for the elements of each vector \(x_i\).
- [in] `stridex`: [rocblas_stride] stride from the start of one vector \((x_i)\) and the next one \((x_{i+1})\). There are no restrictions placed on stride \(x\), however the user should take care to ensure that stride \(x\) is of appropriate size, for a typical case this means stride \(x\) \(\geq m \times \text{incx}\).
- [inout] `y`: device pointer to the first vector \((y_0)\) in the batch.
- [in] `incy`: [rocblas_int] specifies the increment for the elements of each vector \(y_i\).
- [in] `stridex`: [rocblas_stride] stride from the start of one vector \((y_i)\) and the next one \((y_{i+1})\). There are no restrictions placed on stride \(y\), however the user should take care to ensure that stride \(y\) is of appropriate size, for a typical case this means stride \(y\) \(\geq n \times \text{incy}\).
- [inout] `A`: device pointer to the first matrix \((A_1)\) in the batch.
- [in] `lda`: [rocblas_int] specifies the leading dimension of each \(A_i\).
- [in] `strideA`: [rocblas_stride] stride from the start of one matrix \((A_i)\) and the next one \((A_{i+1})\)
- [in] `batch_count`: [rocblas_int] number of instances in the batch

```c
rocblas_status rocblas_dger_strided_batched(rocblas_handle handle, rocblas_int m, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, rocblas_stride stridex, const double *y, rocblas_int incy, rocblas_stride stridey, double *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)
```

### BLAS Level 2 API

xSYR performs the matrix-vector operations

\[ A := A + \alpha x x^T \]

where \(\alpha\) is a scalar, \(x\) is a vector, and \(A\) is an \(n\) by \(n\) symmetric matrix.

### Parameters

- [in] `handle`: [rocblas_handle] handle to the rocblas library context queue.
rocblas_status rocblas_dsyr (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, double *A, rocblas_int lda)

rocblas_status rocblas_ssyr_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const float *alpha, const float *x[], rocblas_int incx, float *const A[], rocblas_int lda, rocblas_int batch_count)

BLAS Level 2 API.

xSYR_batched performs a batch of matrix-vector operations

\[ A[i] := A[i] + \alpha x[i] x[i]^T \]

where alpha is a scalar, x is an array of vectors, and A is an array of n by n symmetric matrices, for i = 1, \ldots, \text{batch\_count}

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] uplo: [rocblas_fill] specifies whether the upper ‘rocblas\_fill\_upper’ or lower ‘rocblas\_fill\_lower’ if rocblas\_fill\_upper, the lower part of A is not referenced if rocblas\_fill\_lower, the upper part of A is not referenced
- [in] n: [rocblas_int] the number of rows and columns of matrix A.
- [in] alpha: device pointer or host pointer to scalar alpha.
- [in] x: device pointer storing vector x.
- [in] incx: [rocblas_int] specifies the increment for the elements of x.
- [inout] A: device pointer storing matrix A.
- [in] lda: [rocblas_int] specifies the leading dimension of A.
- [in] batch_count: [rocblas_int] number of instances in the batch
**rocblas_status** `rocblas_ssyr_strided_batched` *(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const float *alpha, const float *x, rocblas_int incx, rocblas_stride stridex, float *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)*

BLAS Level 2 API.

xSYR_strided_batched performs the matrix-vector operations

\[ A[i] := A[i] + \alpha x[i] x[i]^T \]

where \( \alpha \) is a scalar, vectors, and \( A \) is an array of \( n \) by \( n \) symmetric matrices, for \( i = 1, \ldots, \text{batch\_count} \).

**Parameters**

- **[in]** `handle`: [rocblas_handle] handle to the rocblas library context queue.
- **[in]** `uplo`: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’ if rocblas_fill_upper, the lower part of \( A \) is not referenced if rocblas_fill_lower, the upper part of \( A \) is not referenced.
- **[in]** `n`: [rocblas_int] the number of rows and columns of each matrix \( A \).
- **[in]** `alpha`: device pointer or host pointer to scalar \( \alpha \).
- **[in]** `x`: device pointer to the first vector \( x_1 \).
- **[in]** `incx`: [rocblas_int] specifies the increment for the elements of each \( x_i \).
- **[in]** `stridex`: [rocblas_stride] specifies the pointer increment between vectors \( (x_i) \) and \( (x_i+1) \).
- **[inout]** `A`: device pointer to the first matrix \( A_1 \).
- **[in]** `lda`: [rocblas_int] specifies the leading dimension of each \( A_i \).
- **[in]** `strideA`: [rocblas_stride] stride from the start of one matrix \( (A_i) \) and the next one \( (A_i+1) \).
- **[in]** `batch_count`: [rocblas_int] number of instances in the batch.

**rocblas_status** `rocblas_dsyr_strided_batched` *(rocblas_handle handle, rocblas_fill uplo, rocblas_int n, const double *alpha, const double *x, rocblas_int incx, rocblas_stride stridex, double *A, rocblas_int lda, rocblas_stride strideA, rocblas_int batch_count)*

**rocblas_status** `rocblas_strmm` *(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, float*B, rocblas_int ldb)*

BLAS Level 3 API.

trmm performs one of the matrix-matrix operations

\[ B := \alpha \text{op}(A) B \] or \[ B := \alpha B \text{op}(A) \]

where \( \alpha \) is a scalar, \( B \) is an \( m \) by \( n \) matrix, \( A \) is a unit, or non-unit, upper or lower triangular matrix and \( \text{op}(A) \) is one of
\[
\begin{align*}
\text{op}( A ) &= A \quad \text{or} \quad \text{op}( A ) = A^T \quad \text{or} \quad \text{op}( A ) = A^H.
\end{align*}
\]

**Parameters**

- **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.
- **[in] side**: [rocblas_side] rocblas_side_left: \(C := \alpha \text{op}( A )B\). rocblas_side_right: \(C := \alpha B \text{op}( A )\).
- **[in] uplo**: [rocblas_fill] rocblas_fill_upper: \(A\) is an upper triangular matrix. rocblas_fill_lower: \(A\) is a lower triangular matrix.
- **[in] transA**: [rocblas_operation] transB: \(\text{op}(A) = A\). rocblas_operation_transpose: \(\text{op}(A) = A^T\). rocblas_operation_conjugate_transpose: \(\text{op}(A) = A^H\).
- **[in] diag**: [rocblas_diagonal] rocblas_diagonal_unit: \(A\) is assumed to be unit triangular. rocblas_diagonal_non_unit: \(A\) is not assumed to be unit triangular.
- **[in] m**: [rocblas_int] \(m\) specifies the number of rows of \(B\). \(m \geq 0\).
- **[in] n**: [rocblas_int] \(n\) specifies the number of columns of \(B\). \(n \geq 0\).
- **[in] alpha**: \(\alpha\) specifies the scalar \(\alpha\). When \(\alpha\) is zero then \(A\) is not referenced and \(B\) need not be set before entry.
- **[in] A**: pointer storing matrix \(A\) on the GPU. of dimension ( \(lda, k\), where \(k = m\) when rocblas_side_left and is \(n\) when rocblas_side_right only the upper/lower triangular part is accessed.
- **[in] lda**: [rocblas_int] \(lda\) specifies the first dimension of \(A\). if side = rocblas_side_left, \(lda \geq \max(1, m)\), if side = rocblas_side_right, \(lda \geq \max(1, n)\).
- **[in] B**: pointer storing matrix \(B\) on the GPU.
- **[in] ldb**: [rocblas_int] \(ldb\) specifies the first dimension of \(B\). \(ldb \geq \max(1, m)\).

**rocblas_dtrmm**

\[
\text{rocblas_status rocblas_dtrmm}(\text{rocblas_handle handle}, \text{rocblas_side side}, \text{rocblas_fill uplo}, \text{rocblas_operation transA}, \text{rocblas_diagonal diag}, \text{rocblas_int m}, \text{rocblas_int n}, \text{const double *alpha}, \text{const double *A}, \text{rocblas_int lda}, \text{double *B}, \text{rocblas_int ldb})
\]

**rocblas_strtri**

\[
\text{rocblas_status rocblas_strtri}(\text{rocblas_handle handle}, \text{rocblas_fill uplo}, \text{rocblas_diagonal diag}, \text{rocblas_int n}, \text{const float *A}, \text{rocblas_int lda}, \text{float *invA}, \text{rocblas_int ldinvA})
\]

BLAS Level 3 API.

trtri compute the inverse of a matrix \(A\), namely, \(invA\)

and write the result into \(invA\);
• [in] A: device pointer storing matrix A.
• [in] lda: [rocblas_int] specifies the leading dimension of A.
• [out] invA: device pointer storing matrix invA.
• [in] ldinvA: [rocblas_int] specifies the leading dimension of invA.

rocblas_status rocblas_dtrtri (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const double *A, rocblas_int lda, double *invA, rocblas_int ldinvA)

rocblas_status rocblas_strtri_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const float *const A[], rocblas_int lda, float *invA[], rocblas_int ldinvA, rocblas_int batch_count)

BLAS Level 3 API.

trtri_batched compute the inverse of A_i and write into invA_i where A_i and invA_i are the i-th matrices in the batch, for i = 1, . . . , batch_count.

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] uplo: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’
• [in] diag: [rocblas_diagonal] = ‘rocblas_diagonal_non_unit’, A is non-unit triangular; = ‘rocblas_diagonal_unit’, A is unit triangular;
• [in] n: [rocblas_int]
• [in] A: device array of device pointers storing each matrix A_i.
• [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
• [out] invA: device array of device pointers storing the inverse of each matrix A_i. Partial in-place operation is supported, see below. If UPLO = ‘U’, the leading N-by-N upper triangular part of the invA will store the inverse of the upper triangular matrix, and the strictly lower triangular part of invA is cleared. If UPLO = ‘L’, the leading N-by-N lower triangular part of the invA will store the inverse of the lower triangular matrix, and the strictly upper triangular part of invA is cleared.
• [in] ldinvA: [rocblas_int] specifies the leading dimension of each invA_i.
• [in] batch_count: [rocblas_int] numbers of matrices in the batch

rocblas_status rocblas_dtrtri_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const double *const A[], rocblas_int lda, double *invA[], rocblas_int ldinvA, rocblas_int batch_count)

rocblas_status rocblas_strtri_strided_batched (rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const float *const A[], rocblas_int lda, float *invA[], rocblas_int ldinvA, rocblas_int stride_a, rocblas_int ldstride_invA, rocblas_int batch_count)

BLAS Level 3 API.

trtri_strided_batched compute the inverse of A_i and write into invA_i where A_i and invA_i are the i-th matrices in the batch, for i = 1, . . . , batch_count.
Parameters

- **[in] handle**: [rocblas_handle] handle to the rocblas library context queue.
- **[in] uplo**: [rocblas_fill] specifies whether the upper ‘rocblas_fill_upper’ or lower ‘rocblas_fill_lower’
- **[in] diag**: [rocblas_diagonal] = ‘rocblas_diagonal_non_unit’, A is non-unit triangular; = ‘rocblas_diagonal_unit’, A is unit triangular;
- **[in] n**: [rocblas_int]
- **[in] A**: device pointer pointing to address of first matrix A_i.
- **[in] lda**: [rocblas_int] specifies the leading dimension of each A.
- **[in] stride_a**: [rocblas_stride] “batch stride a”: stride from the start of one A_i matrix to the next A_(i + 1).
- **[out] invA**: device pointer storing the inverses of each matrix A_i. Partial inplace operation is supported, see below. If UPLO = ‘U’, the leading N-by-N upper triangular part of the invA will store the inverse of the upper triangular matrix, and the strictly lower triangular part of invA is cleared. If UPLO = ‘L’, the leading N-by-N lower triangular part of the invA will store the inverse of the lower triangular matrix, and the strictly upper triangular part of invA is cleared.
- **[in] ldinvA**: [rocblas_int] specifies the leading dimension of each invA_i.
- **[in] stride_invA**: [rocblas_stride] “batch stride invA”: stride from the start of one invA_i matrix to the next invA_(i + 1).
- **[in] batch_count**: [rocblas_int] numbers of matrices in the batch

rocblas_status rocblas_dtrtri_strided_batched(rocblas_handle handle, rocblas_fill uplo, rocblas_diagonal diag, rocblas_int n, const double *A, rocblas_int lda, rocblas_stride stride_a, double *invA, rocblas_int ldinvA, rocblas_stride stride_invA, rocblas_int batch_count)

rocblas_status rocblas_strsm(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, float *B, rocblas_int ldb)

BLAS Level 3 API.

trsm solves

\[ \text{op}(A) \cdot X = \alpha B \quad \text{or} \quad X \cdot \text{op}(A) = \alpha B, \]

where \( \alpha \) is a scalar, \( X \) and \( B \) are \( m \) by \( n \) matrices, \( A \) is triangular matrix and \( \text{op}(A) \) is one of

\[ \text{op}(A) = A \quad \text{or} \quad \text{op}(A) = A^T \quad \text{or} \quad \text{op}(A) = A^H. \]

The matrix \( X \) is overwritten on \( B \).

Note about memory allocation: When trsm is launched with \( k \) evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF_TRSM_B_CHNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF_TRSM_B_CHNK be set to the desired chunk of right hand sides to be used at a time.
(where \( k \) is \( m \) when \( \text{rocblas侧_left} \) and is \( n \) when \( \text{rocblas侧_right} \))

**Parameters**

- **[in]** `handle` [rocblas_handle] handle to the rocblas library context queue.
- **[in]** `side` [rocblas_side] `rocblas侧_left`: \( \text{op}(A)X = \alpha B \). `rocblas侧_right`: \( X\text{op}(A) = \alpha B \).
- **[in]** `uplo` [rocblas_fill] `rocblas_fill_upper`: \( A \) is an upper triangular matrix. `rocblas_fill_lower`: \( A \) is a lower triangular matrix.
- **[in]** `diag` [rocblas_diagonal] `rocblas_diagonal_unit`: \( A \) is assumed to be unit triangular. `rocblas_diagonal_non_unit`: \( A \) is not assumed to be unit triangular.
- **[in]** `m` [rocblas_int] \( m \) specifies the number of rows of \( B \). \( m \geq 0 \).
- **[in]** `n` [rocblas_int] \( n \) specifies the number of columns of \( B \). \( n \geq 0 \).
- **[in]** `alpha`: device pointer or host pointer specifying the scalar \( \alpha \). When \( \alpha \) is \&zero then \( A \) is not referenced and \( B \) need not be set before entry.
- **[in]** `A`: device pointer storing matrix \( A \). of dimension ( \( \text{lda} \), \( k \) ), where \( k \) is \( m \) when \( \text{rocblas侧_left} \) and is \( n \) when \( \text{rocblas侧_right} \) only the upper/lower triangular part is accessed.
- **[in]** `lda` [rocblas_int] \( \text{lda} \) specifies the first dimension of \( A \). if \( \text{side} = \text{rocblas侧_left} \), \( \text{lda} \geq \max(1, m) \), if \( \text{side} = \text{rocblas侧_right} \), \( \text{lda} \geq \max(1, n) \).
- **[inout]** `B`: device pointer storing matrix \( B \).
- **[in]** `ldb` [rocblas_int] \( \text{ldb} \) specifies the first dimension of \( B \). \( \text{ldb} \geq \max(1, m) \).

\[
\text{rocblas_dtrsm}( \text{handle}, \text{side}, \text{uplo}, \text{transA}, \text{diag}, m, n, \alpha, A, \text{lda}, B, \text{ldb})
\]

\[
\text{rocblas_strsm_batched}( \text{handle}, \text{side}, \text{uplo}, \text{transA}, \text{diag}, m, n, \alpha, A, \text{lda}, B, \text{ldb}, \text{batch_count})
\]

**BLAS Level 3 API.**

`trsm_batched` performs the following batched operation:

\[
\text{op}(A_i)X_i = \alpha B_i \text{ or } X_i\text{op}(A_i) = \alpha B_i, \text{ for } i = 1, \ldots, \text{batch}_\rightarrow\text{count}.
\]

where \( \alpha \) is a scalar, \( X \) and \( B \) are batched \( m \) by \( n \) matrices, \( A \) is triangular batched matrix and \( \text{op}(A) \) is one of

\[
\text{op}(A) = A \text{ or } \text{op}(A) = A^T \text{ or } \text{op}(A) = A^H.
\]

Each matrix \( X_i \) is overwritten on \( B_i \) for \( i = 1, \ldots, \text{batch}_\rightarrow\text{count} \).

Note about memory allocation: When `trsm` is launched with a \( k \) evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated
memory found in the handle to increase overall performance. This memory can be managed by using the
environment variable WORKBUF_TRSM_B_CHNK. When this variable is not set the device memory
used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce
performance. Under these circumstances it is recommended that WORKBUF_TRSM_B_CHNK be set to
the desired chunk of right hand sides to be used at a time. (where k is m when rocblas_side_left and is n
when rocblas_side_right)

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] side: [rocblas_side] rocblas_side_left: op(A)*X = alpha*B. rocblas_side_right:
  X*op(A) = alpha*B.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: each A_i is an upper triangular matrix.
  rocblas_fill_lower: each A_i is a lower triangular matrix.
- [in] transA: [rocblas_operation] transB: op(A) = A. rocblas_operation_transpose:
  op(A) = A^T. rocblas_operation_conjugate_transpose: op(A) = A^H.
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: each A_i is assumed to be unit triangu-
  lar. rocblas_diagonal_non_unit: each A_i is not assumed to be unit triangular.
- [in] m: [rocblas_int] m specifies the number of rows of each B_i. m >= 0.
- [in] n: [rocblas_int] n specifies the number of columns of each B_i. n >= 0.
- [in] alpha: device pointer or host pointer specifying the scalar alpha. When alpha is &zero
  then A is not referenced and B need not be set before entry.
- [in] A: device array of device pointers storing each matrix A_i on the GPU. Matricies are of
dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only
the upper/lower triangular part is accessed.
- [in] lda: [rocblas_int] lda specifies the first dimension of each A_i. if side =
  rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).
- [inout] B: device array of device pointers storing each matrix B_i on the GPU.
- [in] ldb: [rocblas_int] ldb specifies the first dimension of each B_i. ldb >= max( 1, m ).
- [in] batch_count: [rocblas_int] number of trsm operatons in the batch.

rocblas_status rocblas_dtrsm_batched (rocblas_handle handle, rocblas_side side, rocblas_fill
uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const
double *alpha, const double *const A[], rocblas_int lda, double *B[],
rocblas_int ldb, rocblas_int batch_count)

rocblas_status rocblas_strsm_strided_batched (rocblas_handle handle, rocblas_side side, rocblas_fill
uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const
float *alpha, const float *A, rocblas_int lda, rocblas_stride stride_a, float *B, rocblas_int
ldb, rocblas_stride stride_b, rocblas_int batch_count)

BLAS Level 3 API.

trsm_strided_batched performs the following strided batched operation:
\[
op(A_i) \times X_i = \alpha \times B_i \quad \text{or} \quad X_i \times \op(A_i) = \alpha \times B_i, \quad \text{for} \ i = 1, \ldots, \text{batch} \rightarrow \text{count}.
\]

where alpha is a scalar, X and B are strided batched m by n matrices, A is triangular strided batched matrix and \(\op(A)\) is one of

\[
\op(A) = A \quad \text{or} \quad \op(A) = A^T \quad \text{or} \quad \op(A) = A^H.
\]

Each matrix \(X_i\) is overwritten on \(B_i\) for \(i = 1, \ldots, \text{batch} \rightarrow \text{count}\).

Note about memory allocation: When trsm is launched with a \(k\) evenly divisible by the internal block size of 128, and is no larger than 10 of these blocks, the API takes advantage of utilizing pre-allocated memory found in the handle to increase overall performance. This memory can be managed by using the environment variable WORKBUF_TRSM_B_CHNK. When this variable is not set the device memory used for temporary storage will default to 1 MB and may result in chunking, which in turn may reduce performance. Under these circumstances it is recommended that WORKBUF_TRSM_B_CHNK be set to the desired chunk of right hand sides to be used at a time. (where \(k\) is m when rocblas_side_left and is n when rocblas_side_right)

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] side: [rocblas_side] rocblas_side_left: \(\op(A) \times X = \alpha \times B\). rocblas_side_right: \(X \times \op(A) = \alpha \times B\).
- [in] uplo: [rocblas_fill] rocblas_fill_upper: each \(A_i\) is an upper triangular matrix. rocblas_fill_lower: each \(A_i\) is a lower triangular matrix.
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: each \(A_i\) is assumed to be unit triangular. rocblas_diagonal_non_unit: each \(A_i\) is not assumed to be unit triangular.
- [in] m: [rocblas_int] \(m\) specifies the number of rows of each \(B_i\). \(m \geq 0\).
- [in] n: [rocblas_int] \(n\) specifies the number of columns of each \(B_i\). \(n \geq 0\).
- [in] alpha: device pointer or host pointer specifying the scalar alpha. When alpha is 
&zero then A is not referenced and B need not be set before entry.
- [in] A: device pointer pointing to the first matrix \(A_1\) of dimension (lda, k), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.
- [in] lda: [rocblas_int] \(lda\) specifies the first dimension of each \(A_i\). if side = rocblas_side_left, \(lda \geq \max(1, m)\), if side = rocblas_side_right, \(lda \geq \max(1, n)\).
- [in] stride_a: [rocblas_stride] stride from the start of one \(A_i\) matrix to the next \(A_{i+1}\).
- [inout] B: device pointer pointing to the first matrix \(B_1\).
- [in] ldb: [rocblas_int] \(ldb\) specifies the first dimension of each \(B_i\). \(ldb \geq \max(1, m)\).
- [in] stride_b: [rocblas_stride] stride from the start of one \(B_i\) matrix to the next \(B_{i+1}\).
- [in] batch_count: [rocblas_int] number of trsm operatons in the batch.
**rocblas_status rocblas_dtrsm_strided_batched**

```c
rocblas_status rocblas_dtrsm_strided_batched(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, rocblas_stride stride_a, double *B, rocblas_int ldb, rocblas_stride stride_b, rocblas_int batch_count)
```

**rocblas_status rocblas_sgemm**

```c
rocblas_status rocblas_sgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A, rocblas_int lda, const float *B, rocblas_int ldb, const float *beta, float *C, rocblas_int ldc)
```

BLAS Level 3 API.

xGEMM performs one of the matrix-matrix operations

\[ C = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C, \]

where \( \text{op}(X) \) is one of

\[
\begin{align*}
\text{op}(X) &= X & \text{or} \\
\text{op}(X) &= X^{\top} & \text{or} \\
\text{op}(X) &= X^{\ast H}
\end{align*}
\]

alpha and beta are scalars, and A, B and C are matrices, with \( \text{op}(A) \) an \( m \) by \( k \) matrix, \( \text{op}(B) \) a \( k \) by \( n \) matrix and \( C \) an \( m \) by \( n \) matrix.

**Parameters**

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of \( \text{op}(A) \)
- [in] transB: [rocblas_operation] specifies the form of \( \text{op}(B) \)
- [in] m: [rocblas_int] number or rows of matrices \( \text{op}(A) \) and \( C \)
- [in] n: [rocblas_int] number of columns of matrices \( \text{op}(A) \) and \( C \)
- [in] k: [rocblas_int] number of columns of matrix \( \text{op}(A) \) and number of rows of matrix \( \text{op}(B) \)
- [in] alpha: device pointer or host pointer specifying the scalar \( \alpha \)
- [in] A: device pointer storing matrix \( A \)
- [in] lda: [rocblas_int] specifies the leading dimension of \( A \)
- [in] B: device pointer storing matrix \( B \)
- [in] ldb: [rocblas_int] specifies the leading dimension of \( B \)
- [in] beta: device pointer or host pointer specifying the scalar \( \beta \)
- [inout] C: device pointer storing matrix \( C \) on the GPU.
- [in] ldc: [rocblas_int] specifies the leading dimension of \( C \)

**rocblas_status rocblas_dgemm**

```c
rocblas_status rocblas_dgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *A, rocblas_int lda, const double *B, rocblas_int ldb, const double *beta, double *C, rocblas_int ldc)
```
rocblas_status rocblas_hgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *A, rocblas_int lda, const rocblas_half *B, rocblas_int ldb, const rocblas_half *beta, rocblas_half *C, rocblas_int ldc)

rocblas_status rocblas_cgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, const rocblas_float_complex *B, rocblas_int ldb, const rocblas_float_complex *beta, rocblas_float_complex *C, rocblas_int ldc)

rocblas_status rocblas_zgemm(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, const rocblas_double_complex *B, rocblas_int ldb, const rocblas_double_complex *beta, rocblas_double_complex *C, rocblas_int ldc)

rocblas_status rocblas_sgemm_batched(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A[], rocblas_int lda, const float *beta, float *C[], rocblas_int ldc, rocblas_int batch_count)

BLAS Level 3 API.

xGEMM_BATCHED performs one of the batched matrix-matrix operations C_{i} = \alpha \cdot op( A_{i} ) \cdot op( B_{i} ) + \beta \cdot C_{i}, \text{ for } i = 1, \ldots, \text{batch\_count}. \text{ where op( X ) is one of op( X ) = X or op( X ) = X^{*T} or op( X ) = X^{*H}, alpha and beta are scalars, and A, B and C are strided batched matrices, with op( A ) an m by k by batch\_count strided\_batched matrix, op( B ) an k by n by batch\_count strided\_batched matrix and C an m by n by batch\_count strided\_batched matrix.}

Parameters

- [in] handle: [rocblas_handle handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of op( A )
- [in] transB: [rocblas_operation] specifies the form of op( B )
- [in] m: [rocblas_int] matrix dimension m.
- [in] k: [rocblas_int] matrix dimension k.
- [in] alpha: device pointer or host pointer specifying the scalar alpha.
- [in] A: device array of device pointers storing each matrix A_{i}.
- [in] lda: [rocblas_int] specifies the leading dimension of each A_{i}.
- [in] B: device array of device pointers storing each matrix B_{i}.
- [in] ldb: [rocblas_int] specifies the leading dimension of each B_{i}.
- [in] beta: device pointer or host pointer specifying the scalar beta.
- [inout] C: device array of device pointers storing each matrix C_{i}.
- [in] ldc: [rocblas_int] specifies the leading dimension of each C_{i}.
- [in] batch_count: [rocblas_int] number of gemm operations in the batch
rocblas_status rocblas_dgemm_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *const A[], rocblas_int lda, const double *const B[], rocblas_int ldb, const double *beta, double *const C[], rocblas_int ldc, rocblas_int batch_count)

rocblas_status rocblas_hgemm_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *const A[], rocblas_int lda, const rocblas_half *const B[], rocblas_int ldb, const rocblas_half *beta, rocblas_half *const C[], rocblas_int ldc, rocblas_int batch_count)

rocblas_status rocblas_cgemm_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_float_complex *alpha, const rocblas_float_complex *const A[], rocblas_int lda, const rocblas_float_complex *const B[], rocblas_int ldb, const rocblas_float_complex *beta, rocblas_float_complex *const C[], rocblas_int ldc, rocblas_int batch_count)

rocblas_status rocblas_zgemm_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_double_complex *alpha, const rocblas_double_complex *const A[], rocblas_int lda, const rocblas_double_complex *const B[], rocblas_int ldb, const rocblas_double_complex *beta, rocblas_double_complex *const C[], rocblas_int ldc, rocblas_int batch_count)

rocblas_status rocblas_sgemm_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A, rocblas_int lda, rocblas_stride stride_a, const float *B, rocblas_int ldb, rocblas_stride stride_b, const float *beta, float *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

BLAS Level 3 API.

xGEMM_STRIDED_BATCHED performs one of the strided batched matrix-matrix operations

\[ C_i = \alpha \cdot \text{op}(A_i) \cdot \text{op}(B_i) + \beta \cdot C_i, \text{ for } i = 1, \ldots, \text{batch_count}. \]

where \( \text{op}(X) \) is one of

\[
\begin{align*}
\text{op}(X) &= X \\
\text{op}(X) &= X^\text{H} \quad \text{or} \\
\text{op}(X) &= X^\text{T} \quad \text{or}
\end{align*}
\]

alpha and beta are scalars, and A, B and C are strided batched matrices, with \( \text{op}(A) \) an m by k by batch_count strided_batched matrix, \( \text{op}(B) \) an k by n by batch_count strided_batched matrix and C an m by n by batch_count strided_batched matrix.

2.12. ROCm Libraries 219
Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transA: [rocblas_operation] specifies the form of op( A )
- [in] transB: [rocblas_operation] specifies the form of op( B )
- [in] m: [rocblas_int] matrix dimension m.
- [in] k: [rocblas_int] matrix dimension k.
- [in] alpha: device pointer or host pointer specifying the scalar alpha.
- [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
- [in] stride_a: [rocblas_stride] stride from the start of one A_i matrix to the next A_(i + 1).
- [in] B: device pointer pointing to the first matrix B_1.
- [in] ldb: [rocblas_int] specifies the leading dimension of each B_i.
- [in] stride_b: [rocblas_stride] stride from the start of one B_i matrix to the next B_(i + 1).
- [in] beta: device pointer or host pointer specifying the scalar beta.
- [inout] C: device pointer pointing to the first matrix C_1.
- [in] ldc: [rocblas_int] specifies the leading dimension of each C_i.
- [in] stride_c: [rocblas_stride] stride from the start of one C_i matrix to the next C_(i + 1).
- [in] batch_count: [rocblas_int] number of gemm operations in the batch

rocblas_status rocblas_dgemm_strided_batched
(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *A, rocblas_int lda, rocblas_stride stride_a, const double *B, rocblas_int ldb, rocblas_stride stride_b, const double *beta, double *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_hgemm_strided_batched
(rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_half *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_half *beta, rocblas_half *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)
rocblas_status rocblas_hgemm_kernel_name (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_half *alpha, const rocblas_half *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_half *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_half *beta, rocblas_half *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_sgemm_kernel_name (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const float *alpha, const float *A, rocblas_int lda, rocblas_stride stride_a, const float *B, rocblas_int ldb, rocblas_stride stride_b, const float *beta, float *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_dgemm_kernel_name (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const double *alpha, const double *A, rocblas_int lda, rocblas_stride stride_a, const double *B, rocblas_int ldb, rocblas_stride stride_b, const double *beta, double *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_cgemm_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_float_complex *alpha, const rocblas_float_complex *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_float_complex *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_float_complex *beta, rocblas_float_complex *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_zgemm_strided_batched (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const rocblas_double_complex *alpha, const rocblas_double_complex *A, rocblas_int lda, rocblas_stride stride_a, const rocblas_double_complex *B, rocblas_int ldb, rocblas_stride stride_b, const rocblas_double_complex *beta, rocblas_double_complex *C, rocblas_int ldc, rocblas_stride stride_c, rocblas_int batch_count)

rocblas_status rocblas_sgeam (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, const float *alpha, const float *A, rocblas_int lda, const float *beta, const float *B, rocblas_int ldb, float *C, rocblas_int ldc)
BLAS Level 3 API.

xGEAM performs one of the matrix-matrix operations
\[ C = \alpha \cdot \text{op}( A ) + \beta \cdot \text{op}( B ), \]
where \( \text{op}( X ) \) is one of
\[
\begin{align*}
\text{op}( X ) &= X \\
\text{op}( X ) &= X^T \\
\text{op}( X ) &= X^H,
\end{align*}
\]
alpha and beta are scalars, and A, B and C are matrices, with \( \text{op}( A ) \) an m by n matrix, \( \text{op}( B ) \) an m by n matrix, and C an m by n matrix.

Parameters
- \([\text{in}] \) handle: [rocblas_handle] handle to the rocblas library context queue.
- \([\text{in}] \) transA: [rocblas_operation] specifies the form of \( \text{op}( A ) \)
- \([\text{in}] \) transB: [rocblas_operation] specifies the form of \( \text{op}( B ) \)
- \([\text{in}] \) m: [rocblas_int] matrix dimension m.
- \([\text{in}] \) n: [rocblas_int] matrix dimension n.
- \([\text{in}] \) alpha: device pointer or host pointer specifying the scalar alpha.
- \([\text{in}] \) A: device pointer storing matrix A.
- \([\text{in}] \) lda: [rocblas_int] specifies the leading dimension of A.
- \([\text{in}] \) beta: device pointer or host pointer specifying the scalar beta.
- \([\text{in}] \) B: device pointer storing matrix B.
- \([\text{in}] \) ldb: [rocblas_int] specifies the leading dimension of B.
- \([\text{inout}] \) C: device pointer storing matrix C.
- \([\text{in}] \) ldc: [rocblas_int] specifies the leading dimension of C.

rocblas_status rocblas_dgeam (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, const double *alpha, const double *A, rocblas_int lda, const double *beta, const double *B, rocblas_int ldb, double *C, rocblas_int ldc)

rocblas_status rocblas_gemm_ex (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const void *alpha, const void *a, rocblas_datatype a_type, rocblas_int lda, const void *b, rocblas_datatype b_type, rocblas_int ldb, const void *beta, const void *c, rocblas_datatype c_type, rocblas_int ldc, void *d, rocblas_datatype d_type, rocblas_int ldd, rocblas_datatype compute_type, rocblas_gemm_algo algo, int32_t solution_index, uint32_t flags)

BLAS EX API.

GEMM_EX performs one of the matrix-matrix operations
\[ D = \alpha \cdot \text{op}( A ) \cdot \text{op}( B ) + \beta \cdot C, \]
where op( X ) is one of

\[
\begin{align*}
\text{op( X )} & = X & \text{or} \\
\text{op( X )} & = X^{\ast_T} & \text{or} \\
\text{op( X )} & = X^{\ast_H},
\end{align*}
\]

alpha and beta are scalars, and A, B, C, and D are matrices, with op( A ) an m by k matrix, op( B ) a k by n matrix and C and D are m by n matrices.

Supported types are as follows:

- rocblas_datatype_f64_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f32_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f16_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_bf16_r = a_type = b_type = c_type = d_type; rocblas_datatype_f32_r = compute_type
- rocblas_datatype_i8_r = a_type = b_type; rocblas_datatype_i32_r = c_type = d_type = compute_type
- rocblas_datatype_f32_c = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f64_c = a_type = b_type = c_type = d_type = compute_type

Below are restrictions for rocblas_datatype_i8_r = a_type = b_type; rocblas_datatype_i32_r = c_type = d_type = compute_type:

- k must be a multiple of 4
- lda must be a multiple of 4 if transA == rocblas_operation_transpose
- ldb must be a multiple of 4 if transB == rocblas_operation_none
- for transA == rocblas_operation_transpose or transB == rocblas_operation_none the matrices A and B must have each 4 consecutive values in the k dimension packed. This packing can be achieved with the following pseudo-code. The code assumes the original matrices are in A and B, and the packed matrices are A_packed and B_packed. The size of the A_packed matrix is the same as the size of the A matrix, and the size of the B_packed matrix is the same as the size of the B matrix.

```c
if(transA == rocblas_operation_none)
{
    int nb = 4;
    for(int i_m = 0; i_m < m; i_m++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            A_packed[i_k % nb + (i_m + (i_k / nb) * lda) * nb] = A[i_m + i_k * lda];
        }
    }
}
else
{
    A_packed = A;
}
if(transB == rocblas_operation_transpose)
{
(continues on next page)
```
```c
int nb = 4;
for(int i_n = 0; i_n < m; i_n++)
{
    for(int i_k = 0; i_k < k; i_k++)
    {
        B_packed[i_k % nb + (i_n + (i_k / nb) * lda) * nb] = B[i_n + i_k * lda];
    }
}
else
{
    B_packed = B;
}
```

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] transB: [rocblas_operation] specifies the form of op(B).
- [in] m: [rocblas_int] matrix dimension m.
- [in] k: [rocblas_int] matrix dimension k.
- [in] alpha: [const void *] device pointer or host pointer specifying the scalar alpha. Same datatype as compute_type.
- [in] a: [void *] device pointer storing matrix A.
- [in] a_type: [rocblas_datatype] specifies the datatype of matrix A.
- [in] lda: [rocblas_int] specifies the leading dimension of A.
- [in] b: [void *] device pointer storing matrix B.
- [in] b_type: [rocblas_datatype] specifies the datatype of matrix B.
- [in] ldb: [rocblas_int] specifies the leading dimension of B.
- [in] beta: [const void *] device pointer or host pointer specifying the scalar beta. Same datatype as compute_type.
- [in] c: [void *] device pointer storing matrix C.
- [in] c_type: [rocblas_datatype] specifies the datatype of matrix C.
- [in] ldc: [rocblas_int] specifies the leading dimension of C.
- [out] d: [void *] device pointer storing matrix D.
- [in] d_type: [rocblas_datatype] specifies the datatype of matrix D.
- [in] ldd: [rocblas_int] specifies the leading dimension of D.
- [in] compute_type: [rocblas_datatype] specifies the datatype of computation.
- [in] solution_index: [int32_t] reserved for future use.
\[ \text{flags} = [\text{uint32}_t] \text{ reserved for future use.} \]

```c
rocblas_status rocblas_gemm_batched_ex (rocblas_handle handle, rocblas_operation transA,
rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const void *alpha, const void *
a, rocblas_datatype a_type, rocblas_int lda, const void *b, rocblas_datatype b_type, rocblas_int ldb,
const void *beta, const void *c, rocblas_datatype c_type, rocblas_int ldc, void *d, rocblas_datatype
d_type, rocblas_int ldd, rocblas_int batch_count,
rocblas_datatype compute_type, rocblas_gemm_algo algo, int32_t solution_index, uint32_t flags)
```

**BLAS EX API.**

\textit{GEMM\_BATCHED\_EX} performs one of the batched matrix-matrix operations \( D_{i} = \alpha \cdot \text{op}(A_{i}) \cdot \text{op}(B_{i}) + \beta \cdot C_{i} \), for \( i = 1, \ldots, \text{batch\_count} \). where \( \text{op}(X) \) is one of \( \text{op}(X) = X \) or \( \text{op}(X) = X^{\text{T}} \) or \( \text{op}(X) = X^{\text{H}} \), alpha and beta are scalars, and A, B, C, and D are batched pointers to matrices, with \( \text{op}(A) \) an \( m \times k \) by \( \text{batch\_count} \) batched matrix, \( \text{op}(B) \) a \( k \times n \) by \( \text{batch\_count} \) batched matrix and \( C \) and \( D \) are \( m \times n \) by \( \text{batch\_count} \) batched matrices. The batched matrices are an array of pointers to matrices. The number of pointers to matrices is \( \text{batch\_count} \). Supported types are as follows:

- rocblas_datatype\_f64\_r = a\_type = b\_type = c\_type = d\_type = compute\_type
- rocblas_datatype\_f32\_r = a\_type = b\_type = c\_type = d\_type = compute\_type
- rocblas_datatype\_f16\_r = a\_type = b\_type = c\_type = d\_type = compute\_type
- rocblas_datatype\_bf16\_r = a\_type = b\_type = c\_type = d\_type = compute\_type
- rocblas_datatype\_i8\_r = a\_type = b\_type; rocblas_datatype\_i32\_r = c\_type = d\_type = compute\_type
- rocblas_datatype\_f32\_c = a\_type = b\_type = c\_type = d\_type = compute\_type
- rocblas_datatype\_f64\_c = a\_type = b\_type = c\_type = d\_type = compute\_type

Below are restrictions for rocblas_datatype\_i8\_r = a\_type = b\_type; rocblas_datatype\_i32\_r = c\_type = d\_type = compute\_type:

- \( k \) must be a multiple of 4
- \( \text{lda} \) must be a multiple of 4 if \( \text{transA} = \text{rocblas\_operation\_transpose} \)
- \( \text{ldb} \) must be a multiple of 4 if \( \text{transB} = \text{rocblas\_operation\_none} \)
- for \( \text{transA} = \text{rocblas\_operation\_transpose} \) or \( \text{transB} = \text{rocblas\_operation\_none} \) the matrices A and B must have each 4 consecutive values in the k dimension packed. This packing can be achieved with the following pseudo-code. The code assumes the original matrices are in A and B, and the packed matrices are A\_packed and B\_packed. The size of the A\_packed matrix is the same as the size of the A matrix, and the size of the B\_packed matrix is the same as the size of the B matrix.

```c
if (transA == rocblas_operation_transpose)
{
    int nb = 4;
    for(int i_m = 0; i_m < m; i_m++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            // Pseudo-code continues...
        }
    }
}
```

(continues on next page)
A_packed[i_k % nb + (i_m + (i_k / nb) * lda) * nb] = A[i_m + i_k * lda];
}
}
}
else
{
    A_packed = A;
}
if(transB == rocblas_operation_transpose)
{
    int nb = 4;
    for(int i_n = 0; i_n < m; i_n++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            B_packed[i_k % nb + (i_n + (i_k / nb) * lda) * nb] = B[i_n + i_k * lda];
        }
    }
}
else
{
    B_packed = B;
}

Parameters

– [in] transB: [rocblas_operation] specifies the form of op( B ).
– [in] m: [rocblas_int] matrix dimension m.
– [in] alpha: [const void *] device pointer or host pointer specifying the scalar alpha. Same
datatype as compute_type.
– [in] a: [void *] device pointer storing array of pointers to each matrix A_i.
– [in] a_type: [rocblas_datatype] specifies the datatype of each matrix A_i.
– [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
– [in] b: [void *] device pointer storing array of pointers to each matrix B_i.
– [in] b_type: [rocblas_datatype] specifies the datatype of each matrix B_i.
– [in] ldb: [rocblas_int] specifies the leading dimension of each B_i.
– [in] beta: [const void *] device pointer or host pointer specifying the scalar beta. Same
datatype as compute_type.
– [in] c: [void *] device array of device pointers to each matrix C_i.
– [in] c_type: [rocblas_datatype] specifies the datatype of each matrix C_i.
– [in] ldc: [rocblas_int] specifies the leading dimension of each C_i.
rocblas_status rocblas_gemm_strided_batched_ex (rocblas_handle handle, rocblas_operation transA, rocblas_operation transB, rocblas_int m, rocblas_int n, rocblas_int k, const void *alpha, const void *a, rocblas_datatype a_type, rocblas_int lda, rocblas_stride stride_a, const void *b, rocblas_datatype b_type, rocblas_int ldb, rocblas_stride stride_b, const void *beta, const void *c, rocblas_datatype c_type, rocblas_int ldc, rocblas_stride stride_c, void *d, rocblas_datatype d_type, rocblas_int ldd, rocblas_stride stride_d, rocblas_int batch_count, rocblas_datatype compute_type, rocblas_gemm_algo algo, int32_t solution_index, uint32_t flags)

BLAS EX API.

GEMM_STRIDED_BATCHED_EX performs one of the strided_batched matrix-matrix operations

\[ D_i = \alpha \cdot \text{op}(A_i) \cdot \text{op}(B_i) + \beta \cdot C_i, \text{ for } i = 1, \ldots, \text{batch_count} \]

where \( \text{op}(X) \) is one of

\[
\begin{align*}
\text{op}(X) &= X & \text{or} \\
\text{op}(X) &= X^{\ast T} & \text{or} \\
\text{op}(X) &= X^{\ast H}, \\
\end{align*}
\]

alpha and beta are scalars, and \( A, B, C, \) and \( D \) are strided_batched matrices, with \( \text{op}(A) \) an \( m \) by \( k \) by \( \text{batch_count} \) strided_batched matrix, \( \text{op}(B) \) a \( k \) by \( n \) by \( \text{batch_count} \) strided_batched matrix and \( C \) and \( D \) are \( m \) by \( n \) by \( \text{batch_count} \) strided_batched matrices.

The strided_batched matrices are multiple matrices separated by a constant stride. The number of matrices is \( \text{batch_count} \).

Supported types are as follows:

- rocblas_datatype_f64_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f32_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f16_r = a_type = b_type = c_type = d_type = compute_type
- rocblas_datatype_f16_r = a_type = b_type = c_type = d_type; rocblas_datatype_f32_r = compute_type
• rocblas_datatype_bf16_r = a_type = b_type = c_type = d_type; rocblas_datatype_f32_r = compute_type
• rocblas_datatype_i8_r = a_type = b_type; rocblas_datatype_i32_r = c_type = d_type = compute_type
• rocblas_datatype_f32_c = a_type = b_type = c_type = d_type = compute_type
• rocblas_datatype_f64_c = a_type = b_type = c_type = d_type = compute_type

Below are restrictions for rocblas_datatype_i8_r = a_type = b_type; rocblas_datatype_i32_r = c_type = d_type = compute_type:

• k must be a multiple of 4
• lda must be a multiple of 4 if transA == rocblas_operation_transpose
• ldb must be a multiple of 4 if transB == rocblas_operation_none

• for transA == rocblas_operation_transpose or transB == rocblas_operation_none the matrices A and B must have each 4 consecutive values in the k dimension packed. This packing can be achieved with the following pseudo-code. The code assumes the original matrices are in A and B, and the packed matrices are A_packed and B_packed. The size of the A_packed matrix is the same as the size of the A matrix, and the size of the B_packed matrix is the same as the size of the B matrix.

```c
if(transA == rocblas_operation_none)
{
    int nb = 4;
    for(int i_m = 0; i_m < m; i_m++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            A_packed[i_k % nb + (i_m + (i_k / nb) * lda) * nb] = A[i_m + i_k * lda];
        }
    }
}
else
{
    A_packed = A;
}

if(transB == rocblas_operation_transpose)
{
    int nb = 4;
    for(int i_n = 0; i_n < m; i_n++)
    {
        for(int i_k = 0; i_k < k; i_k++)
        {
            B_packed[i_k % nb + (i_n + (i_k / nb) * lda) * nb] = B[i_n + i_k * lda];
        }
    }
}
else
{
    B_packed = B;
}
```

Parameters

• [in] handle: [rocblas_handle] handle to the rocblas library context queue.
• [in] transB: [rocblas_operation] specifies the form of op( B ).
• [in] m: [rocblas_int] matrix dimension m.
• [in] k: [rocblas_int] matrix dimension k.
• [in] alpha: [const void *] device pointer or host pointer specifying the scalar alpha. Same datatype as compute_type.
• [in] a: [void *] device pointer pointing to first matrix A_1.
• [in] a_type: [rocblas_datatype] specifies the datatype of each matrix A_i.
• [in] lda: [rocblas_int] specifies the leading dimension of each A_i.
• [in] stride_a: [rocblas_stride] specifies stride from start of one A_i matrix to the next A_(i + 1).
• [in] b: [void *] device pointer pointing to first matrix B_1.
• [in] b_type: [rocblas_datatype] specifies the datatype of each matrix B_i.
• [in] ldb: [rocblas_int] specifies the leading dimension of each B_i.
• [in] stride_b: [rocblas_stride] specifies stride from start of one B_i matrix to the next B_(i + 1).
• [in] beta: [const void *] device pointer or host pointer specifying the scalar beta. Same datatype as compute_type.
• [in] c: [void *] device pointer pointing to first matrix C_1.
• [in] c_type: [rocblas_datatype] specifies the datatype of each matrix C_i.
• [in] ldc: [rocblas_int] specifies the leading dimension of each C_i.
• [in] stride_c: [rocblas_stride] specifies stride from start of one C_i matrix to the next C_(i + 1).
• [out] d: [void *] device pointer storing each matrix D_i.
• [in] d_type: [rocblas_datatype] specifies the datatype of each matrix D_i.
• [in] ldd: [rocblas_int] specifies the leading dimension of each D_i.
• [in] stride_d: [rocblas_stride] specifies stride from start of one D_i matrix to the next D_(i + 1).
• [in] batch_count: [rocblas_int] number of gemm operations in the batch.
• [in] compute_type: [rocblas_datatype] specifies the data type of computation.
• [in] algo: [rocblas_gemm_algo] enumerant specifying the algorithm type.
• [in] solution_index: [int32_t] reserved for future use.

rocblas_status rocblas_trsm_ex(rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const void *alpha, const void *A, rocblas_int lda, void *B, rocblas_int ldb, const void *invA, rocblas_int invA_size, rocblas_datatype compute_type)
TRSM_EX solves

\begin{align*}
\text{op}(A) \cdot X &= \text{alpha} \cdot B \\
\text{or} \\
X \cdot \text{op}(A) &= \text{alpha} \cdot B,
\end{align*}

where alpha is a scalar, X and B are m by n matrices, A is triangular matrix and \text{op}(A) is one of

\begin{align*}
\text{op}(A) &= A \\
\text{or} \\
\text{op}(A) &= A^T \\
\text{or} \\
\text{op}(A) &= A^H.
\end{align*}

The matrix X is overwritten on B.

TRSM_EX gives the user the ability to reuse the invA matrix between runs. If invA == NULL, rocblas_trsm_ex will automatically calculate invA on every run.

Setting up invA: The accepted invA matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix A, followed by any smaller diagonal block that remains. To set up invA it is recommended that rocblas_trtri_batched be used with matrix A as the input.

Device memory of size 128 x k should be allocated for invA ahead of time, where k is m when rocblas_side_left and is n when rocblas_side_right. The actual number of elements in invA should be passed as invA_size.

To begin, rocblas_trtri_batched must be called on the full 128x128 sized diagonal blocks of matrix A. Below are the restricted parameters:

- \( n = 128 \)
- \( \text{ldinvA} = 128 \)
- \( \text{stride}_\text{invA} = 128 \times 128 \)
- \( \text{batch}_\text{count} = \frac{k}{128} \)

Then any remaining block may be added:

- \( n = k \mod 128 \)
- \( \text{invA} = \text{invA} + \text{stride}_\text{invA} \times \text{previous}_\text{batch}_\text{count} \)
- \( \text{ldinvA} = 128 \)
- \( \text{batch}_\text{count} = 1 \)

Parameters

- \([\text{in}]\ \text{handle}: [\text{rocblas}\_\text{handle}]\ \text{handle to the rocblas library context queue.}\)
- \([\text{in}]\ \text{side}: [\text{rocblas}\_\text{side}]\ \text{rocblas}_\text{side}\_\text{left}: \ \text{op}(A) \cdot X = \text{alpha} \cdot B.\ \text{rocblas}_\text{side}\_\text{right}: X \cdot \text{op}(A) = \text{alpha} \cdot B.\)
- \([\text{in}]\ \text{uplo}: [\text{rocblas}\_\text{fill}]\ \text{rocblas}_\text{fill}\_\text{upper}: A\ \text{is an upper triangular matrix.}\ \text{rocblas}_\text{fill}\_\text{lower}: A\ \text{is a lower triangular matrix.}\)
- \([\text{in}]\ \text{transA}: [\text{rocblas}\_\text{operation}]\ \text{transB}: \ \text{op}(A) = A.\ \text{rocblas}_\text{operation}\_\text{transpose}: \ \text{op}(A) = A^T.\ \text{rocblas}_\text{operation}\_\text{conjugate}\_\text{transpose}: \ \text{op}(A) = A^H.\)
- \([\text{in}]\ \text{diag}: [\text{rocblas}\_\text{diagonal}]\ \text{rocblas}_\text{diagonal}\_\text{unit}: A\ \text{is assumed to be unit triangular.}\ \text{rocblas}_\text{diagonal}\_\text{non}\_\text{unit}: A\ \text{is not assumed to be unit triangular.}\)
- \([\text{in}]\ m: [\text{rocblas}\_\text{int}]\ \text{m specifies the number of rows of B.} \ m \geq 0.\)
- \([\text{in}]\ n: [\text{rocblas}\_\text{int}]\ \text{n specifies the number of columns of B.} \ n \geq 0.\)
- \([\text{in}]\ \text{alpha}: [\text{void}]\ \text{device pointer or host pointer specifying the scalar alpha. When alpha is zero then A is not referenced, and B need not be set before entry.}\)
• [in] A: [void *] device pointer storing matrix A. of dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.

• [in] lda: [rocblas_int] lda specifies the first dimension of A. if side = rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).

• [inout] B: [void *] device pointer storing matrix B. B is of dimension ( ldb, n ). Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

• [in] ldb: [rocblas_int] ldb specifies the first dimension of B. ldb >= max( 1, m ).

• [in] invA: [void *] device pointer storing the inverse diagonal blocks of A. invA is of dimension ( ld_invA, k ), where k is m when rocblas_side_left and is n when rocblas_side_right. ld_invA must be equal to 128.

• [in] invA_size: [rocblas_int] invA_size specifies the number of elements of device memory in invA.

• [in] compute_type: [rocblas_datatype] specifies the datatype of computation

rocblas_status rocblas_trsm_batched_ex (rocblas_handle handle, rocblas_side side, rocblas_fill uplo, rocblas_operation transA, rocblas_diagonal diag, rocblas_int m, rocblas_int n, const void *alpha, const void *A, rocblas_int lda, void *B, rocblas_int ldb, rocblas_int batch_count, const void *invA, rocblas_int invA_size, rocblas_datatype compute_type)

BLAS EX API

TRSM_BATCHED_EX solves

\[(A_i)X_i = \alpha B_i \text{ or } X_i(A_i) = \alpha B_i,\]

for \(i = 1, \ldots, \text{batch_count};\) and where \(\alpha\) is a scalar, \(X\) and \(B\) are arrays of \(m\) by \(n\) matrices, \(A\) is an array of triangular matrices and each \(A_i\) is one of

\[
\begin{align*}
\text{op}(A_i) &= A_i & \text{or} \quad \text{op}(A_i) &= A_i^T & \text{or} \quad \text{op}(A_i) &= A_i^H.
\end{align*}
\]

Each matrix \(X_i\) is overwritten on \(B_i\).

TRSM_EX gives the user the ability to reuse the invA matrix between runs. If invA == NULL, rocblas_trsm_batched_ex will automatically calculate each invA_i on every run.

Setting up invA: Each accepted invA_i matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix \(A_i\), followed by any smaller diagonal block that remains. To set up each invA_i it is recommended that rocblas_trtri_batched be used with matrix \(A_i\) as the input. invA is an array of pointers of \(\text{batch_count}\) length holding each invA_i.

Device memory of size 128 x \(k\) should be allocated for each invA_i ahead of time, where \(k\) is \(m\) when rocblas_side_left and is \(n\) when rocblas_side_right. The actual number of elements in each invA_i should be passed as invA_size.

To begin, rocblas_trtri_batched must be called on the full 128x128 sized diagonal blocks of each matrix \(A_i\). Below are the restricted parameters:

• \(n = 128\)

• \(ldinvA = 128\)

• \(stride_{invA} = 128x128\)
batch_count = k / 128,

Then any remaining block may be added:

- n = k % 128
- invA = invA + stride_invA * previous_batch_count
- ldinvA = 128
- batch_count = 1

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] side: [rocblas_side] rocblas_side_left: op(A)*X = alpha*B. rocblas_side_right: X*op(A) = alpha*B.
- [in] uplo: [rocblas_fill] rocblas_fill_upper: each A_i is an upper triangular matrix. rocblas_fill_lower: each A_i is a lower triangular matrix.
- [in] transA: [rocblas_operation] transB: op(A) = A. rocblas_operation_transpose: op(A) = A^T. rocblas_operation_conjugate_transpose: op(A) = A^H.
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: each A_i is assumed to be unit triangular. rocblas_diagonal_non_unit: each A_i is not assumed to be unit triangular.
- [in] m: [rocblas_int] m specifies the number of rows of each B_i. m >= 0.
- [in] n: [rocblas_int] n specifies the number of columns of each B_i. n >= 0.
- [in] alpha: [void *] device pointer or host pointer alpha specifying the scalar alpha. When alpha is &zero then A is not referenced, and B need not be set before entry.
- [in] A: [void *] device array of device pointers storing each matrix A_i. each A_i is of dimension ( lda, k ), where k is m when rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is accessed.
- [in] lda: [rocblas_int] lda specifies the first dimension of each A_i. if side = rocblas_side_left, lda >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).
- [inout] B: [void *] device array of device pointers storing each matrix B_i. each B_i is of dimension ( ldb, n ). Before entry, the leading m by n part of the array B_i must contain the right-hand side matrix B_i and on exit is overwritten by the solution matrix X_i
- [in] ldb: [rocblas_int] ldb specifies the first dimension of each B_i. ldb >= max( 1, m ).
- [in] invA: [void *] device array of device pointers storing the inverse diagonal blocks of each A_i. each invA_i is of dimension ( ld_invA, k ), where k is m when rocblas_side_left and is n when rocblas_side_right. ld_invA must be equal to 128.
- [in] invA_size: [rocblas_int] invA_size specifies the number of elements of device memory in each invA_i.
- [in] compute_type: [rocblas_datatype] specifies the datatype of computation.
roblas_status roblas_trsm_strided_batched_ex (rocblas_handle handle, roblas_side side, roblas_fill uplo, roblas_operation transA, roblas_diagonal diag, roblas_int m, roblas_int n, const void *alpha, const void *A, roblas_int lda, roblas_stride stride_A, void *B, roblas_int ldb, roblas_stride stride_B, roblas_int batch_count, const void *invA, roblas_int invA_size, roblas_stride stride_invA, roblas_datatype compute_type)

BLAS EX API

TRSM_STRIDED_BATCHED_EX solves

\[
\text{op}(A_i) \times X_i = \alpha B_i \quad \text{or} \quad X_i \times \text{op}(A_i) = \alpha B_i,
\]

for \(i = 1, \ldots, \text{batch\_count}\); and where alpha is a scalar, X and B are strided batched m by n matrices, A is a strided batched triangular matrix and \(\text{op}(A_i)\) is one of

\[
\text{op}(A_i) = A_i \quad \text{or} \quad \text{op}(A_i) = A_i^T \quad \text{or} \quad \text{op}(A_i) = A_i^H.
\]

Each matrix \(X_i\) is overwritten on \(B_i\).

TRSM_EX gives the user the ability to reuse each \(\text{invA}_i\) matrix between runs. If \(\text{invA} == \text{NULL}\), \roblas_trsm_strided_batched_ex will automatically calculate each \(\text{invA}_i\) on every run.

Setting up \(\text{invA}\): Each accepted \(\text{invA}_i\) matrix consists of the packed 128x128 inverses of the diagonal blocks of matrix \(A_i\), followed by any smaller diagonal block that remains. To set up \(\text{invA}_i\) it is recommended that \roblas_trtri_batched be used with matrix \(A_i\) as the input. \(\text{invA}\) is a contiguous piece of memory holding each \(\text{invA}_i\).

Device memory of size 128 x k should be allocated for each \(\text{invA}_i\) ahead of time, where k is m when \roblas_side_left and is n when \roblas_side_right. The actual number of elements in each \(\text{invA}_i\) should be passed as \(\text{invA}\_size\).

To begin, \roblas_trtri_batched must be called on the full 128x128 sized diagonal blocks of each matrix \(A_i\). Below are the restricted parameters:

- \(n = 128\)
- \(\text{ldinvA} = 128\)
- \(\text{stride} \_\text{invA} = 128 \times 128\)
- \(\text{batch\_count} = k / 128\),

Then any remaining block may be added:

- \(n = k \% 128\)
- \(\text{invA} = \text{invA} + \text{stride} \_\text{invA} \times \text{previous\_batch\_count}\)
- \(\text{ldinvA} = 128\)
- \(\text{batch\_count} = 1\)

Parameters

- [in] handle: [rocblas_handle] handle to the rocblas library context queue.
- [in] side: [rocblas_side] \roblas_side_left: \(\text{op}(A) \times X = \alpha B\). \roblas_side_right: \(X \times \text{op}(A) = \alpha B\).
ReadTheDocs-Breathe Documentation, Release 1.0.0

- [in] uplo: [rocblas_fill] rocblas_fill_upper: each A_i is an upper triangular matrix.
  rocblas_fill_lower: each A_i is a lower triangular matrix.
- [in] transA: [rocblas_operation] transB: op(A) = A. rocblas_operation_transpose: op(A) = A^T.
  rocblas_operation_conjugate_transpose: op(A) = A^H.
- [in] diag: [rocblas_diagonal] rocblas_diagonal_unit: each A_i is assumed to be unit triangular.
  rocblas_diagonal_non_unit: each A_i is not assumed to be unit triangular.
- [in] m: [rocblas_int] m specifies the number of rows of each B_i. m >= 0.
- [in] n: [rocblas_int] n specifies the number of columns of each B_i. n >= 0.
- [in] alpha: [void *] device pointer or host pointer specifying the scalar alpha. When alpha is
  zero then A is not referenced, and B need not be set before entry.
- [in] A: [void *] device pointer storing matrix A. of dimension ( lda, k ), where k is m when
  rocblas_side_left and is n when rocblas_side_right only the upper/lower triangular part is
  accessed.
- [in] lda: [rocblas_int] lda specifies the first dimension of A. if side = rocblas_side_left, lda
  >= max( 1, m ), if side = rocblas_side_right, lda >= max( 1, n ).
- [in] stride_A: [rocblas_stride] The stride between each A matrix.
- [inout] B: [void *] device pointer pointing to first matrix B_i. each B_i is of dimension ( ldb, n ).
  Before entry, the leading m by n part of each array B_i must contain the right-hand side of
  matrix B_i, and on exit is overwritten by the solution matrix X_i.
- [in] ldb: [rocblas_int] ldb specifies the first dimension of each B_i. ldb >= max( 1, m ).
- [in] stride_B: [rocblas_stride] The stride between each B_i matrix.
- [in] invA: [void *] device pointer storing the inverse diagonal blocks of each A_i. invA
  points to the first invA_1. each invA_i is of dimension ( ld_invA, k ), where k is m when
  rocblas_side_left and is n when rocblas_side_right. ld_invA must be equal to 128.
- [in] invA_size: [rocblas_int] invA_size specifies the number of elements of device memory
  in each invA_i.
- [in] compute_type: [rocblas_datatype] specifies the datatype of computation

**const char *rocblas_status_to_string(rocblas_status status)**

BLAS Auxiliary API

rocblas_status_to_string

Returns string representing rocblas_status value

**Parameters**

- [in] status: [rocblas_status] rocBLAS status to convert to string

**rocblas_status rocblas_get_version_string(char *buf, size_t len)**

loads char* buf with the rocblas library version. size_t len is the maximum length of char* buf.

**Parameters**

- [inout] buf: pointer to buffer for version string
rocblas_status rocblas_start_device_memory_size_query (rocblas_handle handle)
Indicates that subsequent rocBLAS kernel calls should collect the optimal device memory size in bytes for their given kernel arguments, and keep track of the maximum. Each kernel call can reuse temporary device memory on the same stream, so the maximum is collected. Returns rocblas_status_size_query_mismatch if another size query is already in progress; returns rocblas_status_success otherwise.

Parameters
- [in] handle: rocblas handle

rocblas_status rocblas_stop_device_memory_size_query (rocblas_handle handle, size_t *size)
Stops collecting optimal device memory size information Returns rocblas_status_size_query_mismatch if a collection is not underway; rocblas_status_invalid_handle if handle is nullptr; rocblas_status_invalid_pointer if size is nullptr; rocblas_status_success otherwise

Parameters
- [in] handle: rocblas handle
- [out] size: maximum of the optimal sizes collected

rocblas_status rocblas_get_device_memory_size (rocblas_handle handle, size_t *size)
Gets the current device memory size for the handle Returns rocblas_status_invalid_handle if handle is nullptr; rocblas_status_invalid_pointer if size is nullptr; rocblas_status_success otherwise

Parameters
- [in] handle: rocblas handle
- [out] size: current device memory size for the handle

rocblas_status rocblas_set_device_memory_size (rocblas_handle handle, size_t size)
Changes the size of allocated device memory at runtime. Any previously allocated device memory is freed.

If size > 0 sets the device memory size to the specified size (in bytes) If size == 0 frees the memory allocated so far, and lets rocBLAS manage device memory in the future, expanding it when necessary Returns rocblas_status_invalid_handle if handle is nullptr; rocblas_status_invalid_pointer if size is nullptr; rocblas_status_success otherwise

Parameters
- [in] handle: rocblas handle
- [in] size: size of allocated device memory

bool rocblas_is_managing_device_memory (rocblas_handle handle)
Returns true when device memory in handle is managed by rocBLAS

Parameters
- [in] handle: rocblas handle

file rocblas-types.h
#include "rocblas_bfloat16.h"#include <float.h>#include <math.h>#include <stdbool.h>#include <stddef.h>#include <stdint.h>#include "rocblas-complex-types.h” rocblas-types.h defines data types used by rocblas
Defines

__STDC_WANT_IEC_60559_TYPES_EXT__

Typedefs

typedef struct _rocblas_handle *rocblas_handle
rocblas_handle is a structure holding the rocblas library context. It must be initialized using rocblas_create_handle() and the returned handle must be passed to all subsequent library function calls. It should be destroyed at the end using rocblas_destroy_handle().

typedef struct ihipStream_t *hipStream_t
typedef int32_t rocblas_int
To specify whether int32 or int64 is used.
typedef int64_t rocblas_stride
typedef float rocblas_float
typedef double rocblas_double

Enums

enum rocblas_operation
Used to specify whether the matrix is to be transposed or not.
parameter constants. numbering is consistent with CBLAS, ACML and most standard C BLAS libraries
Values:
enumerator rocblas_operation_none = 111
Operate with the matrix.
enumerator rocblas_operation_transpose = 112
Operate with the transpose of the matrix.
enumerator rocblas_operation_conjugate_transpose = 113
Operate with the conjugate transpose of the matrix.

enum rocblas_fill
Used by the Hermitian, symmetric and triangular matrix routines to specify whether the upper or lower triangle is being referenced.
Values:
enumerator rocblas_fill_upper = 121
Upper triangle.
enumerator rocblas_fill_lower = 122
Lower triangle.
enumerator rocblas_fill_full = 123

enum rocblas_diagonal
It is used by the triangular matrix routines to specify whether the matrix is unit triangular.
Values:
enumerator rocblas_diagonal_non_unit = 131
Non-unit triangular.
enumerator rocblas_diagonal_unit = 132
    Unit triangular.

doc enum rocblas_side
    Indicates the side matrix A is located relative to matrix B during multiplication.
    Values:
        enumerator rocblas_side_left = 141
            Multiply general matrix by symmetric, Hermitian or triangular matrix on the left.
        enumerator rocblas_side_right = 142
            Multiply general matrix by symmetric, Hermitian or triangular matrix on the right.
        enumerator rocblas_side_both = 143

doc enum rocblas_status
    rocblas status codes definition
    Values:
        enumerator rocblas_status_success = 0
            success
        enumerator rocblas_status_invalid_handle = 1
            handle not initialized, invalid or null
        enumerator rocblas_status_not_implemented = 2
            function is not implemented
        enumerator rocblas_status_invalid_pointer = 3
            invalid pointer argument
        enumerator rocblas_status_invalid_size = 4
            invalid size argument
        enumerator rocblas_status_memory_error = 5
            failed internal memory allocation, copy or dealloc
        enumerator rocblas_status_internal_error = 6
            other internal library failure
        enumerator rocblas_status_perf_degraded = 7
            performance degraded due to low device memory
        enumerator rocblas_status_size_query_mismatch = 8
            unmatched start/stop size query
        enumerator rocblas_status_size_increased = 9
            queried device memory size increased
        enumerator rocblas_status_size_unchanged = 10
            queried device memory size unchanged
        enumerator rocblas_status_invalid_value = 11
            passed argument not valid
        enumerator rocblas_status_continue = 12
            nothing preventing function to proceed

doc enum rocblas_datatype
    Indicates the precision width of data stored in a blas type.
    Values:
enumerator rocblas_datatype_f16_r = 150
   16 bit floating point, real
enumerator rocblas_datatype_f32_r = 151
   32 bit floating point, real
enumerator rocblas_datatype_f64_r = 152
   64 bit floating point, real
enumerator rocblas_datatype_f16_c = 153
   16 bit floating point, complex
enumerator rocblas_datatype_f32_c = 154
   32 bit floating point, complex
enumerator rocblas_datatype_f64_c = 155
   64 bit floating point, complex
enumerator rocblas_datatype_i8_r = 160
   8 bit signed integer, real
enumerator rocblas_datatype_u8_r = 161
   8 bit unsigned integer, real
enumerator rocblas_datatype_i32_r = 162
   32 bit signed integer, real
enumerator rocblas_datatype_u32_r = 163
   32 bit unsigned integer, real
enumerator rocblas_datatype_i8_c = 164
   8 bit signed integer, complex
enumerator rocblas_datatype_u8_c = 165
   8 bit unsigned integer, complex
enumerator rocblas_datatype_i32_c = 166
   32 bit signed integer, complex
enumerator rocblas_datatype_u32_c = 167
   32 bit unsigned integer, complex
enumerator rocblas_datatype_bf16_r = 168
   16 bit bfloat, real
enumerator rocblas_datatype_bf16_c = 169
   16 bit bfloat, complex

enum rocblas_pointer_mode
   Indicates the pointer is device pointer or host pointer. This is typically used for scalars such as alpha and beta.
   Values:
   enumerator rocblas_pointer_mode_host = 0
      Scalar values affected by this variable will be located on the host.
   enumerator rocblas_pointer_mode_device = 1
      Scalar values affected by this variable will be located on the device.

enum rocblas_layer_mode
   Indicates if layer is active with bitmask.
   Values:
**enumerator rocblas_layer_mode_none** = 0b0000000000

No logging will take place.

**enumerator rocblas_layer_mode_log_trace** = 0b0000000001

A line containing the function name and value of arguments passed will be printed with each rocBLAS function call.

**enumerator rocblas_layer_mode_log_bench** = 0b0000000010

Outputs a line each time a rocBLAS function is called, this line can be used with rocblas-bench to make the same call again.

**enumerator rocblas_layer_mode_log_profile** = 0b0000000100

Outputs a YAML description of each rocBLAS function called, along with its arguments and number of times it was called.

**enum rocblas_gemm_algo**

Indicates if layer is active with bitmask.

**Values:**

**enumerator rocblas_gemm_algo_standard** = 0b0000000000

---

**file rocblas.h**

```
#include "rocblas-auxiliary.h"
#include "rocblas-export.h"
#include "rocblas-functions.h"
#include "rocblas-types.h"
#include "rocblas-version.h"   rocblas.h includes other *.h and exposes a common interface
```

**file buildinfo.cpp**

```
#include <stdio.h>
#include <sstream>
#include <string.h>
#include "definitions.h"
#include "rocblas-types.h"
#include "rocblas-functions.h"
#include "rocblas-version.h"
```

---

**Defines**

**TO_STR2 (x)**

**TO_STR (x)**

**VERSION_STRING**

---

**Functions**

**rocblas_status rocblas_get_version_string** (char *buf, size_t len)

loads char* buf with the rocblas library version. size_t len is the maximum length of char* buf.

**Parameters**

- [inout] buf: pointer to buffer for version string
- [in] len: length of buf

---

**file handle.cpp**

```
#include "handle.h"
#include <cstdlib>
```
Functions

void open_log_stream(const char *environment_variable_name, std::ostream *&log_os, std::ofstream &log_ofs)

Logging function.

open_log_stream Open stream log_os for logging. If the environment variable with name environment_variable_name is not set, then stream log_os to std::cerr. Else open a file at the full logfile path contained in the environment variable. If opening the file succeeds, stream to the file else stream to std::cerr.

[out] log_os std::ostream*& Output stream. Stream to std::cerr if environment_variable_name is not set, else set to stream to log_ofs

Parameters

• [in] environment_variable_name: const char* Name of environment variable that contains the full logfile path.

[out] log_ofs std::ofstream& Output file stream. If log_ofs->is_open()==true, then log_os will stream to log_ofs. Else it will stream to std::cerr.

file rochblas_auxiliary.cpp

#include <stdio.h>
#include <hip/hip_runtime.h>
#include “definitions.h”
#include “rochblas-types.h”
#include “handle.h”
#include “logging.h”
#include “utility.h”
#include “rochblas_unique_ptrchpp”
#include “rochblas-auxiliary.h”

Functions

rocblas_pointer_mode rocblas_pointer_to_mode (void *ptr)

Indicates whether the pointer is on the host or device.

rocblas_status rocblas_get_pointer_mode(rocblas_handle handle, rocblas_pointer_mode *mode)

get rocblas_pointer_mode

rocblas_status rocblas_set_pointer_mode(rocblas_handle handle, rocblas_pointer_mode mode)

set rocblas_pointer_mode

rocblas_status rocblas_create_handle(rocblas_handle *handle)

create handle

rocblas_status rocblas_destroy_handle(rocblas_handle handle)

destroy handle

rocblas_status rocblas_set_stream(rocblas_handle handle, hipStream_t stream_id)

remove any streams from handle, and add one

rocblas_status rocblas_get_stream(rocblas_handle handle, hipStream_t *stream_id)

get stream [0] from handle

__global__ void copy_void_ptr_vector_kernel (rocblas_int n, rocblas_int elem_size, const void *x_h, rocblas_int incx, void *y_d, rocblas_int incy)

void x_h, rocblas_int incx, void *y_d, rocblas_int incy)

copy vector from host to device

rocblas_status rocblas_get_vector(rocblas_int n, rocblas_int elem_size, const void *x_d, rocblas_int incx, void *y_h, rocblas_int incy)

copy vector from device to host

__global__ void copy_void_ptr_matrix_kernel (rocblas_int rows, rocblas_int cols, size_t elem_size, const void *a, rocblas_int lda, void *b, rocblas_int ldb)
rocblas_status rocblas_set_matrix(rocblas_int rows, rocblas_int cols, rocblas_int elem_size,
    const void *a_h, rocblas_int lda, void *b_d, rocblas_int ldb)

    copy matrix from host to device

rocblas_status rocblas_get_matrix(rocblas_int rows, rocblas_int cols, rocblas_int elem_size,
    const void *a_d, rocblas_int lda, void *b_h, rocblas_int ldb)

    copy matrix from device to host

Variables

constexpr size_t VEC_BUFF_MAX_BYTES = 1048576
constexpr rocblas_int NB_X = 256
constexpr size_t MAT_BUFF_MAX_BYTES = 1048576
constexpr rocblas_int MATRIX_DIM_X = 128
constexpr rocblas_int MATRIX_DIM_Y = 8

file status.cpp
    #include <hip/hip_runtime_api.h>#include “rocblas.h”#include “status.h”

Functions

rocblas_status get_rocblas_status_for_hip_status(hipError_t status)

dir ROCm_Libraries/rocBLAS

dir ROCm_Libraries

dir ROCm_Libraries/rocBLAS/src

dir ROCm_Libraries/rocBLAS/src/src

2.12.2 hipBLAS

2.12.2.1 Introduction

Please Refer here for Github link hipBLAS

hipBLAS is a BLAS marshalling library, with multiple supported backends. It sits between the application and a ‘worker’ BLAS library, marshalling inputs into the backend library and marshalling results back to the application. hipBLAS exports an interface that does not require the client to change, regardless of the chosen backend. Currently, hipBLAS supports rocblas and cuBLAS as backends.
2.12.2.1 Installing pre-built packages

Download pre-built packages either from ROCm’s package servers or by clicking the github releases tab and manually downloading, which could be newer. Release notes are available for each release on the releases tab.

```
sudo apt update && sudo apt install hipblas
```

2.12.2.1.2 Quickstart hipBLAS build

Bash helper build script (Ubuntu only)

The root of this repository has a helper bash script `install.sh` to build and install hipBLAS on Ubuntu with a single command. It does not take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it’s a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access, so it may prompt you for a password.

```
./install -h -- shows help
./install -id -- build library, build dependencies and install (-d flag only needs to be passed once on a system)
```

Manual build (all supported platforms)

If you use a distro other than Ubuntu, or would like more control over the build process, the hipblas build has helpful information on how to configure cmake and manually build.

2.12.2.2 Build

2.12.2.2.1 Dependencies For Building Library

CMake 3.5 or later

The build infrastructure for hipBLAS is based on Cmake v3.5. This is the version of cmake available on ROCm supported platforms. If you are on a headless machine without the x-windows system, we recommend using `ccmake`; if you have access to X-windows, we recommend using `cmake-gui`.

Install one-liners cmake:

```
Ubuntu: sudo apt install cmake-qt-gui
Fedora: sudo dnf install cmake-gui
```

2.12.2.2.2 Build Library Using Script (Ubuntu only)

The root of this repository has a helper bash script `install.sh` to build and install hipBLAS on Ubuntu with a single command. It does not take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it’s a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access, so it may prompt you for a password.

```
./install.sh -h -- shows help
./install.sh -id -- build library, build dependencies and install (-d flag only needs to be passed once on a system)
```
2.12.2.2.3 Build Library Using Individual Commands

```bash
mkdir -p [HIPBLAS_BUILD_DIR]/release
cd [HIPBLAS_BUILD_DIR]/release

# Default install location is in /opt/rocm, define -DCMAKE_INSTALL_PREFIX=<path> to specify other
# Default build config is 'Release', define -DCMAKE_BUILD_TYPE=<config> to specify other
CXX=/opt/rocm/bin/hcc ccmake [HIPBLAS_SOURCE]
make -j$(nproc)
sudo make install # sudo required if installing into system directory such as /opt/rocm
```

2.12.2.2.4 Build Library + Tests + Benchmarks + Samples Using Individual Commands

The repository contains source for clients that serve as samples, tests and benchmarks. Clients source can be found in the clients subdir.

**Dependencies (only necessary for hipBLAS clients)**

The hipBLAS samples have no external dependencies, but our unit test and benchmarking applications do. These clients introduce the following dependencies:

1. boost
2. **lapack**
   - lapack itself brings a dependency on a fortran compiler
3. googletest

Linux distros typically have an easy installation mechanism for boost through the native package manager.

Ubuntu: sudo apt install libboost-program-options-dev
Fedora: sudo dnf install boost-program-options

Unfortunately, googletest and lapack are not as easy to install. Many distros do not provide a googletest package with pre-compiled libraries, and the lapack packages do not have the necessary cmake config files for cmake to configure linking the cblas library. hipBLAS provide a cmake script that builds the above dependencies from source. This is an optional step; users can provide their own builds of these dependencies and help cmake find them by setting the CMAKE_PREFIX_PATH definition. The following is a sequence of steps to build dependencies and install them to the cmake default /usr/local.

(optional, one time only)

```bash
mkdir -p [HIPBLAS_BUILD_DIR]/release/deps
cd [HIPBLAS_BUILD_DIR]/release/deps
cmake --DBUILD_BOOST=OFF [HIPBLAS_SOURCE]/ddeps # assuming boost is installed
--through package manager as above
make -j$(nproc) install
```

Once dependencies are available on the system, it is possible to configure the clients to build. This requires a few extra cmake flags to the library cmake configure script. If the dependencies are not installed into system defaults (like /usr/local), you should pass the CMAKE_PREFIX_PATH to cmake to help find them.

-DCMAKE_PREFIX_PATH="<semicolon separated paths>"

2.12. ROCm Libraries 243
# Default install location is in /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to specify other
CXX=/opt/rocm/bin/hcc cmake -DBUILD_CLIENTS_TESTS=ON -DBUILD_CLIENTS_BENCHMARKS=ON
make -j$(nproc)
sudo make install  # sudo required if installing into system directory such as /opt/rocm

## 2.12.2.5 Common build problems

- **Issue:** HIP (/opt/rocm/hip) was built using hcc 1.0.xxx-xxx-xxx-xxx, but you are using /opt/rocm/hcc/hcc with version 1.0.yyy-yyy-yyy-yyy from hipcc. (version does not match). Please rebuild HIP including cmake or update HCC_HOME variable.

  **Solution:** Download HIP from github and use hcc to build from source and then use the build HIP instead of /opt/rocm/hip one or singly overwrite the new build HIP to this location.

- **Issue:** For Carrizo - HCC RUNTIME ERROR: Fail to find compatible kernel

  **Solution:** Add the following to the cmake command when configuring: -DCMAKE_CXX_FLAGS="-amdgpu-target=gfx801"

- **Issue:** For MI25 (Vega10 Server) - HCC RUNTIME ERROR: Fail to find compatible kernel

  **Solution:** export HCC_AMDGPU_TARGET=gfx900

### 2.12.2.3 Running

#### 2.12.2.3.1 Notice

Before reading this Wiki, it is assumed hipBLAS with the client applications has been successfully built as described in Build hipBLAS libraries and verification code

**Samples**

cd [BUILD_DIR]/clients/staging
./example-sscal

Example code that calls hipBLAS you can also see the following blog on the right side Example C code calling hipBLAS routine.

**Unit tests**

**Run tests with the following:**

cd [BUILD_DIR]/clients/staging
./hipblas-test

To run specific tests, use --gtest_filter=match where match is a ‘:’-separated list of wildcard patterns (called the positive patterns) optionally followed by a ‘-’ and another ‘:’-separated pattern list (called the negative patterns). For example, run gemv tests with the following:

cd [BUILD_DIR]/clients/staging
./hipblas-test --gtest_filter=*gemv*
Functions supported

A list of exported functions from hipblas can be found on the wiki

Platform: rocBLAS or cuBLAS

hipBLAS is a marshalling library, so it runs with either rocBLAS or cuBLAS configured as the backend BLAS library, chosen at cmake configure time.

2.12.2.3.2 hipBLAS interface examples

The hipBLAS interface is compatible with rocBLAS and cuBLAS-v2 APIs. Porting a CUDA application which originally calls the cuBLAS API to an application calling hipBLAS API should be relatively straightforward. For example, the hipBLAS SGEMV interface is

2.12.2.3.3 GEMV API

```c
hipblasStatus_t
hipblasSgemv( hipblasHandle_t handle,
             hipblasOperation_t trans,
             int m, int n, const float *alpha,
             const float *A, int lda,
             const float *x, int incx, const float *beta,
             float *y, int incy );
```

2.12.2.3.4 Batched and strided GEMM API

hipBLAS GEMM can process matrices in batches with regular strides. There are several permutations of these API's, the following is an example that takes everything

```c
hipblasStatus_t
hipblasSgemmStridedBatched( hipblasHandle_t handle,
                            hipblasOperation_t transa, hipblasOperation_t transb,
                            int m, int n, int k, const float *alpha,
                            const float *A, int lda, long long bsa,
                            const float *B, int ldb, long long bsb, const float *beta,
                            float *C, int ldc, long long bsc,
                            int batchCount);
```

hipBLAS assumes matrices A and vectors x, y are allocated in GPU memory space filled with data. Users are responsible for copying data from/to the host and device memory.

2.12.3 rocRAND

The rocRAND project provides functions that generate pseudo-random and quasi-random numbers.

The rocRAND library is implemented in the HIP programming language and optimised for AMD’s latest discrete GPUs. It is designed to run on top of AMD’s Radeon Open Compute ROCm runtime, but it also works on CUDA enabled GPUs.

Additionally, the project includes a wrapper library called hipRAND which allows user to easily port CUDA applications that use cuRAND library to the HIP layer. In ROCm environment hipRAND uses rocRAND, however in CUDA environment cuRAND is used instead.
2.12.3.1 Supported Random Number Generators

- XORWOW
- MRG32k3a
- Mersenne Twister for Graphic Processors (MTGP32)
- Philox (4x32, 10 rounds)
- bSobol32

2.12.3.2 Requirements

- Git
- cmake (3.0.2 or later)
- C++ compiler with C++11 support
- For AMD platforms:
  - ROCm (1.7 or later)
  - HCC compiler, which must be set as C++ compiler on ROCm platform.
- For CUDA platforms:
  - HIP (hcc is not required)
  - Latest CUDA SDK

Optional:

- GTest (required only for tests; building tests is enabled by default)
  - Use GTEST_ROOT to specify GTest location (also see FindGTest)
  - Note: If GTest is not already installed, it will be automatically downloaded and built
- TestU01 (required only for crush tests)
  - Use TESTU01_ROOT_DIR to specify TestU01 location
  - Note: If TestU01 is not already installed, it will be automatically downloaded and built
- Fortran compiler (required only for Fortran wrapper)
  - gfortran is recommended.
- Python 2.7+ or 3.5+ (required only for Python wrapper)

If some dependencies are missing, cmake script automatically downloads, builds and installs them. Setting DEPENDENCIES_FORCE_DOWNLOAD option ON forces script to not to use system-installed libraries, and to download all dependencies.
2.12.3.3 Build and Install

```bash
git clone https://github.com/ROCmSoftwarePlatform/rocRAND.git

# Go to rocRAND directory, create and go to build directory
cd rocRAND; mkdir build; cd build

# Configure rocRAND, setup options for your system
# Build options: BUILD_TEST, BUILD_BENCHMARK (off by default), BUILD_CRUSH_TEST (off by default)
# ! IMPORTANT!
# On ROCm platform set C++ compiler to HCC. You can do it by adding 'CXX=<path-to-hcc>' or just
# 'CXX=hcc' before 'cmake', or setting cmake option 'CMAKE_CXX_COMPILER' to path to the HCC compiler.

# [CXX=hcc] cmake -DBUILD_BENCHMARK=ON ../. # or cmake-gui ../.
# Build
# For ROCM-1.6, if a HCC runtime error is caught, consider setting
# HCC_AMDGPU_TARGET=<arch> in front of make as a workaround
make -j4
# Optionally, run tests if they're enabled
ctest --output-on-failure
# Install
[sudo] make install
```

Note: Existing gtest library in the system (especially static gtest libraries built with other compilers) may cause build failure; if errors are encountered with existing gtest library or other dependencies, DEPENDENCIES_FORCE_DOWNLOAD flag can be passed to cmake, as mentioned before, to help solve the problem.

Note: To disable inline assembly optimisations in rocRAND (for both the host library and the device functions provided in rocrand_kernel.h) set cmake option ENABLE_INLINE_ASM to OFF.

2.12.3.4 Running Unit Tests

```bash
# Go to rocRAND build directory
cd rocRAND; cd build
# To run all tests
test
# To run unit tests
./test/<unit-test-name>
```

2.12.3.5 Running Benchmarks

```bash
# Go to rocRAND build directory
cd rocRAND; cd build
# To run benchmark for generate functions:
# engine -> all, xorwow, mrg32k3a, mtgp32, philox, sobol32
# distribution -> all, uniform-uint, uniform-float, uniform-double, normal-float, normal-double,
# log-normal-float, log-normal-double, poisson
# Further option can be found using --help
./benchmark/benchmark_rocrand_generate --engine <engine> --dis <distribution>
```

(continues on next page)
# To run benchmark for device kernel functions:
# engine -> all, xorwow, mrg32k3a, mtgp32, philox, sobol32
# distribution -> all, uniform-uint, uniform-float, uniform-double, normal-float,
#                log-normal-float, log-normal-double, poisson, discrete-poisson,
#                discrete-custom
# further option can be found using --help
./benchmark/benchmark_rocrand_kernel --engine <engine> --dis <distribution>
# To compare against cuRAND (cuRAND must be supported):
./benchmark/benchmark_curand_generate --engine <engine> --dis <distribution>
./benchmark/benchmark_curand_kernel --engine <engine> --dis <distribution>

## 2.12.3.6 Running Statistical Tests

# Go to rocRAND build directory
cd rocRAND; cd build
# To run "crush" test, which verifies that generated pseudorandom
# numbers are of high quality:
# engine -> all, xorwow, mrg32k3a, mtgp32, philox
./test/crush_test_rocrand --engine <engine>
# To run Pearson Chi-squared and Anderson-Darling tests, which verify
# that distribution of random number agrees with the requested distribution:
# engine -> all, xorwow, mrg32k3a, mtgp32, philox, sobol32
# distribution -> all, uniform-float, uniform-double, normal-float, normal-double,
#                log-normal-float, log-normal-double, poisson
./test/stat_test_rocrand_generate --engine <engine> --dis <distribution>

## 2.12.3.7 Documentation

# go to rocRAND doc directory
cd rocRAND; cd doc
# run doxygen
doxygen Doxyfile
# open html/index.html

## 2.12.3.8 Wrappers

- C++ wrappers for host API of rocRAND and hipRAND are in files rocrand.hpp and hiprand.hpp.
- Fortran wrappers.
- Python wrappers: rocRAND and hipRAND.

### Support

Bugs and feature requests can be reported through the issue tracker.
2.12.4 rocFFT

rocFFT is a software library for computing Fast Fourier Transforms (FFT) written in HIP. It is part of AMD’s software ecosystem based on ROCm. In addition to AMD GPU devices, the library can also be compiled with the CUDA compiler using HIP tools for running on Nvidia GPU devices.

The rocFFT library:

- Provides a fast and accurate platform for calculating discrete FFTs.
- Supports single and double precision floating point formats.
- Supports 1D, 2D, and 3D transforms.
- Supports computation of transforms in batches.
- Supports real and complex FFTs.
- Supports lengths that are any combination of powers of 2, 3, 5.

2.12.4.1 API design

Please refer to the rocFFTAPI for current documentation. Work in progress.

2.12.4.2 Installing pre-built packages

Download pre-built packages either from ROCm’s package servers or by clicking the github releases tab and manually downloading, which could be newer. Release notes are available for each release on the releases tab.

```
sudo apt update && sudo apt install rocfft
```

2.12.4.3 Quickstart rocFFT build

Bash helper build script (Ubuntu only) The root of this repository has a helper bash script install.sh to build and install rocFFT on Ubuntu with a single command. It does not take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it’s a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access, so it may prompt you for a password. */install -h – shows help */install -id – build library, build dependencies and install globally (-d flag only needs to be specified once on a system) */install -c --cuda – build library and clients for cuda backend into a local directory Manual build (all supported platforms) If you use a distro other than Ubuntu, or would like more control over the build process, the rocfft build wiki has helpful information on how to configure cmake and manually build.

2.12.4.4 Manual build (all supported platforms)

If you use a distro other than Ubuntu, or would like more control over the build process, the rocfft build wiki has helpful information on how to configure cmake and manually build.

Library and API Documentation

Please refer to the Library documentation for current documentation.
2.12.4.5 Example

The following is a simple example code that shows how to use rocFFT to compute a 1D single precision 16-point complex forward transform.

```cpp
#include <iostream>
#include <vector>
#include "hip/hip_runtime_api.h"
#include "hip/hip_vector_types.h"
#include "rocfft.h"

int main()
{
    // rocFFT gpu compute
    // ==============================================================

    size_t N = 16;
    size_t Nbytes = N * sizeof(float2);

    // Create HIP device buffer
    float2 *x;
    hipMalloc(&x, Nbytes);

    // Initialize data
    std::vector<float2> cx(N);
    for (size_t i = 0; i < N; i++)
    {
        cx[i].x = 1;
        cx[i].y = -1;
    }

    // Copy data to device
    hipMemcpy(x, cx.data(), Nbytes, hipMemcpyHostToDevice);

    // Create rocFFT plan
    rocfft_plan plan = NULL;
    size_t length = N;
    rocfft_plan_create(&plan, rocfft_placement_inplace, rocfft_transform_type_complex_forward, rocfft_precision_single, 1, &length, 1, NULL);

    // Execute plan
    rocfft_execute(plan, (void**) &x, NULL, NULL);

    // Wait for execution to finish
    hipDeviceSynchronize();

    // Destroy plan
    rocfft_plan_destroy(plan);

    // Copy result back to host
    std::vector<float2> y(N);
    hipMemcpy(y.data(), x, Nbytes, hipMemcpyDeviceToHost);

    // Print results
    for (size_t i = 0; i < N; i++)
    {
        std::cout << y[i].x << ", " << y[i].y << std::endl;
    }
}
```

(continues on next page)
2.12.4.6 API

This section provides details of the library API

2.12.4.6.1 Types

There are few data structures that are internal to the library. The pointer types to these structures are given below. The user would need to use these types to create handles and pass them between different library functions.

```c
typedef struct rocfft_plan_t *rocfft_plan
    Pointer type to plan structure.
    This type is used to declare a plan handle that can be initialized with rocfft_plan_create

typedef struct rocfft_plan_description_t *rocfft_plan_description
    Pointer type to plan description structure.
    This type is used to declare a plan description handle that can be initialized with rocfft_plan_description_create

typedef struct rocfft_execution_info_t *rocfft_execution_info
    Pointer type to execution info structure.
    This type is used to declare an execution info handle that can be initialized with rocfft_execution_info_create
```

2.12.4.6.2 Library Setup and Cleanup

The following functions deals with initialization and cleanup of the library.

```c
rocfft_status rocfft_setup ()
    Library setup function, called once in program before start of library use.

rocfft_status rocfft_cleanup ()
    Library cleanup function, called once in program after end of library use.
```

2.12.4.6.3 Plan

The following functions are used to create and destroy plan objects.

```c
rocfft_status rocfft_plan_create (rocfft_plan *plan, rocfft_result_placement placement,
    rocfft_transform_type transform_type, rocfft_precision precision, size_t dimensions, const size_t *lengths, size_t number_of_transforms, const rocfft_plan_description description)
    Create an FFT plan.
    This API creates a plan, which the user can execute subsequently. This function takes many of the fundamental parameters needed to specify a transform. The parameters are self explanatory. The dimensions parameter can
take a value of 1, 2 or 3. The ‘lengths’ array specifies size of data in each dimension. Note that lengths[0] is the size of the innermost dimension, lengths[1] is the next higher dimension and so on. The ‘number_of_transforms’ parameter specifies how many transforms (of the same kind) needs to be computed. By specifying a value greater than 1, a batch of transforms can be computed with a single api call. Additionally, a handle to a plan description can be passed for more detailed transforms. For simple transforms, this parameter can be set to null ptr.

**Parameters**

- [out] plan: plan handle
- [in] placement: placement of result
- [in] transform_type: type of transform
- [in] precision: precision
- [in] dimensions: dimensions
- [in] lengths: dimensions sized array of transform lengths
- [in] number_of_transforms: number of transforms
- [in] description: description handle created by rocfft_plan_description_create; can be null ptr for simple transforms

**rocfft_status rocfft_plan_destroy (rocfft_plan plan)**

Destroy an FFT plan.

This API frees the plan. This function destructs a plan after it is no longer needed.

**Parameters**

- [in] plan: plan handle

The following functions are used to query for information after a plan is created.

**rocfft_status rocfft_plan_get_work_buffer_size (const rocfft_plan plan, size_t *size_in_bytes)**

Get work buffer size.

This is one of plan query functions to obtain information regarding a plan. This API gets the work buffer size.

**Parameters**

- [in] plan: plan handle
- [out] size_in_bytes: size of needed work buffer in bytes

**rocfft_status rocfft_plan_get_print (const rocfft_plan plan)**

Print all plan information.

This is one of plan query functions to obtain information regarding a plan. This API prints all plan info to stdout to help user verify plan specification.

**Parameters**

- [in] plan: plan handle
2.12.4.6.4 Plan description

Most of the times, `rocfft_plan_create()` is all is needed to fully specify a transform. And the description object can be skipped. But when a transform specification has more details a description object need to be created and set up and the handle passed to the `rocfft_plan_create()`. Functions referred below can be used to manage plan description in order to specify more transform details. The plan description object can be safely deleted after call to the plan api `rocfft_plan_create()`.

```c
rocfft_status rocfft_plan_description_create (rocfft_plan_description *description)
```

Create plan description.

This API creates a plan description with which the user can set more plan properties

Parameters

- [out] description: plan description handle

```c
rocfft_status rocfft_plan_description_destroy (rocfft_plan_description description)
```

Destroy a plan description.

This API frees the plan description

Parameters

- [in] description: plan description handle

```c
rocfft_status rocfft_plan_description_set_data_layout (rocfft_plan_description description, rocfft_array_type in_array_type, rocfft_array_type out_array_type, const size_t *in_offsets, const size_t *out_offsets, size_t in_strides_size, const size_t *in_strides, size_t in_distance, size_t out_strides_size, const size_t *out_strides, size_t out_distance)
```

Set data layout.

This is one of plan description functions to specify optional additional plan properties using the description handle. This API specifies the layout of buffers. This function can be used to specify input and output array types. Not all combinations of array types are supported and error code will be returned for unsupported cases. Additionally, input and output buffer offsets can be specified. The function can be used to specify custom layout of data, with the ability to specify stride between consecutive elements in all dimensions. Also, distance between transform data members can be specified. The library will choose appropriate defaults if offsets/strides are set to null ptr and/or distances set to 0.

Parameters

- [in] description: description handle
- [in] in_array_type: array type of input buffer
- [in] out_array_type: array type of output buffer
- [in] in_offsets: offsets, in element units, to start of data in input buffer
- [in] out_offsets: offsets, in element units, to start of data in output buffer
- [in] in_strides_size: size of in_strides array (must be equal to transform dimensions)
- [in] in_strides: array of strides, in each dimension, of input buffer; if set to null ptr library chooses defaults
- [in] in_distance: distance between start of each data instance in input buffer
• \[\text{in}] \ \text{out}\_strides\_size: \text{size of out}\_strides array (must be equal to transform dimensions)

• \[\text{in}] \ \text{out}\_strides: \text{array of strides, in each dimension, of output buffer; if set to null ptr library chooses defaults}

• \[\text{in}] \ \text{out}\_distance: \text{distance between start of each data instance in output buffer}

2.12.4.6.5 Execution

The following details the execution function. After a plan has been created, it can be used to compute a transform on specified data. Aspects of the execution can be controlled and any useful information returned to the user.

rocfft\_execute(const rocfft\_plan plan, void *in\_buffer[], void *out\_buffer[], rocfft\_execution\_info info)

Execute an FFT plan.

This API executes an FFT plan on buffers given by the user. If the transform is in-place, only the input buffer is needed and the output buffer parameter can be set to NULL. For not in-place transforms, output buffers have to be specified. Note that both input and output buffer are arrays of pointers, this is to facilitate passing planar buffers where real and imaginary parts are in 2 separate buffers. For the default interleaved format, just a unit sized array holding the pointer to input/output buffer need to be passed. The final parameter in this function is an execution\_info handle. This parameter serves as a way for the user to control execution, as well as for the library to pass any execution related information back to the user.

Parameters

• \[\text{in}] \ \text{plan}: \text{plan handle}

• \[\text{inout}] \ \text{in}\_buffer: \text{array (of size 1 for interleaved data, of size 2 for planar data) of input buffers}

• \[\text{inout}] \ \text{out}\_buffer: \text{array (of size 1 for interleaved data, of size 2 for planar data) of output buffers, can be nullptr for inplace result placement}

• \[\text{in}] \ \text{info}: \text{execution info handle created by rocfft\_execution\_info\_create}

2.12.4.6.6 Execution info

The execution api \text{rocfft\_execute()} takes a rocfft\_execution\_info parameter. This parameter needs to be created and setup by the user and passed to the execution api. The execution info handle encapsulates information such as execution mode, pointer to any work buffer etc. It can also hold information that are side effect of execution such as event objects. The following functions deal with managing execution info object. Note that the \text{set} functions below need to be called before execution and \text{get} functions after execution.

rocfft\_status rocfft\_execution\_info\_create(rocfft\_execution\_info *info)

Create execution info.

This API creates an execution info with which the user can control plan execution & retrieve execution information.

Parameters

• \[\text{out}] \ \text{info}: \text{execution info handle}

rocfft\_status rocfft\_execution\_info\_destroy(rocfft\_execution\_info info)

Destroy an execution info.

This API frees the execution info

Parameters
rocfft_execution_info_set_work_buffer(rocfft_execution_info info, void *work_buffer, size_t size_in_bytes)

Set work buffer in execution info.

This is one of the execution info functions to specify optional additional information to control execution. This API specifies work buffer needed. It has to be called before the call to rocfft_execute. When a non-zero value is obtained from rocfft_plan_get_work_buffer_size, that means the library needs a work buffer to compute the transform. In this case, the user has to allocate the work buffer and pass it to the library via this api.

Parameters

- [in] info: execution info handle
- [in] work_buffer: work buffer
- [in] size_in_bytes: size of work buffer in bytes

rocfft_execution_info_set_stream(rocfft_execution_info info, void *stream)

Set stream in execution info.

This is one of the execution info functions to specify optional additional information to control execution. This API specifies compute stream. It has to be called before the call to rocfft_execute. It is the underlying device queue/stream where the library computations would be inserted. The library assumes user has created such a stream in the program and merely assigns work to the stream.

Parameters

- [in] info: execution info handle
- [in] stream: underlying compute stream

2.12.4.6.7 Enumerations

This section provides all the enumerations used.

enum rocfft_status
rocfft status/error codes

Values:

enumerator rocfft_status_success
enumerator rocfft_status_failure
enumerator rocfft_status_invalid_arg_value
enumerator rocfft_status_invalid_dimensions
enumerator rocfft_status_invalid_array_type
enumerator rocfft_status_invalid_strides
enumerator rocfft_status_invalid_distance
enumerator rocfft_status_invalid_offset

enum rocfft_transform_type
Type of transform.

Values:

enumerator rocfft_transform_type_complex_forward
enumerator rocfft_transform_type_complex_inverse
enumerator rocfft_transform_type_real_forward
enumerator rocfft_transform_type_real_inverse

enum rocfft_precision
  Precision.
  Values:
  enumerator rocfft_precision_single
  enumerator rocfft_precision_double

enum rocfft_result_placement
  Result placement.
  Values:
  enumerator rocfft_placement_inplace
  enumerator rocfft_placement_notinplace

enum rocfft_array_type
  Array type.
  Values:
  enumerator rocfft_array_type_complex_interleaved
  enumerator rocfft_array_type_complex_planar
  enumerator rocfft_array_type_real
  enumerator rocfft_array_type_hermitian_interleaved
  enumerator rocfft_array_type_hermitian_planar

enum rocfft_execution_mode
  Execution mode.
  Values:
  enumerator rocfft_exec_mode_nonblocking
  enumerator rocfft_exec_mode_nonblocking_with_flush
  enumerator rocfft_exec_mode_blocking

2.12.5 rocSPARSE

2.12.5.1 Introduction

rocSPARSE is a library that contains basic linear algebra subroutines for sparse matrices and vectors written in HiP
for GPU devices. It is designed to be used from C and C++ code.

The functionality of rocSPARSE is organized in the following categories:

- **Sparse Auxiliary Functions** describe available helper functions that are required for subsequent library calls.
- **Sparse Level 1 Functions** describe operations between a vector in sparse format and a vector in dense format.
- **Sparse Level 2 Functions** describe operations between a matrix in sparse format and a vector in dense format.
• **Sparse Level 3 Functions** describe operations between a matrix in sparse format and multiple vectors in dense format.
• **Sparse Extra Functions** describe operations that manipulate sparse matrices.
• **Preconditioner Functions** describe manipulations on a matrix in sparse format to obtain a preconditioner.
• **Sparse Conversion Functions** describe operations on a matrix in sparse format to obtain a different matrix format.

The code is open and hosted here: https://github.com/ROCmSoftwarePlatform/rocSPARSE

### 2.12.5.2 Building and Installing

#### 2.12.5.2.1 Prerequisites

rocSPARSE requires a ROCm enabled platform, more information [here](https://www.amd.com/en).

#### 2.12.5.2.2 Installing pre-built packages

rocSPARSE can be installed from AMD ROCm repository. For detailed instructions on how to set up ROCm on different platforms, see the AMD ROCm Platform Installation Guide for Linux.

rocSPARSE can be installed on e.g. Ubuntu using

```
sudo apt-get update
sudo apt-get install rocsparse
```

Once installed, rocSPARSE can be used just like any other library with a C API. The header file will need to be included in the user code in order to make calls into rocSPARSE, and the rocSPARSE shared library will become link-time and run-time dependent for the user application.

#### 2.12.5.2.3 Building rocSPARSE from Open-Source repository

**Building rocSPARSE from source**

Building from source is not necessary, as rocSPARSE can be used after installing the pre-built packages as described above. If desired, the following instructions can be used to build rocSPARSE from source. Furthermore, the following compile-time dependencies must be met

- git
- CMake 3.5 or later
- AMD ROCm
- rocPRIM
- googletest (optional, for clients)
- libboost-program-options (optional, for clients)

**Download rocSPARSE**

The rocSPARSE source code is available at the [rocSPARSE github page](https://github.com/ROCmSoftwarePlatform/rocSPARSE). Download the master branch using:

```
git clone -b master https://github.com/ROCmSoftwarePlatform/rocSPARSE.git
cd rocSPARSE
```
Below are steps to build different packages of the library, including dependencies and clients. It is recommended to install rocSPARSE using the install.sh script.

### 2.12.5.2.4 Using install.sh to build dependencies with library

The following table lists common uses of install.sh to build dependencies + library.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./install.sh</td>
<td>Print help information.</td>
</tr>
<tr>
<td>./install.sh -h</td>
<td>Build dependencies and library in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.</td>
</tr>
<tr>
<td>./install.sh -d</td>
<td>Build library in your local directory. It is assumed dependencies are available.</td>
</tr>
<tr>
<td>./install.sh -i</td>
<td>Build library, then build and install rocSPARSE package in /opt/rocm/rocsparse. You will be prompted for sudo access. This will install for all users.</td>
</tr>
</tbody>
</table>

### 2.12.5.2.5 Using install.sh to build dependencies with library and clients

The client contains example code, unit tests and benchmarks. Common uses of install.sh to build them are listed in the table below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./install.sh</td>
<td>Print help information.</td>
</tr>
<tr>
<td>./install.sh -h</td>
<td>Build dependencies, library and client in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.</td>
</tr>
<tr>
<td>./install.sh -dc</td>
<td>Build library and client in your local directory. It is assumed dependencies are available.</td>
</tr>
<tr>
<td>./install.sh -c</td>
<td>Build library, dependencies and client, then build and install rocSPARSE package in /opt/rocm/rocsparse. You will be prompted for sudo access. This will install for all users.</td>
</tr>
<tr>
<td>./install.sh -idc</td>
<td>Build library and client, then build and install rocSPARSE package in opt/rocm/rocsparse. You will be prompted for sudo access. This will install for all users.</td>
</tr>
</tbody>
</table>
2.12.5.2.6 Using individual commands to build rocSPARSE

CMake 3.5 or later is required in order to build rocSPARSE. The rocSPARSE library contains both, host and device code, therefore the HCC compiler must be specified during cmake configuration process.

rocSPARSE can be built using the following commands:

```bash
# Create and change to build directory
mkdir -p build/release ; cd build/release

# Default install path is /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to adjust it
CXX=/opt/rocm/bin/hcc cmake ../..

# Compile rocSPARSE library
make -j$(nproc)

# Install rocSPARSE to /opt/rocm
sudo make install
```

Boost and GoogleTest is required in order to build rocSPARSE clients.

rocSPARSE with dependencies and client can be built using the following commands:

```bash
# Install boost on Ubuntu
sudo apt install libboost-program-options-dev

# Install googletest
mkdir -p build/release/deps ; cd build/release/deps
cmake ../../../deps
sudo make -j$(nproc) install

# Change to build directory
cd..

# Configure rocSPARSE
# Build options:
#   BUILD_CLIENTS_TESTS - build tests (OFF)
#   BUILD_CLIENTS_BENCHMARKS - build benchmarks (OFF)
#   BUILD_CLIENTS_SAMPLES - build examples (ON)
#   BUILD_VERBOSE - verbose output (OFF)
#   BUILD_SHARED_LIBS - build rocSPARSE as a shared library (ON)

# Default install path is /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to adjust it
CXX=/opt/rocm/bin/hcc cmake ../.. -DBUILD_CLIENTS_TESTS=ON 
   -DBUILD_CLIENTS_BENCHMARKS=ON 
   -DBUILD_CLIENTS_SAMPLES=ON 
   -DBUILD_VERBOSE=OFF 
   -DBUILD_SHARED_LIBS=ON

# Compile rocSPARSE library
make -j$(nproc)

# Install rocSPARSE to /opt/rocm
sudo make install
```
2.12.5.2.7 Common build problems

1. **Issue:** HIP (/opt/rocm/hip) was built using hcc 1.0.xxx-xxx-xxx-xxx, but you are using /opt/rocm/bin/hcc with version 1.0.yyy-yyy-yyy-yyy from hipcc (version mismatch). Please rebuild HIP including cmake or update HCC_HOME variable.

   **Solution:** Download HIP from github and use hcc to build from source and then use the built HIP instead of /opt/rocm/hip.

2. **Issue:** HCC RUNTIME ERROR: Failed to find compatible kernel

   **Solution:** Add the following to the cmake command when configuring: `-DCMAKE_CXX_FLAGS="-amdgpu-target=gfx803,gfx900,gfx906,gfx908"`

3. **Issue:** Could not find a package configuration file provided by “ROCM” with any of the following names:
   ROCMConfig.cmake | br | rocm-config.cmake

   **Solution:** Install ROCm cmake modules

2.12.5.2.8 Simple Test

You can test the installation by running one of the rocSPARSE examples, after successfully compiling the library with clients.

```bash
# Navigate to clients binary directory
$ cd rocSPARSE/build/release/clients/staging

# Execute rocSPARSE example
$ ./example_csrmv 1000
```

2.12.5.2.9 Supported Targets

Currently, rocSPARSE is supported under the following operating systems

- Ubuntu 16.04
- Ubuntu 18.04
- CentOS 7
- SLES 15

To compile and run rocSPARSE, AMD ROCm Platform is required.

The following HIP capable devices are currently supported

- gfx803 (e.g. Fiji)
- gfx900 (e.g. Vega10, MI25)
- gfx906 (e.g. Vega20, MI50, MI60)
- gfx908
2.12.5.3 Device and Stream Management

`hipSetDevice()` and `hipGetDevice()` are HIP device management APIs. They are NOT part of the rocSPARSE API.

2.12.5.3.1 Asynchronous Execution

All rocSPARSE library functions, unless otherwise stated, are non-blocking and executed asynchronously with respect to the host. They may return before the actual computation has finished. To force synchronization, `hipDeviceSynchronize()` or `hipStreamSynchronize()` can be used. This will ensure that all previously executed rocSPARSE functions on the device / this particular stream have completed.

2.12.5.3.2 HIP Device Management

Before a HIP kernel invocation, users need to call `hipSetDevice()` to set a device, e.g. device 1. If users do not explicitly call it, the system by default sets it as device 0. Unless users explicitly call `hipSetDevice()` to set to another device, their HIP kernels are always launched on device 0.

The above is a HIP (and CUDA) device management approach and has nothing to do with rocSPARSE. rocSPARSE honors the approach above and assumes users have already set the device before a rocSPARSE routine call.

Once users set the device, they create a handle with `rocsparse_create_handle()`. Subsequent rocSPARSE routines take this handle as an input parameter. rocSPARSE ONLY queries (by `hipGetDevice()`) the user’s device; rocSPARSE does NOT set the device for users. If rocSPARSE does not see a valid device, it returns an error message. It is the users’ responsibility to provide a valid device to rocSPARSE and ensure the device safety.

Users CANNOT switch devices between `rocsparse_create_handle()` and `rocsparse_destroy_handle()`. If users want to change device, they must destroy the current handle and create another rocSPARSE handle.

2.12.5.3.3 HIP Stream Management

HIP kernels are always launched in a queue (also known as stream).

If users do not explicitly specify a stream, the system provides a default stream, maintained by the system. Users cannot create or destroy the default stream. However, users can freely create new streams (with `hipStreamCreate()` and bind it to the rocSPARSE handle. HIP kernels are invoked in rocSPARSE routines. The rocSPARSE handle is always associated with a stream, and rocSPARSE passes its stream to the kernels inside the routine. One rocSPARSE routine only takes one stream in a single invocation. If users create a stream, they are responsible for destroying it.

2.12.5.3.4 Multiple Streams and Multiple Devices

If the system under test has multiple HIP devices, users can run multiple rocSPARSE handles concurrently, but can NOT run a single rocSPARSE handle on different discrete devices. Each handle is associated with a particular singular device, and a new handle should be created for each additional device.
2.12.5.4 Storage Formats

2.12.5.4.1 COO storage format

The Coordinate (COO) storage format represents a $m \times n$ matrix by

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m</code></td>
<td>number of rows (integer).</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of columns (integer).</td>
</tr>
<tr>
<td><code>nnz</code></td>
<td>number of non-zero elements (integer).</td>
</tr>
<tr>
<td><code>coo_val</code></td>
<td>array of $nnz$ elements containing the data (floating point).</td>
</tr>
<tr>
<td><code>coo_row_ind</code></td>
<td>array of $nnz$ elements containing the row indices (integer).</td>
</tr>
<tr>
<td><code>coo_col_ind</code></td>
<td>array of $nnz$ elements containing the column indices (integer).</td>
</tr>
</tbody>
</table>

The COO matrix is expected to be sorted by row indices and column indices per row. Furthermore, each pair of indices should appear only once. Consider the following $3 \times 5$ matrix and the corresponding COO structures, with $m = 3$, $n = 5$ and $nnz = 8$ using zero based indexing:

$$A = \begin{pmatrix} 1.0 & 2.0 & 0.0 & 3.0 & 0.0 \\ 0.0 & 4.0 & 5.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 7.0 & 8.0 \end{pmatrix}$$

where

- `$coo_val[8]$` = \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0\}
- `$coo_row_ind[8]$` = \{0, 0, 0, 1, 1, 2, 2\}
- `$coo_col_ind[8]$` = \{0, 1, 3, 1, 2, 0, 3, 4\}

2.12.5.4.2 CSR storage format

The Compressed Sparse Row (CSR) storage format represents a $m \times n$ matrix by

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m</code></td>
<td>number of rows (integer).</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of columns (integer).</td>
</tr>
<tr>
<td><code>nnz</code></td>
<td>number of non-zero elements (integer).</td>
</tr>
<tr>
<td><code>csr_val</code></td>
<td>array of $nnz$ elements containing the data (floating point).</td>
</tr>
<tr>
<td><code>csr_row_ptr</code></td>
<td>array of $m+1$ elements that point to the start of every row (integer).</td>
</tr>
<tr>
<td><code>csr_col_ind</code></td>
<td>array of $nnz$ elements containing the column indices (integer).</td>
</tr>
</tbody>
</table>

The CSR matrix is expected to be sorted by column indices within each row. Furthermore, each pair of indices should appear only once. Consider the following $3 \times 5$ matrix and the corresponding CSR structures, with $m = 3$, $n = 5$ and $nnz = 8$ using one based indexing:

$$A = \begin{pmatrix} 1.0 & 2.0 & 0.0 & 3.0 & 0.0 \\ 0.0 & 4.0 & 5.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 7.0 & 8.0 \end{pmatrix}$$

where

- `$csr_val[8]$` = \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0\}
- `$csr_row_ptr[4]$` = \{1, 4, 6, 9\}
- `$csr_col_ind[8]$` = \{1, 2, 4, 2, 3, 1, 4, 5\}
2.12.5.4.3 ELL storage format

The Ellpack-Itpack (ELL) storage format represents a $m \times n$ matrix by

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>number of rows (integer)</td>
</tr>
<tr>
<td>n</td>
<td>number of columns (integer)</td>
</tr>
<tr>
<td>ell_width</td>
<td>maximum number of non-zero elements per row (integer)</td>
</tr>
<tr>
<td>ell_val</td>
<td>array of $m \times \text{ell_width}$ elements containing the data (floating point)</td>
</tr>
<tr>
<td>ell_col_ind</td>
<td>array of $m \times \text{ell_width}$ elements containing the column indices (integer)</td>
</tr>
</tbody>
</table>

The ELL matrix is assumed to be stored in column-major format. Rows with less than $\text{ell_width}$ non-zero elements are padded with zeros ($\text{ell_val}$) and $-1$ ($\text{ell_col_ind}$). Consider the following $3 \times 5$ matrix and the corresponding ELL structures, with $m = 3$, $n = 5$ and

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>number of rows (integer)</td>
</tr>
<tr>
<td>n</td>
<td>number of columns (integer)</td>
</tr>
<tr>
<td>nnz</td>
<td>number of non-zero elements of the COO part (integer)</td>
</tr>
<tr>
<td>ell_width</td>
<td>maximum number of non-zero elements per row of the ELL part (integer)</td>
</tr>
<tr>
<td>ell_val</td>
<td>array of $m \times \text{ell_width}$ elements containing the ELL part data (floating point)</td>
</tr>
<tr>
<td>ell_col_ind</td>
<td>array of $m \times \text{ell_width}$ elements containing the ELL part column indices (integer)</td>
</tr>
<tr>
<td>coo_val</td>
<td>array of $\text{nnz}$ elements containing the COO part data (floating point)</td>
</tr>
<tr>
<td>coo_row_ind</td>
<td>array of $\text{nnz}$ elements containing the COO part row indices (integer)</td>
</tr>
<tr>
<td>coo_col_ind</td>
<td>array of $\text{nnz}$ elements containing the COO part column indices (integer)</td>
</tr>
</tbody>
</table>

The HYB format is a combination of the ELL and COO sparse matrix formats. Typically, the regular part of the matrix is stored in ELL storage format, and the irregular part of the matrix is stored in COO storage format. Three different partitioning schemes can be applied when converting a CSR matrix to a matrix in HYB storage format. For further details on the partitioning schemes, see rocsparse_hyb_partition_.

2.12.5.5 Types

2.12.5.5.1 rocsparse_handle

typedef struct _rocsparse_handle *rocsparse_handle

Handle to the rocSPARSE library context queue.

The rocSPARSE handle is a structure holding the rocSPARSE library context. It must be initialized using rocsparse_create_handle() and the returned handle must be passed to all subsequent library function calls. It should be destroyed at the end using rocsparse_destroy_handle().

2.12.5.5.2 rocsparse_mat_descr

typedef struct _rocsparse_mat_descr *rocsparse_mat_descr

Descriptor of the matrix.

The rocSPARSE matrix descriptor is a structure holding all properties of a matrix. It must be initialized using rocsparse_create_mat_descr() and the returned descriptor must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using rocsparse_destroy_mat_descr().

2.12. ROCm Libraries
2.12.5.5.3 rocsparse_mat_info

```c
typedef struct _rocsparse_mat_info *rocsparse_mat_info
```

Info structure to hold all matrix meta data.

The rocSPARSE matrix info is a structure holding all matrix information that is gathered during analysis routines. It must be initialized using `rocsparse_create_mat_info()` and the returned info structure must be passed to all subsequent library calls that require additional matrix information. It should be destroyed at the end using `rocsparse_destroy_mat_info()`.

2.12.5.5.4 rocsparse_hyb_mat

```c
typedef struct _rocsparse_hyb_mat *rocsparse_hyb_mat
```

HYB matrix storage format.

The rocSPARSE HYB matrix structure holds the HYB matrix. It must be initialized using `rocsparse_create_hyb_mat()` and the returned HYB matrix must be passed to all subsequent library calls that involve the matrix. It should be destroyed at the end using `rocsparse_destroy_hyb_mat()`.

2.12.5.5.5 rocsparse_action

```c
enum rocsparse_action
```

Specify where the operation is performed on.

The `rocsparse_action` indicates whether the operation is performed on the full matrix, or only on the sparsity pattern of the matrix.

Values:

- `enumerator rocsparse_action_symbolic = 0`
  - Operate only on indices.

- `enumerator rocsparse_action_numeric = 1`
  - Operate on data and indices.

2.12.5.5.6 rocsparse_hyb_partition

```c
enum rocsparse_hyb_partition
```

HYB matrix partitioning type.

The `rocsparse_hyb_partition` type indicates how the hybrid format partitioning between COO and ELL storage formats is performed.

Values:

- `enumerator rocsparse_hyb_partition_auto = 0`
  - automatically decide on ELL nnz per row.

- `enumerator rocsparse_hyb_partition_user = 1`
  - user given ELL nnz per row.

- `enumerator rocsparse_hyb_partition_max = 2`
  - max ELL nnz per row, no COO part.
2.12.5.5.7 rocsparse_index_base

enum rocsparse_index_base
   Specify the matrix index base.

   The rocsparse_index_base indicates the index base of the indices. For a given rocsparse_mat_descr, the rocsparse_index_base can be set using rocsparse_set_mat_index_base(). The current rocsparse_index_base of a matrix can be obtained by rocsparse_get_mat_index_base().

   Values:

   enumerator rocsparse_index_base_zero = 0
     zero based indexing.

   enumerator rocsparse_index_base_one = 1
     one based indexing.

2.12.5.5.8 rocsparse_matrix_type

enum rocsparse_matrix_type
   Specify the matrix type.

   The rocsparse_matrix_type indices the type of a matrix. For a given rocsparse_mat_descr, the rocsparse_matrix_type can be set using rocsparse_set_mat_type(). The current rocsparse_matrix_type of a matrix can be obtained by rocsparse_get_mat_type().

   Values:

   enumerator rocsparse_matrix_type_general = 0
     general matrix type.

   enumerator rocsparse_matrix_type_symmetric = 1
     symmetric matrix type.

   enumerator rocsparse_matrix_type_hermitian = 2
     hermitian matrix type.

   enumerator rocsparse_matrix_type_triangular = 3
     triangular matrix type.

2.12.5.5.9 rocsparse_fill_mode

enum rocsparse_fill_mode
   Specify the matrix fill mode.

   The rocsparse_fill_mode indicates whether the lower or the upper part is stored in a sparse triangular matrix. For a given rocsparse_mat_descr, the rocsparse_fill_mode can be set using rocsparse_set_mat_fill_mode(). The current rocsparse_fill_mode of a matrix can be obtained by rocsparse_get_mat_fill_mode().

   Values:

   enumerator rocsparse_fill_mode_lower = 0
     lower triangular part is stored.

   enumerator rocsparse_fill_mode_upper = 1
     upper triangular part is stored.
2.12.5.5.10 rocsparse_diag_type

enum rocsparse_diag_type
    Indicates if the diagonal entries are unity.

The rocsparse_diag_type indicates whether the diagonal entries of a matrix are unity or not. If rocsparse_diag_type_unit is specified, all present diagonal values will be ignored. For a given rocsparse_mat_descr, the rocsparse_diag_type can be set using rocsparse_set_mat_diag_type(). The current rocsparse_diag_type of a matrix can be obtained by rocsparse_get_mat_diag_type().

Values:

enumerator rocsparse_diag_type_non_unit = 0
    diagonal entries are non-unity.

enumerator rocsparse_diag_type_unit = 1
    diagonal entries are unity

2.12.5.5.11 rocsparse_operation

enum rocsparse_operation
    Specify whether the matrix is to be transposed or not.

The rocsparse_operation indicates the operation performed with the given matrix.

Values:

enumerator rocsparse_operation_none = 111
    Operate with matrix.

enumerator rocsparse_operation_transpose = 112
    Operate with transpose.

enumerator rocsparse_operation_conjugate_transpose = 113
    Operate with conj. transpose.

2.12.5.5.12 rocsparse_pointer_mode

enum rocsparse_pointer_mode
    Indicates if the pointer is device pointer or host pointer.

The rocsparse_pointer_mode indicates whether scalar values are passed by reference on the host or device. The rocsparse_pointer_mode can be changed by rocsparse_set_pointer_mode(). The currently used pointer mode can be obtained by rocsparse_get_pointer_mode().

Values:

enumerator rocsparse_pointer_mode_host = 0
    scalar pointers are in host memory.

enumerator rocsparse_pointer_mode_device = 1
    scalar pointers are in device memory.
2.12.5.5.13 rocsparse_analysis_policy

enum rocsparse_analysis_policy
Specify policy in analysis functions.

The rocsparse_analysis_policy specifies whether gathered analysis data should be re-used or not. If meta data from a previous e.g. rocsparse_csrilu0_analysis() call is available, it can be re-used for subsequent calls to e.g. rocsparse_csrsv_analysis() and greatly improve performance of the analysis function.

Values:

enumerator rocsparse_analysis_policy_reuse = 0
try to re-use meta data.

enumerator rocsparse_analysis_policy_force = 1
force to re-build meta data.

2.12.5.5.14 rocsparse_solve_policy

enum rocsparse_solve_policy
Specify policy in triangular solvers and factorizations.

This is a placeholder.

Values:

enumerator rocsparse_solve_policy_auto = 0
automatically decide on level information.

2.12.5.5.15 rocsparse_layer_mode

enum rocsparse_layer_mode
Indicates if layer is active with bitmask.

The rocsparse_layer_mode bit mask indicates the logging characteristics.

Values:

enumerator rocsparse_layer_mode_none = 0x0
layer is not active.

enumerator rocsparse_layer_mode_log_trace = 0x1
layer is in logging mode.

enumerator rocsparse_layer_mode_log_bench = 0x2
layer is in benchmarking mode.

2.12.5.5.16 rocsparse_status

enum rocsparse_status
List of rocsparse status codes definition.

This is a list of the rocsparse_status types that are used by the rocSPARSE library.

Values:

enumerator rocsparse_status_success = 0
success.
enumerator rocsparse_status_invalid_handle = 1
    handle not initialized, invalid or null.

enumerator rocsparse_status_not_implemented = 2
    function is not implemented.

enumerator rocsparse_status_invalid_pointer = 3
    invalid pointer parameter.

enumerator rocsparse_status_invalid_size = 4
    invalid size parameter.

enumerator rocsparse_status_memory_error = 5
    failed memory allocation, copy, dealloc.

enumerator rocsparse_status_internal_error = 6
    other internal library failure.

enumerator rocsparse_status_invalid_value = 7
    invalid value parameter.

enumerator rocsparse_status_arch_mismatch = 8
    device arch is not supported.

enumerator rocsparse_status_zero_pivot = 9
    encountered zero pivot.

2.12.5.6 Logging

Three different environment variables can be set to enable logging in rocSPARSE: ROCSPARSE_LAYER,
ROCSPARSE_LOG_TRACE_PATH and ROCSPARSE_LOG_BENCH_PATH.

ROCSPARSE_LAYER is a bit mask, where several logging modes can be combined as follows:

<table>
<thead>
<tr>
<th>ROCSPARSE_LAYER</th>
<th>Logging Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>unset</td>
<td>logging is disabled.</td>
</tr>
<tr>
<td>set to 1</td>
<td>trace logging is enabled.</td>
</tr>
<tr>
<td>set to 2</td>
<td>bench logging is enabled.</td>
</tr>
<tr>
<td>set to 3</td>
<td>trace logging and bench logging is enabled.</td>
</tr>
</tbody>
</table>

When logging is enabled, each rocSPARSE function call will write the function name as well as function arguments
to the logging stream. The default logging stream is stderr.

If the user sets the environment variable ROCSPARSE_LOG_TRACE_PATH to the full path name for a file, the file is
opened and trace logging is streamed to that file. If the user sets the environment variable
ROCSPARSE_LOG_BENCH_PATH to the full path name for a file, the file is opened and bench logging is streamed
to that file. If the file cannot be opened, logging output is stream to stderr.

Note that performance will degrade when logging is enabled. By default, the environment variable
ROCSPARSE_LAYER is unset and logging is disabled.
2.12.5.7 Sparse Auxiliary Functions

This module holds all sparse auxiliary functions.

The functions that are contained in the auxiliary module describe all available helper functions that are required for subsequent library calls.

### 2.12.5.7.1 rocsparse_create_handle()

```
rocsparse_status rocsparse_create_handle (rocsparse_handle *handle)
```

Create a rocsparse handle.

- `rocsparse_create_handle` creates the rocSPARSE library context. It must be initialized before any other rocSPARSE API function is invoked and must be passed to all subsequent library function calls. The handle should be destroyed at the end using `rocsparse_destroy_handle()`.

**Parameters**

- `[out] handle`: the pointer to the handle to the rocSPARSE library context.

**Return Value**

- `rocsparse_status_success`: the initialization succeeded.
- `rocsparse_status_invalid_handle`: handle pointer is invalid.
- `rocsparse_status_internal_error`: an internal error occurred.

### 2.12.5.7.2 rocsparse_destroy_handle()

```
rocsparse_status rocsparse_destroy_handle (rocsparse_handle handle)
```

Destroy a rocsparse handle.

- `rocsparse_destroy_handle` destroys the rocSPARSE library context and releases all resources used by the rocSPARSE library.

**Parameters**

- `[in] handle`: the handle to the rocSPARSE library context.

**Return Value**

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: handle is invalid.
- `rocsparse_status_internal_error`: an internal error occurred.
2.12.5.7.3 \texttt{rocsparse\_set\_stream()}

\texttt{rocsparse\_status rocsparse\_set\_stream}(\texttt{rocsparse\_handle handle, hipStream\_t stream})

Specify user defined HIP stream.

\texttt{rocsparse\_set\_stream} specifies the stream to be used by the rocSPARSE library context and all subsequent function calls.

\textbf{Example} This example illustrates, how a user defined stream can be used in rocSPARSE.

```c
// Create rocSPARSE handle
rocsparse\_handle handle;
rocsparse\_create\_handle(handle);

// Create stream
hipStream\_t stream;
hipStreamCreate(&stream);

// Set stream to rocSPARSE handle
rocsparse\_set\_stream(handle, stream);

// Do some work
// ...

// Clean up
rocsparse\_destroy\_handle(handle);
hipStreamDestroy(stream);
```

\textbf{Parameters}

- [inout] handle: the handle to the rocSPARSE library context.
- [in] stream: the stream to be used by the rocSPARSE library context.

\textbf{Return Value}

- rocsparse\_status\_success: the operation completed successfully.
- rocsparse\_status\_invalid\_handle: handle is invalid.

2.12.5.7.4 \texttt{rocsparse\_get\_stream()}

\texttt{rocsparse\_status rocsparse\_get\_stream}(\texttt{rocsparse\_handle handle, hipStream\_t \*stream})

Get current stream from library context.

\texttt{rocsparse\_get\_stream} gets the rocSPARSE library context stream which is currently used for all subsequent function calls.

\textbf{Parameters}

- [in] handle: the handle to the rocSPARSE library context.
- [out] stream: the stream currently used by the rocSPARSE library context.

\textbf{Return Value}

- rocsparse\_status\_success: the operation completed successfully.
- rocsparse\_status\_invalid\_handle: handle is invalid.
### 2.12.5.7.5 rocsparse_set_pointer_mode()

```c
rocsparse_status rocsparse_set_pointer_mode(rocsparse_handle handle, rocsparse_pointer_mode pointer_mode)
```

Specify pointer mode.

`rocsparse_set_pointer_mode` specifies the pointer mode to be used by the rocSPARSE library context and all subsequent function calls. By default, all values are passed by reference on the host. Valid pointer modes are `rocsparse_pointer_mode_host` or `rocsparse_pointer_mode_device`.

**Parameters**
- `[in]` `handle`: the handle to the rocSPARSE library context.
- `[in]` `pointer_mode`: the pointer mode to be used by the rocSPARSE library context.

**Return Value**
- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: handle is invalid.

### 2.12.5.7.6 rocsparse_get_pointer_mode()

```c
rocsparse_status rocsparse_get_pointer_mode(rocsparse_handle handle, rocsparse_pointer_mode *pointer_mode)
```

Get current pointer mode from library context.

`rocsparse_get_pointer_mode` gets the rocSPARSE library context pointer mode which is currently used for all subsequent function calls.

**Parameters**
- `[in]` `handle`: the handle to the rocSPARSE library context.
- `[out]` `pointer_mode`: the pointer mode that is currently used by the rocSPARSE library context.

**Return Value**
- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: handle is invalid.

### 2.12.5.7.7 rocsparse_get_version()

```c
rocsparse_status rocsparse_get_version(rocsparse_handle handle, int *version)
```

Get rocSPARSE version.

`rocsparse_get_version` gets the rocSPARSE library version number.

- `patch = version % 100`
- `minor = version / 100 % 1000`
- `major = version / 100000`

**Parameters**
- `[in]` `handle`: the handle to the rocSPARSE library context.
- `[out]` `version`: the version number of the rocSPARSE library.
Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: handle is invalid.

2.12.5.7.8 rocsparse_get_git_rev()

`rocsparse_status rocsparse_get_git_rev (rocsparse_handle handle, char *rev)`

Get rocSPARSE git revision.

`rocsparse_get_git_rev` gets the rocSPARSE library git commit revision (SHA-1).

Parameters

- `[in] handle`: the handle to the rocSPARSE library context.
- `[out] rev`: the git commit revision (SHA-1).

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: handle is invalid.

2.12.5.7.9 rocsparse_create_mat_descr()

`rocsparse_status rocsparse_create_mat_descr (rocsparse_mat_descr *descr)`

Create a matrix descriptor.

`rocsparse_create_mat_descr` creates a matrix descriptor. It initializes `rocsparse_matrix_type` to `rocsparse_matrix_type_general` and `rocsparse_index_base` to `rocsparse_index_base_zero`. It should be destroyed at the end using `rocsparse_destroy_mat_descr()`.

Parameters

- `[out] descr`: the pointer to the matrix descriptor.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_pointer`: descr pointer is invalid.

2.12.5.7.10 rocsparse_destroy_mat_descr()

`rocsparse_status rocsparse_destroy_mat_descr (rocsparse_mat_descr descr)`

Destroy a matrix descriptor.

`rocsparse_destroy_mat_descr` destroys a matrix descriptor and releases all resources used by the descriptor.

Parameters


Return Value

- `rocsparse_status_success`: the operation completed successfully.
• rocsparse_status_invalid_pointer: descr is invalid.

### 2.12.5.7.11 rocsparse_copy_mat_descr()

```c
rocsparse_status rocsparse_copy_mat_descr(rocsparse_mat_descr dest, const rocsparse_mat_descr src)
```

Copy a matrix descriptor.

rocsparse_copy_mat descr copies a matrix descriptor. Both, source and destination matrix descriptors must be initialized prior to calling rocsparse_copy_mat descr.

**Parameters**

- [out] dest: the pointer to the destination matrix descriptor.
- [in] src: the pointer to the source matrix descriptor.

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_pointer: src or dest pointer is invalid.

### 2.12.5.7.12 rocsparse_set_mat_index_base()

```c
rocsparse_status rocsparse_set_mat_index_base(rocsparse_mat_descr descr, rocsparse_index_base base)
```

Specify the index base of a matrix descriptor.

rocsparse_set_mat_index_base sets the index base of a matrix descriptor. Valid options are rocsparse_index_base_zero or rocsparse_index_base_one.

**Parameters**

- [inout] descr: the matrix descriptor.
- [in] base: rocsparse_index_base_zero or rocsparse_index_base_one.

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_pointer: descr pointer is invalid.
- rocsparse_status_invalid_value: base is invalid.

### 2.12.5.7.13 rocsparse_get_mat_index_base()

```c
rocsparse_index_base rocsparse_get_mat_index_base(const rocsparse_mat_descr descr)
```

Get the index base of a matrix descriptor.

rocsparse_get_mat_index_base returns the index base of a matrix descriptor.

**Return** rocsparse_index_base_zero or rocsparse_index_base_one.

**Parameters**

2.12.5.7.14 rocsparse_set_mat_type()

\texttt{rocsparse_status rocsparse_set_mat_type(rocsparse_mat_descr descr, rocsparse_matrix_type type)}

Specify the matrix type of a matrix descriptor.

\texttt{rocsparse_set_mat_type} sets the matrix type of a matrix descriptor. Valid matrix types are \texttt{rocsparse_matrix_type_general}, \texttt{rocsparse_matrix_type_symmetric}, \texttt{rocsparse_matrix_type_hermitian} or \texttt{rocsparse_matrix_type_triangular}.

\textbf{Parameters}

- [inout] \texttt{descr}: the matrix descriptor.
- [in] \texttt{type}: \texttt{rocsparse_matrix_type_general}, \texttt{rocsparse_matrix_type_symmetric}, \texttt{rocsparse_matrix_type_hermitian} or \texttt{rocsparse_matrix_type_triangular}.

\textbf{Return Value}

- \texttt{rocsparse_status_success}: the operation completed successfully.
- \texttt{rocsparse_status_invalid_pointer}: descr pointer is invalid.
- \texttt{rocsparse_status_invalid_value}: type is invalid.

2.12.5.7.15 rocsparse_get_mat_type()

\texttt{rocsparse_matrix_type rocsparse_get_mat_type(const rocsparse_mat_descr descr)}

Get the matrix type of a matrix descriptor.

\texttt{rocsparse_get_mat_type} returns the matrix type of a matrix descriptor.

\textbf{Return} \texttt{rocsparse_matrix_type_general}, \texttt{rocsparse_matrix_type_symmetric}, \texttt{rocsparse_matrix_type_hermitian} or \texttt{rocsparse_matrix_type_triangular}.

\textbf{Parameters}

- [in] \texttt{descr}: the matrix descriptor.

2.12.5.7.16 rocsparse_set_mat_fill_mode()

\texttt{rocsparse_status rocsparse_set_mat_fill_mode(rocsparse_mat_descr descr, rocsparse_fill_mode fill_mode)}

Specify the matrix fill mode of a matrix descriptor.

\texttt{rocsparse_set_mat_fill_mode} sets the matrix fill mode of a matrix descriptor. Valid fill modes are \texttt{rocsparse_fill_mode_lower} or \texttt{rocsparse_fill_mode_upper}.

\textbf{Parameters}

- [inout] \texttt{descr}: the matrix descriptor.
- [in] \texttt{fill_mode}: \texttt{rocsparse_fill_mode_lower} or \texttt{rocsparse_fill_mode_upper}.

\textbf{Return Value}

- \texttt{rocsparse_status_success}: the operation completed successfully.
- \texttt{rocsparse_status_invalid_pointer}: descr pointer is invalid.
- \texttt{rocsparse_status_invalid_value}: fill_mode is invalid.
2.12.5.7.17 rocsparse_get_mat_fill_mode()

rocsparse_fill_mode rocsparse_get_mat_fill_mode (const rocsparse_mat_descr descr)

Get the matrix fill mode of a matrix descriptor.

rocsparse_get_mat_fill_mode returns the matrix fill mode of a matrix descriptor.

Return rocsparse_fill_mode_lower or rocsparse_fill_mode_upper.

Parameters

• [in] descr: the matrix descriptor.

2.12.5.7.18 rocsparse_set_mat_diag_type()

rocsparse_status rocsparse_set_mat_diag_type (rocsparse_mat_descr descr, rocsparse_diag_type diag_type)

Specify the matrix diagonal type of a matrix descriptor.

rocsparse_set_mat_diag_type sets the matrix diagonal type of a matrix descriptor. Valid diagonal types are rocsparse_diag_type_unit or rocsparse_diag_type_non_unit.

Parameters

• [inout] descr: the matrix descriptor.
• [in] diag_type: rocsparse_diag_type_unit or rocsparse_diag_type_non_unit.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_pointer: descr pointer is invalid.
• rocsparse_status_invalid_value: diag_type is invalid.

2.12.5.7.19 rocsparse_get_mat_diag_type()

rocsparse_diag_type rocsparse_get_mat_diag_type (const rocsparse_mat_descr descr)

Get the matrix diagonal type of a matrix descriptor.

rocsparse_get_mat_diag_type returns the matrix diagonal type of a matrix descriptor.

Return rocsparse_diag_type_unit or rocsparse_diag_type_non_unit.

Parameters

• [in] descr: the matrix descriptor.
2.12.5.7.20 rocsparse_create_hyb_mat()

rocsparse_status rocsparse_create_hyb_mat (rocsparse_hyb_mat *hyb)
Create a HYB matrix structure.

rocsparse_create_hyb_mat creates a structure that holds the matrix in HYB storage format. It should be destroyed at the end using rocsparse_destroy_hyb_mat().

Parameters
  • [inout] hyb: the pointer to the hybrid matrix.

Return Value
  • rocsparse_status_success: the operation completed successfully.
  • rocsparse_status_invalid_pointer: hyb pointer is invalid.

2.12.5.7.21 rocsparse_destroy_hyb_mat()

rocsparse_status rocsparse_destroy_hyb_mat (rocsparse_hyb_mat hyb)
Destroy a HYB matrix structure.

rocsparse_destroy_hyb_mat destroys a HYB structure.

Parameters
  • [in] hyb: the hybrid matrix structure.

Return Value
  • rocsparse_status_success: the operation completed successfully.
  • rocsparse_status_invalid_pointer: hyb pointer is invalid.
  • rocsparse_status_internal_error: an internal error occurred.

2.12.5.7.22 rocsparse_create_mat_info()

rocsparse_status rocsparse_create_mat_info (rocsparse_mat_info *info)
Create a matrix info structure.

rocsparse_create_mat_info creates a structure that holds the matrix info data that is gathered during the analysis routines available. It should be destroyed at the end using rocsparse_destroy_mat_info().

Parameters
  • [inout] info: the pointer to the info structure.

Return Value
  • rocsparse_status_success: the operation completed successfully.
  • rocsparse_status_invalid_pointer: info pointer is invalid.
2.12.5.7.23 rocsparse_destroy_mat_info()

rocsparse_status rocsparse_destroy_mat_info(rocsparse_mat_info info)

Destroy a matrix info structure.

rocsparse_destroy_mat_info destroys a matrix info structure.

Parameters


Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_pointer: info pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.

2.12.5.8 Sparse Level 1 Functions

The sparse level 1 routines describe operations between a vector in sparse format and a vector in dense format. This section describes all rocSPARSE level 1 sparse linear algebra functions.

2.12.5.8.1 rocsparse_axpyi()

rocsparse_status rocsparse_saxpyi(rocsparse_handle handle, rocsparse_int nnz, const float *alpha, const float *x_val, const rocsparse_int *x_ind, float *y, rocsparse_index_base idx_base)

rocsparse_status rocsparse_daxpyi(rocsparse_handle handle, rocsparse_int nnz, const double *alpha, const double *x_val, const rocsparse_int *x_ind, double *y, rocsparse_index_base idx_base)

Scale a sparse vector and add it to a dense vector.

rocsparse_axpyi multiplies the sparse vector \( x \) with scalar \( \alpha \) and adds the result to the dense vector \( y \), such that

\[
y := y + \alpha \cdot x
\]

\[
\text{for}(i = 0; i < \text{nnz}; ++i) \\
\{ \\
\quad y[\text{x_ind}[i]] = y[\text{x_ind}[i]] + \text{alpha} \times \text{x_val}[i]; \\
\}
\]

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of vector \( x \).
- [in] alpha: scalar \( \alpha \).
- [in] x_val: array of \( \text{nnz} \) elements containing the values of \( x \).
• [in] \(x\_\text{ind}\): array of \(nnz\) elements containing the indices of the non-zero values of \(x\).
• [inout] \(y\): array of values in dense format.
• [in] \(\text{idx\_base}\): \text{rocsparse\_index\_base\_zero} or \text{rocsparse\_index\_base\_one}.

**Return Value**

- \text{rocsparse\_status\_success}: the operation completed successfully.
- \text{rocsparse\_status\_invalid\_handle}: the library context was not initialized.
- \text{rocsparse\_status\_invalid\_value}: \text{idx\_base} is invalid.
- \text{rocsparse\_status\_invalid\_size}: \(nnz\) is invalid.
- \text{rocsparse\_status\_invalid\_pointer}: \text{alpha}, \text{x\_val}, \text{x\_ind} or \(y\) pointer is invalid.

\text{rocsparse\_status rocsparse\_caxpyi(rocsparse\_handle handle, rocsparse\_int \(nnz\), const rocsparse\_float\_complex *alpha, const rocsparse\_float\_complex *x\_val, const rocsparse\_int *x\_ind, rocsparse\_float\_complex *y, rocsparse\_index\_base idx\_base)}

Scale a sparse vector and add it to a dense vector.

\text{rocsparse\_axpyi} multiplies the sparse vector \(x\) with scalar \(\alpha\) and adds the result to the dense vector \(y\), such that

\[
y := y + \alpha \cdot x
\]

```c
for(i = 0; i < nnz; ++i)
{
    y[x\_ind[i]] = y[x\_ind[i]] + alpha * x\_val[i];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- [in] \(\text{handle}\): handle to the rocsparse library context queue.
- [in] \(\text{nnz}\): number of non-zero entries of vector \(x\).
- [in] \(\text{alpha}\): scalar \(\alpha\).
- [in] \(\text{x\_val}\): array of \(nnz\) elements containing the values of \(x\).
- [in] \(\text{x\_ind}\): array of \(nnz\) elements containing the indices of the non-zero values of \(x\).
- [inout] \(\text{y}\): array of values in dense format.
- [in] \(\text{idx\_base}\): \text{rocsparse\_index\_base\_zero} or \text{rocsparse\_index\_base\_one}.

**Return Value**

- \text{rocsparse\_status\_success}: the operation completed successfully.
- \text{rocsparse\_status\_invalid\_handle}: the library context was not initialized.
- \text{rocsparse\_status\_invalid\_value}: \text{idx\_base} is invalid.
- \text{rocsparse\_status\_invalid\_size}: \(nnz\) is invalid.
rocsparse_zaxpyi (rocsparse_handle handle, rocsparse_int nnz, const rocsparse_double_complex *alpha, const rocsparse_double_complex *x_val, const rocsparse_int *x_ind, rocsparse_double_complex *y, rocsparse_index_base idx_base)

Scale a sparse vector and add it to a dense vector.

rocsparse_axpyi multiplies the sparse vector \( x \) with scalar \( \alpha \) and adds the result to the dense vector \( y \), such that

\[
y := y + \alpha \cdot x
\]

```c
for(i = 0; i < nnz; ++i)
{
    y[x_ind[i]] = y[x_ind[i]] + alpha * x_val[i];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of vector \( x \).
- [in] alpha: scalar \( \alpha \).
- [in] x_val: array of \( nnz \) elements containing the values of \( x \).
- [in] x_ind: array of \( nnz \) elements containing the indices of the non-zero values of \( x \).
- [inout] y: array of values in dense format.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_value: idx_base is invalid.
- rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: alpha, x_val, x_ind or y pointer is invalid.

### 2.12.5.8.2 rocsparse_doti() (rocsparse_sdoti, rocsparse_ddoti)

rocsparse_status rocsparse_sdoti (rocsparse_handle handle, rocsparse_int nnz, const float *x_val, const rocsparse_int *x_ind, const float *y, float *result, rocsparse_index_base idx_base)

rocsparse_status rocsparse_ddoti (rocsparse_handle handle, rocsparse_int nnz, const double *x_val, const rocsparse_int *x_ind, const double *y, double *result, rocsparse_index_base idx_base)

Compute the dot product of a sparse vector with a dense vector.
rocsparse_doti computes the dot product of the sparse vector $x$ with the dense vector $y$, such that

$$\text{result} := y^T x$$

```c
for(i = 0; i < nnz; ++i)
{
    result += x_val[i] * y[x_ind[i]];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** nnz: number of non-zero entries of vector $x$.
- **[in]** x_val: array of nnz values.
- **[in]** x_ind: array of nnz elements containing the indices of the non-zero values of $x$.
- **[in]** y: array of values in dense format.
- **[out]** result: pointer to the result, can be host or device memory
- **[in]** idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_value: idx_base is invalid.
- rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: x_val, x_ind, y or result pointer is invalid.
- rocsparse_status_memory_error: the buffer for the dot product reduction could not be allocated.
- rocsparse_status_internal_error: an internal error occurred.

```
rocsparse_status rocsparse_cdoti(rocsparse_handle handle, rocsparse_int nnz, const rocsparse_float_complex *x_val, const rocsparse_int *x_ind, const rocsparse_float_complex *y, rocsparse_float_complex *result, rocsparse_index_base idx_base)
```

Compute the dot product of a sparse vector with a dense vector.

rocsparse_doti computes the dot product of the sparse vector $x$ with the dense vector $y$, such that

$$\text{result} := y^T x$$

```c
for(i = 0; i < nnz; ++i)
{
    result += x_val[i] * y[x_ind[i]];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.
Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of vector $x$.
- [in] x_val: array of nnz values.
- [in] x_ind: array of nnz elements containing the indices of the non-zero values of $x$.
- [in] y: array of values in dense format.
- [out] result: pointer to the result, can be host or device memory
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_value: idx_base is invalid.
- rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: x_val, x_ind, y or result pointer is invalid.
- rocsparse_status_memory_error: the buffer for the dot product reduction could not be allocated.
- rocsparse_status_internal_error: an internal error occurred.

```
rocsparse_status rocsparse_zdoti(rocsparse_handle handle, rocsparse_int nnz, const rocsparse_double_complex *x_val, const rocsparse_int *x_ind, const rocsparse_double_complex *y, rocsparse_double_complex *result, rocsparse_index_base idx_base)
```

Compute the dot product of a sparse vector with a dense vector.

```
rocsparse_zdoti(compute the dot product of the sparse vector $x$ with the dense vector $y$, such that

result := $y^T x$
```

```
for(i = 0; i < nnz; ++i)
{
    result += x_val[i] * y[x_ind[i]];
}
```

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of vector $x$.
- [in] x_val: array of nnz values.
- [in] x_ind: array of nnz elements containing the indices of the non-zero values of $x$.
- [in] y: array of values in dense format.
- [out] result: pointer to the result, can be host or device memory
• [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_value: idx_base is invalid.
• rocsparse_status_invalid_size: nnz is invalid.
• rocsparse_status_invalid_pointer: x_val, x_ind, y or result pointer is invalid.
• rocsparse_status_memory_error: the buffer for the dot product reduction could not be allocated.
• rocsparse_status_internal_error: an internal error occurred.

2.12.5.8.3 rocsparse_dotci()

rocsparse_status rocsparse_cdotci(rocsparse_handle handle, rocsparse_int nnz, const rocsparse_float_complex *x_val, const rocsparse_int *x_ind, const rocsparse_float_complex *y, rocsparse_float_complex *result, rocsparse_index_base idx_base)

rocsparse_status rocsparse_zdotci(rocsparse_handle handle, rocsparse_int nnz, const rocsparse_double_complex *x_val, const rocsparse_int *x_ind, const rocsparse_double_complex *y, rocsparse_double_complex *result, rocsparse_index_base idx_base)

Compute the dot product of a complex conjugate sparse vector with a dense vector.

rocsparse_dotci computes the dot product of the complex conjugate sparse vector \( x \) with the dense vector \( y \), such that

\[
\text{result} := \bar{x}^H y
\]

```c
for(i = 0; i < nnz; ++i)
{
    result += conj(x_val[i]) * y[x_ind[i]];
}
```

Note  This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] nnz: number of non-zero entries of vector \( x \).
• [in] x_val: array of nnz values.
• [in] x_ind: array of nnz elements containing the indices of the non-zero values of \( x \).
• [in] y: array of values in dense format.
• [out] result: pointer to the result, can be host or device memory
• [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_value: idx_base is invalid.
• rocsparse_status_invalid_size: nnz is invalid.
• rocsparse_status_invalid_pointer: x_val, x_ind, y or result pointer is invalid.
• rocsparse_status_memory_error: the buffer for the dot product reduction could not be allocated.
• rocsparse_status_internal_error: an internal error occurred.

2.12.5.8.4 rocsparse_gthr()

rocsparse_status rocsparse_sgthr(rocsparse_handle handle, rocsparse_int nnz, const float *y, float *x_val, const rocsparse_int *x_ind, rocsparse_index_base idx_base)
rocsparse_status rocsparse_dgthr(rocsparse_handle handle, rocsparse_int nnz, const double *y, double *x_val, const rocsparse_int *x_ind, rocsparse_index_base idx_base)

Gather elements from a dense vector and store them into a sparse vector.

gothr gains the elements that are listed in x_ind from the dense vector y and stores them in the sparse vector x.

```c
for (i = 0; i < nnz; ++i)
{
    x_val[i] = y[x_ind[i]];
}
```

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] nnz: number of non-zero entries of x.
• [in] y: array of values in dense format.
• [out] x_val: array of nnz elements containing the values of x.
• [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
• [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_value: idx_base is invalid.
• rocsparse_status_invalid_size: nnz is invalid.
• rocsparse_status_invalid_pointer: y, x_val or x_ind pointer is invalid.
rocsparse_status rocsparse_cgthr (rocsparse_handle handle, rocsparse_int nnz, const rocsparse_float_complex*y, rocsparse_float_complex*x_val, const rocsparse_int*x_ind, rocsparse_index_base idx_base)

Gather elements from a dense vector and store them into a sparse vector.

rocsparse_gthr gathers the elements that are listed in x_ind from the dense vector y and stores them in the sparse vector x.

```c
for (i = 0; i < nnz; ++i) {
    x_val[i] = y[x_ind[i]];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of x.
- [in] y: array of values in dense format.
- [out] x_val: array of nnz elements containing the values of x.
- [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_value: idx_base is invalid.
- rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: y, x_val or x_ind pointer is invalid.

rocsparse_status rocsparse_zgthr (rocsparse_handle handle, rocsparse_int nnz, const rocsparse_double_complex*y, rocsparse_double_complex*x_val, const rocsparse_int*x_ind, rocsparse_index_base idx_base)

Gather elements from a dense vector and store them into a sparse vector.

rocsparse_gthr gathers the elements that are listed in x_ind from the dense vector y and stores them in the sparse vector x.

```c
for (i = 0; i < nnz; ++i) {
    x_val[i] = y[x_ind[i]];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of x.
• [in] y: array of values in dense format.
• [out] x_val: array of \( nnz \) elements containing the values of \( x \).
• [in] x_ind: array of \( nnz \) elements containing the indices of the non-zero values of \( x \).
• [in] idx_base: \texttt{rocsparse_index_base_zero} or \texttt{rocsparse_index_base_one}.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_value: \texttt{idx_base} is invalid.
• rocsparse_status_invalid_size: \( nnz \) is invalid.
• rocsparse_status_invalid_pointer: \( y, x_{-}val \) or \( x_{-}ind \) pointer is invalid.

2.12.5.8.5 rocsparse_gthrz()

\texttt{rocsparse_status rocsparse_sgthrz (rocsparse\_handle handle, rocsparse\_int nnz, float \*y, float \*x\_val,}
\texttt{ const rocsparse\_int \*x\_ind, rocsparse\_index\_base idx\_base)}

\texttt{rocsparse_status rocsparsedgthrz (rocsparse\_handle handle, rocsparse\_int nnz, double \*y, double \*x\_val, const rocsparse\_int \*x\_ind, rocsparse\_index\_base idx\_base)}

Gather and zero out elements from a dense vector and store them into a sparse vector.

\texttt{rocsparse\_gthrz} gathers the elements that are listed in \( x_{-}ind \) from the dense vector \( y \) and stores them in the sparse vector \( x \). The gathered elements in \( y \) are replaced by zero.

\begin{verbatim}
for(i = 0; i < nnz; ++i)
{
    x\_val[i] = y[x\_ind[i]];
    y[x\_ind[i]] = 0;
}
\end{verbatim}

\textbf{Note} This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] nnz: number of non-zero entries of \( x \).
• [inout] y: array of values in dense format.
• [out] x_val: array of \( nnz \) elements containing the non-zero values of \( x \).
• [in] x_ind: array of \( nnz \) elements containing the indices of the non-zero values of \( x \).
• [in] idx_base: \texttt{rocsparse\_index\_base\_zero} or \texttt{rocsparse\_index\_base\_one}.

Return Value

• rocsparse\_status\_success: the operation completed successfully.
• rocsparse\_status\_invalid\_handle: the library context was not initialized.
• rocsparse\_status\_invalid\_value: \texttt{idx\_base} is invalid.
• rocsparse\_status\_invalid\_size: \( nnz \) is invalid.
• rocspars_status_invalid_pointer: y, x_val or x_ind pointer is invalid.

rocspars_status rocsparse_cgthrz (rocspars_handle handle, rocspars_int nnz, rocspars_float_complex *y, rocspars_float_complex *x_val, const rocspars_int *x_ind, rocspars_index_base idx_base)

Gather and zero out elements from a dense vector and store them into a sparse vector.

rocspars_gthrz gathers the elements that are listed in x_ind from the dense vector y and stores them in the sparse vector x. The gathered elements in y are replaced by zero.

for (i = 0; i < nnz; ++i)
    {  
    x_val[i] = y[x_ind[i]];
    y[x_ind[i]] = 0;
    }

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

• [in] handle: handle to the rocspars library context queue.
• [in] nnz: number of non-zero entries of x.
• [inout] y: array of values in dense format.
• [out] x_val: array of nnz elements containing the non-zero values of x.
• [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
• [in] idx_base: rocspars_index_base_zero or rocspars_index_base_one.

Return Value

• rocspars_status_success: the operation completed successfully.
• rocspars_status_invalid_handle: the library context was not initialized.
• rocspars_status_invalid_value: idx_base is invalid.
• rocspars_status_invalid_size: nnz is invalid.
• rocspars_status_invalid_pointer: y, x_val or x_ind pointer is invalid.

rocspars_status rocsparse_zgthrz (rocspars_handle handle, rocspars_int nnz, rocspars_double_complex *y, rocspars_double_complex *x_val, const rocspars_int *x_ind, rocspars_index_base idx_base)

Gather and zero out elements from a dense vector and store them into a sparse vector.

rocspars_gthrz gathers the elements that are listed in x_ind from the dense vector y and stores them in the sparse vector x. The gathered elements in y are replaced by zero.

for (i = 0; i < nnz; ++i)
    {  
    x_val[i] = y[x_ind[i]];
    y[x_ind[i]] = 0;
    }

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.
Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of x.
- [inout] y: array of values in dense format.
- [out] x_val: array of nnz elements containing the non-zero values of x.
- [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_value: idx_base is invalid.
- rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: y, x_val or x_ind pointer is invalid.

2.12.5.8.6 rocsparse_roti()

rocsparse_status rocsparse_sroti (rocsparse_handle handle, rocsparse_int nnz, float *x_val, const rocsparse_int *x_ind, float *y, const float *c, const float *s, rocsparse_index_base idx_base)

rocsparse_status rocsparse_droti (rocsparse_handle handle, rocsparse_int nnz, double *x_val, const rocsparse_int *x_ind, double *y, const double *c, const double *s, rocsparse_index_base idx_base)

Apply Givens rotation to a dense and a sparse vector.

rocsparse_roti applies the Givens rotation matrix \( G \) to the sparse vector \( x \) and the dense vector \( y \), where

\[
G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}
\]

for(i = 0; i < nnz; ++i)
{
    x_tmp = x_val[i];
    y_tmp = y[x_ind[i]];
    x_val[i] = c * x_tmp + s * y_tmp;
    y[x_ind[i]] = c * y_tmp - s * x_tmp;
}

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of x.
- [inout] x_val: array of nnz elements containing the non-zero values of x.
- [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
• [inout] y: array of values in dense format.
• [in] c: pointer to the cosine element of G, can be on host or device.
• [in] s: pointer to the sine element of G, can be on host or device.
• [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_value: idx_base is invalid.
• rocsparse_status_invalid_size: nnz is invalid.
• rocsparse_status_invalid_pointer: c, s, x_val, x_ind or y pointer is invalid.

2.12.5.8.7 rocsparse_sctr()

rocsparse_status rocsparse_sctr(rocsparse_handle handle, rocsparse_int nnz, const float *x_val, const rocsparse_int *x_ind, float *y, rocsparse_index_base idx_base)

rocsparse_status rocsparse_dctr(rocsparse_handle handle, rocsparse_int nnz, const double *x_val, const rocsparse_int *x_ind, double *y, rocsparse_index_base idx_base)

Scatter elements from a dense vector across a sparse vector.

rocsparse_sctr scatters the elements that are listed in x_ind from the sparse vector x into the dense vector y. Indices of y that are not listed in x_ind remain unchanged.

```c
for(i = 0; i < nnz; ++i)
{
    y[x_ind[i]] = x_val[i];
}
```

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters
• [in] handle: handle to the rocsparse library context queue.
• [in] nnz: number of non-zero entries of x.
• [in] x_val: array of nnz elements containing the non-zero values of x.
• [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
• [inout] y: array of values in dense format.
• [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_value: idx_base is invalid.
• rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: x_val, x_ind or y pointer is invalid.

```c
rocsparse_status rocsparse_csctr (rocsparse_handle handle, rocsparse_int nnz, const rocsparse_float_complex *x_val, const rocsparse_int *x_ind, rocsparse_float_complex *y, rocsparse_index_base idx_base)
```

Scatter elements from a dense vector across a sparse vector.

rocsparse_scctr scatters the elements that are listed in x_ind from the sparse vector x into the dense vector y. Indices of y that are not listed in x_ind remain unchanged.

```c
for (i = 0; i < nnz; ++i)
{
    y[x_ind[i]] = x_val[i];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
- [in] nnz: number of non-zero entries of x.
- [in] x_val: array of nnz elements containing the non-zero values of x.
- [in] x_ind: array of nnz elements containing the indices of the non-zero values of x.
- [inout] y: array of values in dense format.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_value: idx_base is invalid.
- rocsparse_status_invalid_size: nnz is invalid.
- rocsparse_status_invalid_pointer: x_val, x_ind or y pointer is invalid.

```c
rocsparse_status rocsparse_zsctr (rocsparse_handle handle, rocsparse_int nnz, const rocsparse_double_complex *x_val, const rocsparse_int *x_ind, rocsparse_double_complex *y, rocsparse_index_base idx_base)
```

Scatter elements from a dense vector across a sparse vector.

rocsparse_scctr scatters the elements that are listed in x_ind from the sparse vector x into the dense vector y. Indices of y that are not listed in x_ind remain unchanged.

```c
for (i = 0; i < nnz; ++i)
{
    y[x_ind[i]] = x_val[i];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
• [in] \text{nnz}: number of non-zero entries of \( x \).
• [in] \text{x_val}: array of \text{nnz} elements containing the non-zero values of \( x \).
• [in] \text{x_ind}: array of \text{nnz} elements containing the indices of the non-zero values of \( x \).
• [inout] \text{y}: array of values in dense format.
• [in] \text{idx_base}: \text{rocsparse_index_base_zero} or \text{rocsparse_index_base_one}.

Return Value

• \text{rocsparse_status_success}: the operation completed successfully.
• \text{rocsparse_status_invalid_handle}: the library context was not initialized.
• \text{rocsparse_status_invalid_value}: \text{idx_base} is invalid.
• \text{rocsparse_status_invalid_size}: \text{nnz} is invalid.
• \text{rocsparse_status_invalid_pointer}: \text{x_val}, \text{x_ind} or \text{y} pointer is invalid.

2.12.5.9 Sparse Level 2 Functions

This module holds all sparse level 2 routines.

The sparse level 2 routines describe operations between a matrix in sparse format and a vector in dense format.

2.12.5.9.1 \text{rocsparse_coomv()}

\text{rocsparse_status rocsparse_scoomv}(\text{rocsparse_handle handle}, \text{rocsparse_operation trans}, \text{rocsparse_int m}, \text{rocsparse_int n}, \text{rocsparse_int nnz}, \text{const float *alpha, const rocsparse_mat_descr descr, const float *coo_val, const rocsparse_int *coo_row_ind, const rocsparse_int *coo_col_ind, const float *x, const float *beta, float *y})

\text{rocsparse_status rocsparse_dcoomv}(\text{rocsparse_handle handle}, \text{rocsparse_operation trans}, \text{rocsparse_int m}, \text{rocsparse_int n}, \text{rocsparse_int nnz}, \text{const double *alpha, const rocsparse_mat_descr descr, const double *coo_val, const rocsparse_int *coo_row_ind, const rocsparse_int *coo_col_ind, const double *x, const double *beta, double *y})

Sparse matrix vector multiplication using COO storage format.

\text{rocsparse_coomv} multiplies the scalar \( \alpha \) with a sparse \( m \times n \) matrix, defined in COO storage format, and the dense vector \( x \) and adds the result to the dense vector \( y \) that is multiplied by the scalar \( \beta \), such that

\[
y := \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y,
\]

with

\[
\text{op}(A) = \begin{cases} 
A, & \text{if trans} = \text{rocsparse_operation_none} \\
A^T, & \text{if trans} = \text{rocsparse_operation_transpose} \\
A^H, & \text{if trans} = \text{rocsparse_operation_conjugate_transpose}
\end{cases}
\]

The COO matrix has to be sorted by row indices. This can be achieved by using \text{rocsparse_coosort_by_row}().


```c
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];
}

for(i = 0; i < nnz; ++i)
{
    y[coo_row_ind[i]] += alpha * coo_val[i] * x[coo_col_ind[i]];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only \texttt{trans == rocsparse_operation_none} is supported.

**Parameters**

- [in] \texttt{handle}: handle to the rocsparse library context queue.
- [in] \texttt{trans}: matrix operation type.
- [in] \texttt{m}: number of rows of the sparse COO matrix.
- [in] \texttt{n}: number of columns of the sparse COO matrix.
- [in] \texttt{nnz}: number of non-zero entries of the sparse COO matrix.
- [in] \texttt{alpha}: scalar \( \alpha \).
- [in] \texttt{descr}: descriptor of the sparse COO matrix. Currently, only \texttt{rocsparse_matrix_type_general} is supported.
- [in] \texttt{coo_val}: array of \texttt{nnz} elements of the sparse COO matrix.
- [in] \texttt{coo_row_ind}: array of \texttt{nnz} elements containing the row indices of the sparse COO matrix.
- [in] \texttt{coo_col_ind}: array of \texttt{nnz} elements containing the column indices of the sparse COO matrix.
- [in] \texttt{x}: array of \texttt{n} elements (\texttt{op}(A) = A) or \texttt{m} elements (\texttt{op}(A) = A^T \text{ or } \texttt{op}(A) = A^H).
- [in] \texttt{beta}: scalar \( \beta \).
- [inout] \texttt{y}: array of \texttt{m} elements (\texttt{op}(A) = A) or \texttt{n} elements (\texttt{op}(A) = A^T \text{ or } \texttt{op}(A) = A^H).

**Return Value**

- \texttt{rocsparse_status_success}: the operation completed successfully.
- \texttt{rocsparse_status_invalid_handle}: the library context was not initialized.
- \texttt{rocsparse_status_invalid_size}: \texttt{m}, \texttt{n} or \texttt{nnz} is invalid.
- \texttt{rocsparse_status_invalid_pointer}: \texttt{descr}, \texttt{alpha}, \texttt{coo_val}, \texttt{coo_row_ind}, \texttt{coo_col_ind}, \texttt{x}, \texttt{beta} or \texttt{y} pointer is invalid.
- \texttt{rocsparse_status_arch_mismatch}: the device is not supported.
- \texttt{rocsparse_status_not_implemented}: \texttt{trans != rocsparse_operation_none} or \texttt{rocsparse_matrix_type != rocsparse_matrix_type_general}.
Sparse matrix vector multiplication using COO storage format.

`rocsparse_coomv` multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in COO storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$ y := \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y, $$

with

$$ \text{op}(A) = \begin{cases} 
  A, & \text{if trans == rocsparse_operation_none} \\
  A^T, & \text{if trans == rocsparse_operationTranspose} \\
  A^H, & \text{if trans == rocsparse_operation_conjugate_transpose}
\end{cases} $$

The COO matrix has to be sorted by row indices. This can be achieved by using `rocsparse_coosort_by_row()`.

```c
for (i = 0; i < m; ++i) 
{ 
  y[i] = beta * y[i]; 
}
for (i = 0; i < nnz; ++i) 
{ 
  y[coo_row_ind[i]] += alpha * coo_val[i] * x[coo_col_ind[i]]; 
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only `trans == rocsparse_operation_none` is supported.

**Parameters**

- `[in]` `handle`: handle to the rocsparse library context queue.
- `[in]` `m`: number of rows of the sparse COO matrix.
- `[in]` `n`: number of columns of the sparse COO matrix.
- `[in]` `nnz`: number of non-zero entries of the sparse COO matrix.
- `[in]` `alpha`: scalar $\alpha$.
- `[in]` `descr`: descriptor of the sparse COO matrix. Currently, only `rocsparse_matrix_type_general` is supported.
- `[in]` `coo_row_ind`: array of `nnz` elements containing the row indices of the sparse COO matrix.
- `[in]` `coo_col_ind`: array of `nnz` elements containing the column indices of the sparse COO matrix.
- `[in]` `x`: array of $n$ elements ($\text{op}(A) = A$) or $m$ elements ($\text{op}(A) = A^T$ or $\text{op}(A) = A^H$).
• [in] beta: scalar $\beta$.
• [inout] y: array of $m$ elements ($op(A) = A$) or $n$ elements ($op(A) = A^T$ or $op(A) = A^H$).

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: $m$, $n$ or $nnz$ is invalid.
- rocsparse_status_invalid_pointer: $descr$, $alpha$, $coo\_val$, $coo\_row\_ind$, $coo\_col\_ind$, $x$, $beta\ or\ y$ pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_not_implemented: $trans \neq rocsparse\_operation\_none \ or \ rocsparse\_matrix\_type \neq rocsparse\_matrix\_type\_general$.

```c
rocsparse_status rocsparse_zcoomv(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_double_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_double_complex *coo_val, const rocsparse_int *coo_row_ind, const rocsparse_int *coo_col_ind, const rocsparse_double_complex *x, const rocsparse_double_complex *beta, rocsparse_double_complex *y)
```

Sparse matrix vector multiplication using COO storage format.

$rocsparse\_coomv$ multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in COO storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} 
A, & \text{if } trans = \text{rocsparse\_operation\_none} \\
A^T, & \text{if } trans = \text{rocsparse\_operation\_transpose} \\
A^H, & \text{if } trans = \text{rocsparse\_operation\_conjugate\_transpose} 
\end{cases}$$

The COO matrix has to be sorted by row indices. This can be achieved by using $rocsparse\_coosort\_by\_row()$.

```c
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];
}
for(i = 0; i < nnz; ++i)
{
    y[coo_row_ind[i]] += alpha * coo_val[i] * x[coo_col_ind[i]];
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only $trans = \text{rocsparse\_operation\_none}$ is supported.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
[in] m: number of rows of the sparse COO matrix.
[in] n: number of columns of the sparse COO matrix.
[in] alpha: scalar $\alpha$.
[in] descr: descriptor of the sparse COO matrix. Currently, only rocsparse_matrix_type_general is supported.
[in] coo_row_ind: array of nnz elements containing the row indices of the sparse COO matrix.
[in] coo_col_ind: array of nnz elements containing the column indices of the sparse COO matrix.
[in] x: array of n elements ($\text{op}(A) = A$) or m elements ($\text{op}(A) = A^T$ or $\text{op}(A) = A^H$).
[in] beta: scalar $\beta$.
[inout] y: array of m elements ($\text{op}(A) = A$) or n elements ($\text{op}(A) = A^T$ or $\text{op}(A) = A^H$).

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m, n or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, alpha, coo_val, coo_row_ind, coo_col_ind, x, beta or y pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

### 2.12.5.9.2 rocsparse_csrmv_analysis()

```
rocsparse_status rocsparse_scsrmv_analysis (rocsparse_handle handle, rocsparse_operation trans,
rocsparse_int m, rocsparse_int n, rocsparse_int nnz,
const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const
rocsparse_int *csr_col_ind, rocsparse_mat_info info)
```

Sparse matrix vector multiplication using CSR storage format.

**rocsparse_csrmv_analysis** performs the analysis step for rocsparse_scsrmv(), rocsparse_dcsrmv(), rocsparse_ccsrmv() and rocsparse_zcsrmv(). It is expected that this function will be executed only once for a given matrix and particular operation type. The gathered analysis meta data can be cleared by rocsparse_csrmv_clear().

**Note** If the matrix sparsity pattern changes, the gathered information will become invalid.
**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- **[in]** `handle`: handle to the rocspase library context queue.
- **[in]** `trans`: matrix operation type.
- **[in]** `m`: number of rows of the sparse CSR matrix.
- **[in]** `n`: number of columns of the sparse CSR matrix.
- **[in]** `nnz`: number of non-zero entries of the sparse CSR matrix.
- **[in]** `descr`: descriptor of the sparse CSR matrix.
- **[in]** `csr_val`: array of `nnz` elements of the sparse CSR matrix.
- **[in]** `csr_row_ptr`: array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
- **[in]** `csr_col_ind`: array of `nnz` elements containing the column indices of the sparse CSR matrix.
- **[out]** `info`: structure that holds the information collected during the analysis step.

**Return Value**

- `rocspase_status_success`: the operation completed successfully.
- `rocspase_status_invalid_handle`: the library context was not initialized.
- `rocspase_status_invalid_size`: `m`, `n` or `nnz` is invalid.
- `rocspase_status_invalid_pointer`: `descr`, `csr_val`, `csr_row_ptr`, `csr_col_ind` or `info` pointer is invalid.
- `rocspase_status_memory_error`: the buffer for the gathered information could not be allocated.
- `rocspase_status_internal_error`: an internal error occurred.
- `rocspase_status_not_implemented`: `trans` != `rocspase_operation_none` or `rocspase_matrix_type` != `rocspase_matrix_type_general`.

```c
rocspase_status rocspase_ccsrmv_analysis(rocsparse_handle handle, rocsparse_operation trans,
rocsparse_int m, rocsparse_int n, rocsparse_int nnz,
const rocsparse_mat_descr descr, const rocsparse_float_complex *csr_val,
const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind,
rocsparse_mat_info info)
```

Sparse matrix vector multiplication using CSR storage format.

`rocspase_ccsrmv_analysis` performs the analysis step for `rocspase_scsrmv()`, `rocspase_dcsrmv()`, `rocspase_ccsrmv()` and `rocspase_zcsrmv()`. It is expected that this function will be executed only once for a given matrix and particular operation type. The gathered analysis meta data can be cleared by `rocspase_ccsrmv_clear()`.

**Note** If the matrix sparsity pattern changes, the gathered information will become invalid.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**
• [in] handle: handle to the rocsparse library context queue.
• [in] trans: matrix operation type.
• [in] m: number of rows of the sparse CSR matrix.
• [in] n: number of columns of the sparse CSR matrix.
• [in] nnz: number of non-zero entries of the sparse CSR matrix.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
• [out] info: structure that holds the information collected during the analysis step.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n or nnz is invalid.
• rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind or info pointer is invalid.
• rocsparse_status_memory_error: the buffer for the gathered information could not be allocated.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_zcsrmv_analysis (rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info)

Sparse matrix vector multiplication using CSR storage format.

rocsparse_csrsv_analysis performs the analysis step for rocsparse_scsrmv(), rocsparse_dcsrmv(), rocsparse_ccsrmv() and rocsparse_zcsrmv(). It is expected that this function will be executed only once for a given matrix and particular operation type. The gathered analysis meta data can be cleared by rocsparse_csrsv_clear().

Note If the matrix sparsity pattern changes, the gathered information will become invalid.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] trans: matrix operation type.
• [in] m: number of rows of the sparse CSR matrix.
- [in] \( n \): number of columns of the sparse CSR matrix.
- [in] \( \text{nnz} \): number of non-zero entries of the sparse CSR matrix.
- [in] csr_val: array of \( \text{nnz} \) elements of the sparse CSR matrix.
- [in] csr_row_ptr: array of \( m+1 \) elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of \( \text{nnz} \) elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.

**Return Value**
- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: \( m \), \( n \) or \( \text{nnz} \) is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind or info pointer is invalid.
- rocsparse_status_memory_error: the buffer for the gathered information could not be allocated.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans \(!=\) rocsparse_operation_none or rocsparse_matrix_type \(!=\) rocsparse_matrix_type_general.

### 2.12.5.9.3 rocsparse_csrmv()

```c
rocsparse_status rocsparse_scsrmv(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const float *alpha, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const float *x, const float *beta, float *y)
```

```c
rocsparse_status rocsparse_dcsrmv(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const double *alpha, const rocsparse_mat_descr descr, const double *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const double *x, const double *beta, double *y)
```

Sparse matrix vector multiplication using CSR storage format.

rocsparse_csrmv multiplies the scalar \( \alpha \) with a sparse \( m \times n \) matrix, defined in CSR storage format, and the dense vector \( x \) and adds the result to the dense vector \( y \) that is multiplied by the scalar \( \beta \), such that

\[
y := \alpha \cdot op(A) \cdot x + \beta \cdot y,
\]

with

\[
op(A) = \begin{cases} 
    A, & \text{if trans} == \text{rocsparse_operation_none} \\
    A^T, & \text{if trans} == \text{rocsparse_operation_transpose} \\
    A^H, & \text{if trans} == \text{rocsparse_operation_conjugate_transpose}
\end{cases}
\]

2.12. ROCm Libraries
The `info` parameter is optional and contains information collected by `rocsparse_scsrmv_analysis()`, `rocsparse_dcsrmv_analysis()`, `rocsparse_ccsrmv_analysis()` or `rocsparse_zcsrmv_analysis()`. If present, the information will be used to speed up the `csrmv` computation. If `info == NULL`, general `csrmv` routine will be used instead.

```c
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];
    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only `trans == rocsparse_operation_none` is supported.

**Example** This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

```c
// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(&info);

// Perform analysis step to obtain meta data
rocsparse_scsrmv_analysis(handle,
    rocsparse_operation_none,
    m,
    n,
    nnz,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info);

// Compute y = Ax
rocsparse_scsrmv(handle,
    rocsparse_operation_none,
    m,
    n,
    nnz,
    &alpha,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    x,
    &beta,
    y);

// Do more work
// ...
```

(continues on next page)
Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] alpha: scalar $\alpha$.
- [in] descr: descriptor of the sparse CSR matrix. Currently, only `rocsparse_matrix_type_general` is supported.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [in] info: information collected by `rocsparse_scsrmv_analysis()`, `rocsparse_dcsrmv_analysis()`, `rocsparse_ccsrmv_analysis()` or `rocsparse_dcsrmv_analysis()`, can be NULL if no information is available.
- [in] x: array of n elements ($op(A) == A$) or m elements ($op(A) == A^T$ or $op(A) == A^H$).
- [in] beta: scalar $\beta$.
- [inout] y: array of m elements ($op(A) == A$) or n elements ($op(A) == A^T$ or $op(A) == A^H$).

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m, n or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x, beta or y pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

```
rocsparse_status rocsparse_ccsrmv(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_float_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const rocsparse_float_complex *x, const rocsparse_float_complex *beta, rocsparse_float_complex *y)
```

Sparse matrix vector multiplication using CSR storage format.
rocsparse_csrmm multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in CSR storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if } \text{trans} == \text{rocsparse\_operation\_none} \\ A^T, & \text{if } \text{trans} == \text{rocsparse\_operation\_transpose} \\ A^H, & \text{if } \text{trans} == \text{rocsparse\_operation\_conjugate\_transpose} \end{cases}$$

The info parameter is optional and contains information collected by rocsparse_scsrmv_analysis(), rocsparse_dcsrmv_analysis(), rocsparse_ccsrmv_analysis() or rocsparse_zcsrmv_analysis(). If present, the information will be used to speed up the csrmm computation. If info == NULL, general csrmm routine will be used instead.

```c
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];
    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only trans == rocsparse_operation_none is supported.

**Example** This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

```c
// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(&info);

// Perform analysis step to obtain meta data
rocsparse_scsrmv_analysis(handle,
    rocsparse_operation_none,
    m, n, nnz, descr, csr_val, csr_row_ptr, csr_col_ind, info);

// Compute y = Ax
rocsparse_scsrmv(handle,
    rocsparse_operation_none,
    m, n, nnz, &alpha,
```
```c
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    x,
    &beta,
    y);

// Do more work
// ...

// Clean up
rocsparse_destroy_mat_info(info);
```

### Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** trans: matrix operation type.
- **[in]** m: number of rows of the sparse CSR matrix.
- **[in]** n: number of columns of the sparse CSR matrix.
- **[in]** nnz: number of non-zero entries of the sparse CSR matrix.
- **[in]** alpha: scalar $\alpha$.
- **[in]** descr: descriptor of the sparse CSR matrix. Currently, only `rocsparse_matrix_type_general` is supported.
- **[in]** csr_val: array of nnz elements of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[in]** csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- **[in]** info: information collected by `rocsparse_scsrmv_analysis()`, `rocsparse_dcsrmv_analysis()`, `rocsparse_ccsrmv_analysis()` or `rocsparse_dcsrmv_analysis()`, can be NULL if no information is available.
- **[in]** x: array of n elements ($op(A) == A$) or m elements ($op(A) == A^T$ or $op(A) == A^H$).
- **[in]** beta: scalar $\beta$.
- **[inout]** y: array of m elements ($op(A) == A$) or n elements ($op(A) == A^T$ or $op(A) == A^H$).

### Return Value

- **rocsparse_status_success**: the operation completed successfully.
- **rocsparse_status_invalid_handle**: the library context was not initialized.
- **rocsparse_status_invalid_size**: m, n or nnz is invalid.
- **rocsparse_status_invalid_pointer**: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x, beta or y pointer is invalid.
- **rocsparse_status_arch_mismatch**: the device is not supported.
rocsparse_status rocsparse_zcsrmv(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_double_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const rocsparse_double_complex *x, const rocsparse_double_complex *beta, rocsparse_double_complex *y)

Sparse matrix vector multiplication using CSR storage format.

rocsparse_zcsrmv multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in CSR storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & \text{if } \text{trans} == \text{rocsparse_operation_none} \\ A^T, & \text{if } \text{trans} == \text{rocsparse_operation_transpose} \\ A^H, & \text{if } \text{trans} == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

The info parameter is optional and contains information collected by rocsparse_scsrmv_analysis(), rocsparse_dcsrmv_analysis(), rocsparse_ccsrmv_analysis() or rocsparse_zcsrmv_analysis(). If present, the information will be used to speed up the csrvm computation. If info == NULL, general csrvm routine will be used instead.

```c
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];
    for(j = csr_row_ptr[i]; j < csr_row_ptr[i + 1]; ++j)
    {
        y[i] = y[i] + alpha * csr_val[j] * x[csr_col_ind[j]];
    }
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only trans == rocsparse_operation_none is supported.

**Example** This example performs a sparse matrix vector multiplication in CSR format using additional meta data to improve performance.

```c
// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(&info);

// Perform analysis step to obtain meta data
rocsparse_scsrmv_analysis(handle,
    rocsparse_operation_none,
    m,
    n,
    nnz,
    (continues on next page)```
// Compute y = Ax
rocsparse_scsrmv(handle,
    rocsparse_operation_none,
    m,
    n,
    nnz,
    &alpha,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    x,
    &beta,
    y);

// Do more work
// ...

// Clean up
rocsparse_destroy_mat_info(info);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [in] info: information collected by rocsparse_scsrmv_analysis(), rocsparse_dcsrmv_analysis(),
  rocsparse_cscsmv_analysis() or rocsparse_dcsrmv_analysis(), can be NULL if no information is available.
- [in] x: array of n elements ( op(A) == A) or m elements ( op(A) == Aᵀ or op(A) == Aᴴ).
- [inout] y: array of m elements ( op(A) == A) or n elements ( op(A) == Aᵀ or op(A) == Aᴴ).
Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m, n or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x, beta or y pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.9.4 rocsparse_csrsv_analysis_clear()

rocsparse_status rocsparse_csrsv_clear (rocsparse_handle handle, rocsparse_mat_info info)

Sparse matrix vector multiplication using CSR storage format.

rocsparse_csrsv_clear deallocates all memory that was allocated by rocsparse_scsrmv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_cssrmv_analysis() or rocsparse_zcsmv_analysis(). This is especially useful, if memory is an issue and the analysis data is not required anymore for further computation, e.g. when switching to another sparse matrix format.

Note Calling rocsparse_csrsv_clear is optional. All allocated resources will be cleared, when the opaque rocsparse_mat_info struct is destroyed using rocsparse_destroy_mat_info().

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [inout] info: structure that holds the information collected during analysis step.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_pointer: info pointer is invalid.
- rocsparse_status_memory_error: the buffer for the gathered information could not be deallocated.
- rocsparse_status_internal_error: an internal error occurred.

2.12.5.9.5 rocsparse_ellmv()

rocsparse_status rocsparse_ellmv (rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, const float *alpha, const rocsparse_mat_descr descr, const float *ell_val, const rocsparse_int *ell_col_ind, rocsparse_int ell_width, const float *x, const float *beta, float *y)
**rocsparse_dellmv**

```c
rocsparse_status rocsparse_dellmv(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, const double *alpha, const rocsparse_mat_descr descr, const double *ell_val, const rocsparse_int *ell_col_ind, rocsparse_int ell_width, const double *x, const double *beta, double *y)
```

Sparse matrix vector multiplication using ELL storage format.

`rocsparse_ellmv` multiplies the scalar \( \alpha \) with a sparse \( m \times n \) matrix, defined in ELL storage format, and the dense vector \( x \) and adds the result to the dense vector \( y \) that is multiplied by the scalar \( \beta \), such that

\[
y := \alpha \cdot \text{op}(A) \cdot x + \beta \cdot y,
\]

with

\[
\text{op}(A) = \begin{cases} 
A, & \text{if } \text{trans} == \text{rocsparse_operation_none} \\
A^T, & \text{if } \text{trans} == \text{rocsparse_operation_transpose} \\
A^H, & \text{if } \text{trans} == \text{rocsparse_operation_conjugate_transpose}
\end{cases}
\]

```c
def for (i = 0; i < m; ++i)
{
    y[i] = beta * y[i];
    for (p = 0; p < ell_width; ++p)
    {
        idx = p * m + i;
        if ((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n))
        {
            y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
        }
    }
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only \( \text{trans} == \text{rocsparse_operation_none} \) is supported.

**Parameters**

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** trans: matrix operation type.
- **[in]** m: number of rows of the sparse ELL matrix.
- **[in]** n: number of columns of the sparse ELL matrix.
- **[in]** alpha: scalar \( \alpha \).
- **[in]** descr: descriptor of the sparse ELL matrix. Currently, only \text{rocsparse_matrix_type_general} is supported.
- **[in]** ell_val: array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
- **[in]** ell_col_ind: array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
- **[in]** ell_width: number of non-zero elements per row of the sparse ELL matrix.
- **[in]** x: array of \( n \) elements ( \( \text{op}(A) == A \) ) or \( m \) elements ( \( \text{op}(A) == A^T \) or \( \text{op}(A) == A^H \) ).
• [in] beta: scalar $\beta$.
• [inout] $y$: array of $m$ elements ($op(A) == A$) or $n$ elements ($op(A) == A^T$ or $op(A) == A^H$).

**Return Value**

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: $m$, $n$ or $\text{ell}_\text{width}$ is invalid.
- rocsparse_status_invalid_pointer: descr, alpha, $\text{ell}_\text{val}$, $\text{ell}_\text{col}_\text{ind}$, $x$, beta or $y$ pointer is invalid.
- rocsparse_status_not_implemented: $trans \neq \text{rocsparse_operation_none}$ or rocsparse_matrix_type != rocsparse_matrix_type_general.

```c
rocsparse_status rocsparse_ellmv (rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int n, const rocsparse_float_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_float_complex *ell_val, const rocsparse_int *ell_col_ind, rocsparse_int ell_width, const rocsparse_float_complex *x, const rocsparse_float_complex *beta, rocsparse_float_complex *y)
```

Sparse matrix vector multiplication using ELL storage format.

$rocsparse_ellmv$ multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in ELL storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} 
A, & \text{if } trans == \text{rocsparse_operation_none} \\
A^T, & \text{if } trans == \text{rocsparse_operation_transpose} \\
A^H, & \text{if } trans == \text{rocsparse_operation_conjugate_transpose}
\end{cases}$$

```c
for(i = 0; i < m; ++i)
{
    y[i] = beta * y[i];

    for(p = 0; p < ell_width; ++p)
    {
        idx = p * m + i;

        if((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n))
        {
            y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
        }
    }
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only $trans == \text{rocsparse_operation_none}$ is supported.

**Parameters**

- [in] handle: handle to the rocsparse library context queue.
• [in] trans: matrix operation type.
• [in] m: number of rows of the sparse ELL matrix.
• [in] n: number of columns of the sparse ELL matrix.
• [in] alpha: scalar $\alpha$.
• [in] descr: descriptor of the sparse ELL matrix. Currently, only rocsparse_matrix_type_general is supported.
• [in] ell_val: array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
• [in] ell_col_ind: array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
• [in] ell_width: number of non-zero elements per row of the sparse ELL matrix.
• [in] x: array of n elements ($op(A) == A$) or m elements ($op(A) == A^T$ or $op(A) == A^H$).
• [in] beta: scalar $\beta$.
• [inout] y: array of m elements ($op(A) == A$) or n elements ($op(A) == A^T$ or $op(A) == A^H$).

**Return Value**

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n or ell_width is invalid.
• rocsparse_status_invalid_pointer: descr, alpha, ell_val, ell_col_ind, x, beta or y pointer is invalid.
• rocsparse_status_not_implemented: trans $\neq$ rocsparse_operation_none or rocsparse_matrix_type $\neq$ rocsparse_matrix_type_general.

```c
rocsparsity status rocsparse_zellmv (rocsparsity handle handle, rocsparse operation trans, rocsparse_int m, rocsparse_int n, const rocsparse double complex *alpha, const rocsparse mat descr descr, const rocsparse double complex *ell_val, const rocsparse_int *ell_col_ind, rocsparse_int ell_width, const rocsparse double complex *x, const rocsparse double complex *beta, rocsparse double complex *y)
```

Sparse matrix vector multiplication using ELL storage format.

**rocsparse_zellmv** multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in ELL storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$ y := \alpha \cdot op(A) \cdot x + \beta \cdot y, $$

with

$$ op(A) = \begin{cases} 
A, & \text{if } trans == \text{rocsparse_operation_none} \\
A^T, & \text{if } trans == \text{rocsparse_operation_transpose} \\
A^H, & \text{if } trans == \text{rocsparse_operation_conjugate_transpose}
\end{cases} $$

```c
for (i = 0; i < m; ++i) 
{
    y[i] = beta * y[i];
}
```

(continues on next page)
for (p = 0; p < ell_width; ++p) {
    idx = p * m + i;
    if ((ell_col_ind[idx] >= 0) && (ell_col_ind[idx] < n)) {
        y[i] = y[i] + alpha * ell_val[idx] * x[ell_col_ind[idx]];
    }
}

Note  This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note  Currently, only trans == rocsparse_operation_none is supported.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse ELL matrix.
- [in] n: number of columns of the sparse ELL matrix.
- [in] alpha: scalar $\alpha$.
- [in] descr: descriptor of the sparse ELL matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] ell_val: array that contains the elements of the sparse ELL matrix. Padded elements should be zero.
- [in] ell_col_ind: array that contains the column indices of the sparse ELL matrix. Padded column indices should be -1.
- [in] ell_width: number of non-zero elements per row of the sparse ELL matrix.
- [in] x: array of $n$ elements ($op(A) == A$) or $m$ elements ($op(A) == A^T$ or $op(A) == A^H$).
- [in] beta: scalar $\beta$.
- [inout] y: array of $m$ elements ($op(A) == A$) or $n$ elements ($op(A) == A^T$ or $op(A) == A^H$).

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: $m$, $n$ or ell_width is invalid.
- rocsparse_status_invalid_pointer: descr, alpha, ell_val, ell_col_ind, x, beta or y pointer is invalid.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.
2.12.5.9.6 rocsparse_hybmv()

rocsparse_status rocsparse_shybmv(rocsparse_handle handle, rocsparse_operation trans, const float *alpha, const rocsparse_mat_descr descr, const rocsparse_hyb_mat hyb, const float *x, const float *beta, float *y)

rocsparse_status rocsparse_dhybmv(rocsparse_handle handle, rocsparse_operation trans, const double *alpha, const rocsparse_mat_descr descr, const rocsparse_hyb_mat hyb, const double *x, const double *beta, double *y)

Sparse matrix vector multiplication using HYB storage format.

rocsparse_hybmv multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in HYB storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$ y := \alpha \cdot op(A) \cdot x + \beta \cdot y, $$

with

$$ op(A) = \begin{cases} A, & \text{if } trans == \text{rocsparse_operation_none} \\ A^T, & \text{if } trans == \text{rocsparse_operation_transpose} \\ A^H, & \text{if } trans == \text{rocsparse_operation_conjugate_transpose} \end{cases} $$

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note Currently, only $trans == \text{rocsparse_operation_none}$ is supported.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] alpha: scalar $\alpha$.
- [in] descr: descriptor of the sparse HYB matrix. Currently, only \text{rocsparse_matrix_type_general} is supported.
- [in] hyb: matrix in HYB storage format.
- [in] x: array of $n$ elements ($op(A) == A$) or $m$ elements ($op(A) == A^T$ or $op(A) == A^H$).
- [in] beta: scalar $\beta$.
- [inout] y: array of $m$ elements ($op(A) == A$) or $n$ elements ($op(A) == A^T$ or $op(A) == A^H$).

Return Value

- \text{rocsparse_status_success}: the operation completed successfully.
- \text{rocsparse_status_invalid_handle}: the library context was not initialized.
- \text{rocsparse_status_invalid_size}: hyb structure was not initialized with valid matrix sizes.
- \text{rocsparse_status_invalid_pointer}: descr, alpha, hyb, x, beta or y pointer is invalid.
- \text{rocsparse_status_invalid_value}: hyb structure was not initialized with a valid partitioning type.
- \text{rocsparse_status_arch_mismatch}: the device is not supported.
- \text{rocsparse_status_memory_error}: the buffer could not be allocated.
rocsparse_status rocsparse_chybmv(rocsparse_handle handle, rocsparse_operation trans, const rocsparse_float_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_hyb_mat hyb, const rocsparse_float_complex *x, const rocsparse_float_complex *beta, rocsparse_float_complex *y)

Sparse matrix vector multiplication using HYB storage format.

rocsparse_hybmv multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in HYB storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} 
A, & \text{if } trans == \text{rocsparse_operation_none} \\
A^T, & \text{if } trans == \text{rocsparse_operation_transpose} \\
A^H, & \text{if } trans == \text{rocsparse_operation_conjugate_transpose}
\end{cases}$$

Note This function is non-blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note Currently, only $trans == \text{rocsparse_operation_none}$ is supported.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] alpha: scalar $\alpha$.
- [in] descr: descriptor of the sparse HYB matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] hyb: matrix in HYB storage format.
- [in] x: array of $n$ elements ($op(A) == A$) or $m$ elements ($op(A) == A^T$ or $op(A) == A^H$).
- [in] beta: scalar $\beta$.
- [inout] y: array of $m$ elements ($op(A) == A$) or $n$ elements ($op(A) == A^T$ or $op(A) == A^H$).

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: hyb structure was not initialized with valid matrix sizes.
- rocsparse_status_invalid_pointer: descr, alpha, hyb, x, beta or y pointer is invalid.
- rocsparse_status_invalid_value: hyb structure was not initialized with a valid partitioning type.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_memory_error: the buffer could not be allocated.
rocsparse_status rocsparse_zhybmv (rocsparse_handler handle, rocsparse_operation trans, const rocsparse_double_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_hyb_mat hyb, const rocsparse_double_complex *x, const rocsparse_double_complex *beta, rocsparse_double_complex *y)

Sparse matrix vector multiplication using HYB storage format.

rocsparse_zhybmv multiplies the scalar $\alpha$ with a sparse $m \times n$ matrix, defined in HYB storage format, and the dense vector $x$ and adds the result to the dense vector $y$ that is multiplied by the scalar $\beta$, such that

$$ y := \alpha \cdot op(A) \cdot x + \beta \cdot y, $$

with

$$ op(A) = \begin{cases} A, & \text{if trans == rocsparse_operation_none} \\ A^T, & \text{if trans == rocsparse_operation_transpose} \\ A^H, & \text{if trans == rocsparse_operation_conjugate_transpose} \end{cases} $$

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note Currently, only $\text{trans} == \text{rocsparse_operation_none}$ is supported.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] alpha: scalar $\alpha$.
- [in] descr: descriptor of the sparse HYB matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] hyb: matrix in HYB storage format.
- [in] x: array of $n$ elements ($op(A) == A$) or $m$ elements ($op(A) == A^T$ or $op(A) == A^H$).
- [in] beta: scalar $\beta$.
- [inout] y: array of $m$ elements ($op(A) == A$) or $n$ elements ($op(A) == A^T$ or $op(A) == A^H$).

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: hyb structure was not initialized with valid matrix sizes.
- rocsparse_status_invalid_pointer: descr, alpha, hyb, x, beta or y pointer is invalid.
- rocsparse_status_invalid_value: hyb structure was not initialized with a valid partitioning type.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_memory_error: the buffer could not be allocated.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: \( \text{trans} \neq \text{rocsparse_operation_none} \) or \( \text{rocsparse_matrix_type} \neq \text{rocsparse_matrix_type_general} \).

2.12.5.9.7 rocsparse_csrsv_zero_pivot()

\[ \text{rocsparse_status rocsparse_csrsv_zero_pivot} \left( \text{rocsparse_handle handle, const rocsparse_mat_descr descr, rocsparse_mat_info info, rocsparse_int *position} \right) \]

Sparse triangular solve using CSR storage format. \( \text{rocsparse_csrsv_zero_pivot} \) returns \( \text{rocsparse_status_zero_pivot} \), if either a structural or numerical zero has been found during \( \text{rocsparse_scsrsv_solve()} \), \( \text{rocsparse_dcsrsv_solve()} \), \( \text{rocsparse_ccsrsv_solve()} \) or \( \text{rocsparse_zcsrsv_solve()} \) computation. The first zero pivot \( j \) at \( A_{j,j} \) is stored in \( \text{position} \), using same index base as the CSR matrix.

\( \text{position} \) can be in host or device memory. If no zero pivot has been found, \( \text{position} \) is set to -1 and \( \text{rocsparse_status_success} \) is returned instead.

**Note** \( \text{rocsparse_csrsv_zero_pivot} \) is a blocking function. It might influence performance negatively.

**Parameters**

- [in] \( \text{handle} \): handle to the rocsparse library context queue.
- [in] \( \text{descr} \): descriptor of the sparse CSR matrix.
- [in] \( \text{info} \): structure that holds the information collected during the analysis step.
- [inout] \( \text{position} \): pointer to zero pivot \( j \), can be in host or device memory.

**Return Value**

- \( \text{rocsparse_status_success} \): the operation completed successfully.
- \( \text{rocsparse_status_invalid_handle} \): the library context was not initialized.
- \( \text{rocsparse_status_invalid_pointer} \): \( \text{info} \) or \( \text{position} \) pointer is invalid.
- \( \text{rocsparse_status_internal_error} \): an internal error occurred.
- \( \text{rocsparse_status_zero_pivot} \): zero pivot has been found.

2.12.5.9.8 rocsparse_csrsv_buffer_size()

\[ \text{rocsparse_status rocsparse_scsrsv_buffer_size} \left( \text{rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, size_t *buffer_size} \right) \]

\[ \text{rocsparse_status rocsparse_dcsrsv_buffer_size} \left( \text{rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const double *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, size_t *buffer_size} \right) \]

Sparse triangular solve using CSR storage format.
rocsparse_csrsv_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_scsrsv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_ccsrsv_analysis(), rocsparse_zcsrsv_analysis(), rocsparse_scsrsv_solve(), rocsparse_dcsrsv_solve(), rocsparse_ccsrsv_solve() and rocsparse_zcsrsv_solve(). The temporary storage buffer must be allocated by the user. The size of the temporary storage buffer is identical to the size returned by rocsparse_scsrilu0_buffer_size(), rocsparse_dcsrilu0_buffer_size(), rocsparse_ccsrilu0_buffer_size() and rocsparse_zcsrilu0_buffer_size() if the matrix sparsity pattern is identical. The user allocated buffer can thus be shared between subsequent calls to those functions.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.
- [in] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_scsrsv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_ccsrsv_analysis(), rocsparse_zcsrsv_analysis(), rocsparse_scsrsv_solve(), rocsparse_dcsrsv_solve(), rocsparse_ccsrsv_solve() and rocsparse_zcsrsv_solve().

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind, info or buffer_size pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_ccsrsv_buffer_size(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_desc descr, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, size_t *buffer_size)

Sparse triangular solve using CSR storage format.

2.12. ROCm Libraries 313
rocsparse_csrsv_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_scsrsv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_ccsrsv_analysis(), rocsparse_zcsrsv_analysis(), rocsparse_scsrsv_solve(), rocsparse_dcsrsv_solve(), rocsparse_ccsrsv_solve() and rocsparse_zcsrsv_solve(). The temporary storage buffer must be allocated by the user. The size of the temporary storage buffer is identical to the size returned by rocsparse_scsrilu0_buffer_size(), rocsparse_dcsrilu0_buffer_size(), rocsparse_ccsrilu0_buffer_size() and rocsparse_zcsrilu0_buffer_size() if the matrix sparsity pattern is identical. The user allocated buffer can thus be shared between subsequent calls to those functions.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.
- [in] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_scsrsv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_ccsrsv_analysis(), rocsparse_zcsrsv_analysis(), rocsparse_scsrsv_solve(), rocsparse_dcsrsv_solve(), rocsparse_ccsrsv_solve() and rocsparse_zcsrsv_solve().

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind, info or buffer_size pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

Sparse triangular solve using CSR storage format.
The function `rocsparse_csrsv_buffer_size` returns the size of the temporary storage buffer that is required by various CSR-based solve functions, including `rocsparse_scsrsv_analysis()`, `rocsparse_dcsrsv_analysis()`, `rocsparse_ccsrsv_analysis()`, `rocsparse_zcsrsv_analysis()`, `rocsparse_scsrsv_solve()`, `rocsparse_dcsrsv_solve()`, `rocsparse_ccsrsv_solve()`, and `rocsparse_zcsrsv_solve()`. The temporary storage buffer must be allocated by the user. The size of the temporary storage buffer is identical to the size returned by `rocsparse_scsrilu0_buffer_size()`, `rocsparse_dcsrilu0_buffer_size()`, `rocsparse_ccsrilu0_buffer_size()`, and `rocsparse_zcsrilu0_buffer_size()` if the matrix sparsity pattern is identical. The user allocated buffer can thus be shared between subsequent calls to those functions.

**Parameters**

- `[in] handle`: handle to the rocsparse library context queue.
- `[in] m`: number of rows of the sparse CSR matrix.
- `[in] nnz`: number of non-zero entries of the sparse CSR matrix.
- `[in] csr_row_ptr`: array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
- `[in] csr_col_ind`: array of `nnz` elements containing the column indices of the sparse CSR matrix.
- `[out] info`: structure that holds the information collected during the analysis step.
- `[in] buffer_size`: number of bytes of the temporary storage buffer required by various CSR-based solve functions.

**Return Value**

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: `m` or `nnz` is invalid.
- `rocsparse_status_invalid_pointer`: `descr`, `csr_val`, `csr_row_ptr`, `csr_col_ind`, `info`, or `buffer_size` pointer is invalid.
- `rocsparse_status_internal_error`: an internal error occurred.
- `rocsparse_status_not_implemented`: `trans` != `rocsparse_operation_none` or `rocsparse_matrix_type` != `rocsparse_matrix_type_general`. 
2.12.5.9.9 rocsparse_csrsv_analysis()

rocsparse_status rocsparse_csrsv_analysis(
    rocsparse_handle handle,
    rocsparse_operation trans,
    rocsparse_int m, rocsparse_int nnz,
    const rocsparse_mat_descr descr,
    const float *csr_val,
    const rocsparse_int *csr_row_ptr,
    const rocsparse_int *csr_col_ind,
    rocsparse_mat_info info,
    rocsparse_analysis_policy analysis,
    rocsparse_solve_policy solve,
    void *temp_buffer)

rocsparse_status rocsparse_dcsrsv_analysis(
    rocsparse_handle handle,
    rocsparse_operation trans,
    rocsparse_int m, rocsparse_int nnz,
    const rocsparse_mat_descr descr,
    const double *csr_val,
    const rocsparse_int *csr_row_ptr,
    const rocsparse_int *csr_col_ind,
    rocsparse_mat_info info,
    rocsparse_analysis_policy analysis,
    rocsparse_solve_policy solve,
    void *temp_buffer)

Sparse triangular solve using CSR storage format.

rocsparse_csrsv_analysis performs the analysis step for rocsparse_scsrsv_solve(), rocsparse_dcsrsv_solve(), rocsparse_ccsrsv_solve() and rocsparse_zcsrsv_solve(). It is expected that this function will be executed only once for a given matrix and particular operation type. The analysis meta data can be cleared by rocsparse_csrsv_clear().

rocsparse_csrsv_analysis can share its meta data with rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis() and rocsparse_zcsrilu0_analysis(). Selecting rocsparse_analysis_policy_force policy can greatly improve computation performance of meta data. However, the user need to make sure that the sparsity pattern remains unchanged. If this cannot be assured, rocsparse_analysis_policy_force has to be used.

Note If the matrix sparsity pattern changes, the gathered information will become invalid.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.
- [in] analysis: rocsparse_analysis_policy_reuse or rocsparse_analysis_policy_force.
- [in] temp_buffer: temporary storage buffer allocated by the user.
Return Value

- `rocspars_status_success`: the operation completed successfully.
- `rocspars_status_invalid_handle`: the library context was not initialized.
- `rocspars_status_invalid_size`: `m` or `nnz` is invalid.
- `rocspars_status_invalid_pointer`: `descr`, `csr_row_ptr`, `csr_col_ind`, `info` or `temp_buffer` pointer is invalid.
- `rocspars_status_internal_error`: an internal error occurred.
- `rocspars_status_not_implemented`: `trans` != `rocspars_operation_none` or `rocspars_matrix_type` != `rocspars_matrix_type_general`.

```c
rocspars_status rocspars_ccsrsrv_analysis(
    rocspars_handle handle,
    rocspars_operation trans,
    rocspars_int m,
    rocspars_int nnz,
    const rocspars_mat_descr descr,
    const rocspars_float_complex *csr_val,
    const rocspars_int *csr_row_ptr,
    const rocspars_int *csr_col_ind,
    rocspars_mat_info info,
    rocspars_analysis_policy analysis,
    rocspars_solve_policy solve,
    void *temp_buffer)
```

Sparse triangular solve using CSR storage format.

`rocspars_ccsrsrv_analysis` performs the analysis step for `rocspars_scsrsrv_solve()`, `rocspars_dcsrsrv_solve()`, `rocspars_ccsrsrv_solve()` and `rocspars_zcsrsrv_solve()`. It is expected that this function will be executed only once for a given matrix and particular operation type. The analysis meta data can be cleared by `rocspars_csrsv_clear()`.

`rocspars_ccsrsrv_analysis` can share its meta data with `rocspars_scsrilu0_analysis()`, `rocspars_dcsrilu0_analysis()`, `rocspars_ccsrilu0_analysis()` and `rocspars_zcsrilu0_analysis()`. Selecting `rocspars_analysis_policy_reuse` policy can greatly improve computation performance of meta data. However, the user need to make sure that the sparsity pattern remains unchanged. If this cannot be assured, `rocspars_analysis_policy_force` has to be used.

**Note** If the matrix sparsity pattern changes, the gathered information will become invalid.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- `in` `handle`: handle to the rocsparse library context queue.
- `in` `trans`: matrix operation type.
- `in` `m`: number of rows of the sparse CSR matrix.
- `in` `nnz`: number of non-zero entries of the sparse CSR matrix.
- `in` `descr`: descriptor of the sparse CSR matrix.
- `in` `csr_val`: array of `nnz` elements of the sparse CSR matrix.
- `in` `csr_row_ptr`: array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
- `in` `csr_col_ind`: array of `nnz` elements containing the column indices of the sparse CSR matrix.
- `out` `info`: structure that holds the information collected during the analysis step.
• [in] analysis: rocsparse_analysis_policy_reuse or rocsparse_analysis_policy_force.
• [in] temp_buffer: temporary storage buffer allocated by the user.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m or nnz is invalid.
• rocsparse_status_invalid_pointer: descr, csr_row_ptr, csr_col_ind, info or temp_buffer pointer is invalid.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_zcsrsv_analysis (rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, rocsparse_analysis_policy analysis, rocsparse_solve_policy solve, void *temp_buffer)

Sparse triangular solve using CSR storage format.

rocsparse_csrsv_analysis performs the analysis step for rocsparse_scsrsv_solve(), rocsparse_dcsrsv_solve(), rocsparse_ccsrsv_solve() and rocsparse_zcsrsv_solve(). It is expected that this function will be executed only once for a given matrix and particular operation type. The analysis meta data can be cleared by rocsparse_csrsv_clear().

rocsparse_csrsv_analysis can share its meta data with rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis() and rocsparse_zcsrilu0_analysis(). Selecting rocsparse_analysis_policy_reuse policy can greatly improve computation performance of meta data. However, the user need to make sure that the sparsity pattern remains unchanged. If this cannot be assured, rocsparse_analysis_policy_force has to be used.

Note If the matrix sparsity pattern changes, the gathered information will become invalid.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters
• [in] handle: handle to the rocsparse library context queue.
• [in] trans: matrix operation type.
• [in] m: number of rows of the sparse CSR matrix.
• [in] nnz: number of non-zero entries of the sparse CSR matrix.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
• [out] info: structure that holds the information collected during the analysis step.
• [in] analysis: rocsparse_analysis_policy_reuse or rocsparse_analysis_policy_force.
• [in] temp_buffer: temporary storage buffer allocated by the user.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m or nnz is invalid.
• rocsparse_status_invalid_pointer: descr, csr_row_ptr, csr_col_ind, info or temp_buffer pointer is invalid.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.9.10 rocsparse_csrsv_solve()

rocsparse_status rocsparse_csrsv_solve(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const float *alpha, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const float *x, float *y, rocsparse_solve_policy policy, void *temp_buffer)

rocsparse_status rocsparse_dcsrsv_solve(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const double *alpha, const rocsparse_mat_descr descr, const double *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const double *x, double *y, rocsparse_solve_policy policy, void *temp_buffer)

Sparse triangular solve using CSR storage format.
rocsparse_csrsv_solve solves a sparse triangular linear system of a sparse \( m \times m \) matrix, defined in CSR storage format, a dense solution vector \( y \) and the right-hand side \( x \) that is multiplied by \( \alpha \), such that

\[ op(A) \cdot y = \alpha \cdot x, \]

with

\[ op(A) = \begin{cases} 
  A, & \text{if } trans = \text{rocsparse_operation_none} \\
  A^T, & \text{if } trans = \text{rocsparse_operation_transpose} \\
  A^H, & \text{if } trans = \text{rocsparse_operation_conjugate_transpose} 
\end{cases} \]

rocsparse_csrsv_solve requires a user allocated temporary buffer. Its size is returned by rocsparse_scsrsv_buffer_size(), rocsparse_dcsrsv_buffer_size(), rocsparse_ccsrsv_buffer_size() or rocsparse_zcsrsv_buffer_size(). Furthermore, analysis meta data is required. It can be obtained
by \texttt{rocsparse_scsrsv_analysis()}, \texttt{rocsparse_dcsrsv_analysis()}, \texttt{rocsparse_ccsrsv_analysis()} or \texttt{rocsparse_zcsrsv_analysis()}. \texttt{rocsparse_csrsv_solve} reports the first zero pivot (either numerical or structural zero). The zero pivot status can be checked calling \texttt{rocsparse_csrsv_zero_pivot()}. If \texttt{rocsparse_diag_type} == \texttt{rocsparse_diag_type_unit}, no zero pivot will be reported, even if $A_{j,j} = 0$ for some $j$.

**Note** The sparse CSR matrix has to be sorted. This can be achieved by calling \texttt{rocsparse_csrsort()}.  

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.  

**Note** Currently, only \texttt{trans} == \texttt{rocsparse_operation_none} is supported.

**Example** Consider the lower triangular $m \times m$ matrix $L$, stored in CSR storage format with unit diagonal. The following example solves $L \cdot y = x$.

```c
// Create rocSPARSE handle
rocsparse_handle handle;
rocsparse_create_handle(handle);

// Create matrix descriptor
rocsparse_mat_descr descr;
rocsparse_create_mat_descr(descr);
rocsparse_set_mat_fill_mode(descr, rocsparse_fill_mode_lower);
rocsparse_set_mat_diag_type(descr, rocsparse_diag_type_unit);

// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(info);

// Obtain required buffer size
size_t buffer_size;
rocsparse_dcsrsv_buffer_size(handle,
    rocsparse_operation_none,
    m,
    nnz,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size);

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);

// Perform analysis step
rocsparse_dcsrsv_analysis(handle,
    rocsparse_operation_none,
    m,
    nnz,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    rocsparse_analysis_policy_reuse,

(continues on next page)```
In the previous page, we had:

```c
rocspars Solve_policy_auto,
    temp_buffer);

// Solve Ly = x
rocspars dcsrsv solve(handle,
    rocspars operation none,
    m,
    nnz,
    alpha,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    x,
    y,
    rocspars solve_policy auto,
    temp_buffer);

// No zero pivot should be found, with L having unit diagonal

// Clean up
hipFree(temp_buffer);
rocspars destroy mat_info(info);
rocspars destroy mat_descr(descr);
rocspars destroy handle(handle);
```

### Parameters

- **[in]** handle: handle to the rocspars library context queue.
- **[in]** trans: matrix operation type.
- **[in]** m: number of rows of the sparse CSR matrix.
- **[in]** nnz: number of non-zero entries of the sparse CSR matrix.
- **[in]** alpha: scalar $\alpha$.
- **[in]** descr: descriptor of the sparse CSR matrix.
- **[in]** csr_val: array of nnz elements of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[in]** csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- **[in]** info: structure that holds the information collected during the analysis step.
- **[in]** x: array of m elements, holding the right-hand side.
- **[out]** y: array of m elements, holding the solution.
- **[in]** policy: rocspars solve_policy auto.
- **[in]** temp_buffer: temporary storage buffer allocated by the user.

### Return Value

- rocspars status success: the operation completed successfully.
• rocsparsstatus_invalid_handle: the library context was not initialized.
• rocsparsstatus_invalid_size: m or nnz is invalid.
• rocsparsstatus.invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x or y pointer is invalid.
• rocsparsstatus_arch_mismatch: the device is not supported.
• rocsparsstatus_internal_error: an internal error occurred.
• rocsparsstatus_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparsstatus rocsparsccsrsv_solvex(rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const rocsparse_float_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const rocsparse_float_complex *x, rocsparse_float_complex *y, rocsparse_solve_policy policy, void *temp_buffer)

Sparse triangular solve using CSR storage format.

rocsparsccsrsv_solvex solves a sparse triangular linear system of a sparse m \times m matrix, defined in CSR storage format, a dense solution vector y and the right-hand side x that is multiplied by \( \alpha \), such that

\[
\mathbf{op}(\mathbf{A}) \cdot \mathbf{y} = \alpha \cdot \mathbf{x},
\]

with

\[
\mathbf{op}(\mathbf{A}) = \begin{cases} 
\mathbf{A}, & \text{if trans == rocsparsoperation\_none} \\
\mathbf{A}^T, & \text{if trans == rocsparsoperation\_transpose} \\
\mathbf{A}^H, & \text{if trans == rocsparsoperation\_conjugate\_transpose}
\end{cases}
\]

rocsparsccsrsv_solvex requires a user allocated temporary buffer. Its size is returned by rocsparsccsrsv_buffer_size(), rocsparsdesrsv_buffer_size(), rocsparsccsrsv_buffer_size() or rocsparsccsrsv_buffer_size(). Furthermore, analysis meta data is required. It can be obtained by rocsparsccsrsv_analysis(), rocsparsdesrsv_analysis(), rocsparsccsrsv_analysis() or rocsparsccsrsv_analysis(). rocsparsccsrsv_solvex reports the first zero pivot (either numerical or structural zero). The zero pivot status can be checked calling rocsparsccsrsv_zero_pivot(). If rocsparsdiag_type == rocsparsdiag_type_unit, no zero pivot will be reported, even if \( A_{j,j} = 0 \) for some \( j \).

Note The sparse CSR matrix has to be sorted. This can be achieved by calling rocsparsccsorsort().

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note Currently, only trans == rocsparsoperation\_none is supported.

Example Consider the lower triangular m \times m matrix \( L \), stored in CSR storage format with unit diagonal. The following example solves \( L \cdot \mathbf{y} = \mathbf{x} \).

```c
// Create rocSPARSE handle
rocspars_handle handle;
rocspars_create_handle(&handle);

// Create matrix descriptor
```
rocsparse_mat_descr descr;
rocsparse_create_mat_descr(&descr);
rocsparse_set_mat_fill_mode(descr, rocsparse_fill_mode_lower);
rocsparse_set_mat_diag_type(descr, rocsparse_diag_type_unit);

// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(&info);

// Obtain required buffer size
size_t buffer_size;
rocsparse_dcsrsv_buffer_size(handle,
   rocsparse_operation_none,
   m,
   nnz,
   descr,
   csr_val,
   csr_row_ptr,
   csr_col_ind,
   info,
   &buffer_size);

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);

// Perform analysis step
rocsparse_dcsrsv_analysis(handle,
   rocsparse_operation_none,
   m,
   nnz,
   descr,
   csr_val,
   csr_row_ptr,
   csr_col_ind,
   info,
   rocsparse_analysis_policy_reuse,
   rocsparse_solve_policy_auto,
   temp_buffer);

// Solve Ly = x
rocsparse_dcsrsv_solve(handle,
   rocsparse_operation_none,
   m,
   nnz,
   &alpha,
   descr,
   csr_val,
   csr_row_ptr,
   csr_col_ind,
   info,
   x,
   y,
   rocsparse_solve_policy_auto,
   temp_buffer);

// No zero pivot should be found, with L having unit diagonal

(continues on next page)
// Clean up
hipFree(temp_buffer);
rocspars_destroy_mat_info(info);
rocspars_destroy_mat_descr(descr);
rocspars_destroy_handle(handle);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] alpha: scalar \( \alpha \).
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [in] info: structure that holds the information collected during the analysis step.
- [in] x: array of m elements, holding the right-hand side.
- [out] y: array of m elements, holding the solution.
- [in] temp_buffer: temporary storage buffer allocated by the user.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x or y pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.
rocsparse_status rocsparse_zcsrsv_solve (rocsparse_handle handle, rocsparse_operation trans, rocsparse_int m, rocsparse_int nnz, const rocsparse_double_complex *alpha, const rocsparse_mat_desc descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, const rocsparse_double_complex *x, rocsparse_double_complex *y, rocsparse_solve_policy policy, void *temp_buffer)

Sparse triangular solve using CSR storage format.

rocsparse_csrsv_solve solves a sparse triangular linear system of a sparse \( m \times m \) matrix, defined in CSR storage format, a dense solution vector \( y \) and the right-hand side \( x \) that is multiplied by \( \alpha \), such that

\[
\text{op}(A) \cdot y = \alpha \cdot x,
\]

with

\[
\text{op}(A) = \begin{cases} 
A, & \text{if } \text{trans} == \text{rocsparse_operation_none} \\
A^T, & \text{if } \text{trans} == \text{rocsparse_operation_transpose} \\
A^H, & \text{if } \text{trans} == \text{rocsparse_operation_conjugate_transpose}
\end{cases}
\]

\rocsparse_csrsv_solve requires a user allocated temporary buffer. Its size is returned by \rocsparse_scsrsv_buffer_size(), \rocsparse_dcsrsv_buffer_size(), \rocsparse_ccsrsv_buffer_size() or \rocsparse_zcsrsv_buffer_size(). Furthermore, analysis meta data is required. It can be obtained by \rocsparse_scsrsv_analysis(), \rocsparse_dcsrsv_analysis(), \rocsparse_ccsrsv_analysis() or \rocsparse_zcsrsv_analysis(). \rocsparse_csrsv_solve reports the first zero pivot (either numerical or structural zero). The zero pivot status can be checked calling \rocsparse_csrsv_zero_pivot(). If \rocsparse_diag_type == \rocsparse_diag_type_unit, no zero pivot will be reported, even if \( A_{j,j} = 0 \) for some \( j \).

**Note** The sparse CSR matrix has to be sorted. This can be achieved by calling \rocsparse_csrsort().

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Currently, only \( \text{trans} == \text{rocsparse_operation_none} \) is supported.

**Example** Consider the lower triangular \( m \times m \) matrix \( L \), stored in CSR storage format with unit diagonal. The following example solves \( L \cdot y = x \).

```c
// Create rocSPARSE handle
rocsparse_handle handle;
rocsparse_create_handle(&handle);

// Create matrix descriptor
rocsparse_mat_desc descr;
rocsparse_create_mat_desc(&descr);
rocsparse_set_mat_fill_mode(descr, rocsparse_fill_mode_lower);
rocsparse_set_mat_diag_type(descr, rocsparse_diag_type_unit);

// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(&info);

// Obtain required buffer size
size_t buffer_size;
```

(continues on next page)
rocsparse_dcsrsrv_buffer_size(handle,
    rocsparse_operation_none,
    m,
    nnz,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size);

    // Allocate temporary buffer
    void* temp_buffer;
    hipMalloc(&temp_buffer, buffer_size);

    // Perform analysis step
    rocsparse_dcsrsrv_analysis(handle,
        rocsparse_operation_none,
        m,
        nnz,
        descr,
        csr_val,
       csr_row_ptr,
        csr_col_ind,
        info,
        rocsparse_analysis_policy_reuse,
        rocsparse_solve_policy_auto,
        temp_buffer);

    // Solve Ly = x
    rocsparse_dcsrsrv_solve(handle,
        rocsparse_operation_none,
        m,
        nnz,
        alpha,
        descr,
        csr_val,
        csr_row_ptr,
        csr_col_ind,
        info,
        x,
        y,
        rocsparse_solve_policy_auto,
        temp_buffer);

    // No zero pivot should be found, with L having unit diagonal

    // Clean up
    hipFree(temp_buffer);
    rocsparse_destroy_mat_info(info);
    rocsparse_destroy_mat_descr(descr);
    rocsparse_destroy_handle(handle);

Parameters

- [in] handle: handle to the rocsparse library context queue.
• [in] m: number of rows of the sparse CSR matrix.
• [in] nnz: number of non-zero entries of the sparse CSR matrix.
• [in] alpha: scalar α.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
• [in] info: structure that holds the information collected during the analysis step.
• [in] x: array of m elements, holding the right-hand side.
• [out] y: array of m elements, holding the solution.
• [in] temp_buffer: temporary storage buffer allocated by the user.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m or nnz is invalid.
• rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, x or y pointer is invalid.
• rocsparse_status_arch_mismatch: the device is not supported.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.9.11 rocsparse_csrsv_clear()

rocsparse_status rocsparse_csrsv_clear (rocsparse_handle handle, const rocsparse_mat_desc descr, rocsparse_mat_info info)

Sparse triangular solve using CSR storage format.

rocsparse_csrsv_clear deallocates all memory that was allocated by rocsparse_scsrsv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_cscsrsv_analysis() or rocsparse_zcsrsv_analysis(). This is especially useful, if memory is an issue and the analysis data is not required for further computation, e.g. when switching to another sparse matrix format. Calling rocsparse_csrsv_clear is optional. All allocated resources will be cleared, when the opaque rocsparse_mat_info struct is destroyed using rocsparse_destroy_mat_info().

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [inout] info: structure that holds the information collected during the analysis step.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_pointer: info pointer is invalid.
• rocsparse_status_memory_error: the buffer holding the meta data could not be deallocated.
• rocsparse_status_internal_error: an internal error occurred.

2.12.5.10 Sparse Level 3 Functions

This module holds all sparse level 3 routines.

The sparse level 3 routines describe operations between a matrix in sparse format and multiple vectors in dense format that can also be seen as a dense matrix.

2.12.5.10.1 rocsparse_csrmm()

rocsparse_status rocsparse_csrmm(rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, rocsparse_int nnz, const float *alpha, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, const float *B, rocsparse_int ldb, const float *beta, float *C, rocsparse_int ldc)

Sparse matrix dense matrix multiplication using CSR storage format.

rocsparse_csrmm multiplies the scalar $\alpha$ with a sparse $m \times k$ matrix $A$, defined in CSR storage format, and the dense $k \times n$ matrix $B$ and adds the result to the dense $m \times n$ matrix $C$ that is multiplied by the scalar $\beta$, such that

$$ C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C, $$

with

$$ \text{op}(A) = \begin{cases} A, & \text{if } \text{trans}_A = \text{rocsparse_operation_none} \\ A^T, & \text{if } \text{trans}_A = \text{rocsparse_operation_transpose} \\ A^H, & \text{if } \text{trans}_A = \text{rocsparse_operation_conjugate_transpose} \end{cases} $$

and

$$ \text{op}(B) = \begin{cases} B, & \text{if } \text{trans}_B = \text{rocsparse_operation_none} \\ B^T, & \text{if } \text{trans}_B = \text{rocsparse_operation_transpose} \\ B^H, & \text{if } \text{trans}_B = \text{rocsparse_operation_conjugate_transpose} \end{cases} $$
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note Currently, only trans_A == rocsparse_operation_none is supported.

Example This example multiplies a CSR matrix with a dense matrix.

// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8
rocsparse_int m = 3;
rocsparse_int k = 5;
rocsparse_int nnz = 8;

csr_row_ptr[m+1] = {0, 3, 5, 8}; // device memory
csr_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Set dimension n of B
rocsparse_int n = 64;

// Allocate and generate dense matrix B
std::vector<float> hB(k * n);
for(rocsparse_int i = 0; i < k * n; ++i)
{
    hB[i] = static_cast<float>(rand()) / RAND_MAX;
}

// Copy B to the device
float* B;
hipMalloc((void**)&B, sizeof(float) * k * n);
hipMemcpy(B, hB.data(), sizeof(float) * k * n, hipMemcpyHostToDevice);

// alpha and beta
float alpha = 1.0f;
float beta = 0.0f;

// Allocate memory for the resulting matrix C
float* C;
hipMalloc((void**)&C, sizeof(float) * m * n);

// Perform the matrix multiplication
rocsparse_scsrmm(handle,
rocsparse_operation_none,
rocsparse_operation_none,
m,
n,  
k,

nnz,
&alpha,
descr,
csr_val,
csr_row_ptr,
csr_col_ind,
B,
k,
&beta,
C,
m);  

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix A operation type.
- [in] trans_B: matrix B operation type.
- [in] m: number of rows of the sparse CSR matrix A.
- [in] n: number of columns of the dense matrix \( op(B) \) and C.
- [in] k: number of columns of the sparse CSR matrix A.
- [in] nnz: number of non-zero entries of the sparse CSR matrix A.
- [in] alpha: scalar \( \alpha \).
- [in] descr: descriptor of the sparse CSR matrix A. Currently, only rocsparse_matrix_type_general is supported.
- [in] csr_val: array of nnz elements of the sparse CSR matrix A.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix A.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix A.
- [in] B: array of dimension \( ldb \times n \) \( (op(B) == B) \) or \( ldb \times k \) \( (op(B) == B^T) \) or \( op(B) == B^H \).
- [in] ldb: leading dimension of B, must be at least \( \max(1,k) \) \( (op(A) == A^T) \) or \( \max(1,m) \) \( (op(A) == A^T \text{ or } op(A) == A^H) \).
- [in] beta: scalar \( \beta \).
- [inout] C: array of dimension \( ldc \times n \).
- [in] ldc: leading dimension of C, must be at least \( \max(1,k) \) \( (op(A) == A) \) or \( \max(1,m) \) \( (op(A) == A^T \text{ or } op(A) == A^H) \).

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n, k, nnz, ldb or ldc is invalid.
• rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, B, beta or C pointer is invalid.
• rocsparse_status_arch_mismatch: the device is not supported.
• rocsparse_status_not_implemented: trans_A != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_ccsrmm(rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, rocsparse_int nnz, const rocsparse_float_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, const rocsparse_float_complex *B, rocsparse_int ldb, const rocsparse_float_complex *beta, rocsparse_float_complex *C, rocsparse_int ldc)

Sparse matrix dense matrix multiplication using CSR storage format.

rocsparse_ccsrmm multiplies the scalar $\alpha$ with a sparse $m \times k$ matrix $A$, defined in CSR storage format, and the dense $k \times n$ matrix $B$ and adds the result to the dense $m \times n$ matrix $C$ that is multiplied by the scalar $\beta$, such that

$$C := \alpha \cdot op(A) \cdot op(B) + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & \text{if } \text{trans}_A == \text{rocsparse_operation_none} \\ A^T, & \text{if } \text{trans}_A == \text{rocsparse_operation_transpose} \\ A^H, & \text{if } \text{trans}_A == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

and

$$op(B) = \begin{cases} B, & \text{if } \text{trans}_B == \text{rocsparse_operation_none} \\ B^T, & \text{if } \text{trans}_B == \text{rocsparse_operation_transpose} \\ B^H, & \text{if } \text{trans}_B == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

for (i = 0; i < ldc; ++i) {
    for (j = 0; j < n; ++j) {
        C[i][j] = beta * C[i][j];
        for (k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k) {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note Currently, only trans_A == rocsparse_operation_none is supported.

Example This example multiplies a CSR matrix with a dense matrix.
rocsparse_int m = 3;
rocsparse_int k = 5;
rocsparse_int nnz = 8;

csr_row_ptr[m+1] = {0, 3, 5, 8}; // device memory
csr_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Set dimension n of B
rocsparse_int n = 64;

// Allocate and generate dense matrix B
std::vector<float> hB(k * n);
for(rocsparse_int i = 0; i < k * n; ++i)
{
    hB[i] = static_cast<float>(rand()) / RAND_MAX;
}

// Copy B to the device
float* B;
hipMalloc((void**)&B, sizeof(float) * k * n);
hipMemcpy(B, hB.data(), sizeof(float) * k * n, hipMemcpyHostToDevice);

// alpha and beta
float alpha = 1.0f;
float beta = 0.0f;

// Allocate memory for the resulting matrix C
float* C;
hipMalloc((void**)&C, sizeof(float) * m * n);

// Perform the matrix multiplication
rocsparse_scsrmm(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    nnz,
    &alpha,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    B,
    k,
    &beta,
    C,
    m);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix A operation type.
• [in] trans_B: matrix B operation type.
• [in] m: number of rows of the sparse CSR matrix A.
• [in] n: number of columns of the dense matrix op(B) and C.
• [in] k: number of columns of the sparse CSR matrix A.
• [in] nnz: number of non-zero entries of the sparse CSR matrix A.
• [in] alpha: scalar \( \alpha \).
• [in] descr: descriptor of the sparse CSR matrix A. Currently, only rocsparse_matrix_type_general is supported.
• [in] csr_val: array of nnz elements of the sparse CSR matrix A.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix A.
• [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix A.
• [in] B: array of dimension ldb \times n (op(B) == B) or ldb \times k (op(B) == B^T or op(B) == B^H).
• [in] ldb: leading dimension of B, must be at least max(1, k) (op(A) == A) or max(1, m) (op(A) == A^T or op(A) == A^H).
• [in] beta: scalar \( \beta \).
• [inout] C: array of dimension ldc \times n.
• [in] ldc: leading dimension of C, must be at least max(1, m) (op(A) == A) or max(1, k) (op(A) == A^T or op(A) == A^H).

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n, k, nnz, ldb or ldc is invalid.
• rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, B, beta or C pointer is invalid.
• rocsparse_status_arch_mismatch: the device is not supported.
• rocsparse_status_not_implemented: trans_A != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

Sparse matrix dense matrix multiplication using CSR storage format.

rocsparse_status rocsparse_zcsrmm(rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, rocsparse_int nnz, const rocsparse_double_complex *alpha, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, const rocsparse_double_complex *B, rocsparse_int ldb, const rocsparse_double_complex *beta, rocsparse_double_complex *C, rocsparse_int ldc)

rocsparse_zcsrmm multiplies the scalar \( \alpha \) with a sparse \( m \times k \) matrix A, defined in CSR storage format, and the dense \( k \times n \) matrix B and adds the result to the dense \( m \times n \) matrix C that is multiplied by the scalar \( \beta \).
such that

\[ C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C, \]

with

\[
\text{op}(A) = \begin{cases} 
A, & \text{if } \text{trans}_A == \text{rocsparse_operation_none} \\
A^T, & \text{if } \text{trans}_A == \text{rocsparse_operation_transpose} \\
A^H, & \text{if } \text{trans}_A == \text{rocsparse_operation_conjugate_transpose}
\end{cases}
\]

and

\[
\text{op}(B) = \begin{cases} 
B, & \text{if } \text{trans}_B == \text{rocsparse_operation_none} \\
B^T, & \text{if } \text{trans}_B == \text{rocsparse_operation_transpose} \\
B^H, & \text{if } \text{trans}_B == \text{rocsparse_operation_conjugate_transpose}
\end{cases}
\]

```c
for(i = 0; i < ldc; ++i)
{
    for(j = 0; j < n; ++j)
    {
        C[i][j] = beta * C[i][j];
        for(k = csr_row_ptr[i]; k < csr_row_ptr[i + 1]; ++k)
        {
            C[i][j] += alpha * csr_val[k] * B[csr_col_ind[k]][j];
        }
    }
}
```

Note  This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Note  Currently, only \text{trans}_A == \text{rocsparse_operation_none} is supported.

Example  This example multiplies a CSR matrix with a dense matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8
rocsparse_int m = 3;
rocsparse_int k = 5;
rocsparse_int nnz = 8;
csr_row_ptr[m+1] = {0, 3, 5, 8};  // device memory
csr_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Set dimension n of B
rocsparse_int n = 64;

// Allocate and generate dense matrix B
std::vector<float> hB(k * n);
for(rocsparse_int i = 0; i < k * n; ++i)
{
    hB[i] = static_cast<float>(rand()) / RAND_MAX;
}
```

(continues on next page)
// Copy B to the device
float* B;
hipMalloc((void**)&B, sizeof(float) * k * n);
hipMemcpy(B, hB.data(), sizeof(float) * k * n, hipMemcpyHostToDevice);

// alpha and beta
float alpha = 1.0f;
float beta = 0.0f;

// Allocate memory for the resulting matrix C
float* C;
hipMalloc((void**)&C, sizeof(float) * m * n);

// Perform the matrix multiplication
rocSparse_scsrmm(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    nnz,
    &alpha,
    descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    B,
    k,
    &beta,
    C,
    m);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix $A$ operation type.
- [in] trans_B: matrix $B$ operation type.
- [in] m: number of rows of the sparse CSR matrix $A$.
- [in] n: number of columns of the dense matrix $op(B)$ and $C$.
- [in] k: number of columns of the sparse CSR matrix $A$.
- [in] nnz: number of non-zero entries of the sparse CSR matrix $A$.
- [in] alpha: scalar $\alpha$.
- [in] descr: descriptor of the sparse CSR matrix $A$. Currently, only $rocsparse_matrix_type_general$ is supported.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix $A$.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix $A$.
- [in] B: array of dimension $ldb \times n$ ($op(B) == B$) or $ldb \times k$ ($op(B) == B^T$ or $op(B) == B^H$).
• [in] ldb: leading dimension of $B$, must be at least $\max(1, k) \ (\text{op}(A) == A)$ or $\max(1, m) \ (\text{op}(A) == A^T$ or $\text{op}(A) == A^H$).

• [in] beta: scalar $\beta$.

• [inout] C: array of dimension $ldc \times n$.

• [in] ldc: leading dimension of $C$, must be at least $\max(1, m) \ (\text{op}(A) == A)$ or $\max(1, k) \ (\text{op}(A) == A^T$ or $\text{op}(A) == A^H$).

Return Value

• rocsparse_status_success: the operation completed successfully.

• rocsparse_status_invalid_handle: the library context was not initialized.

• rocsparse_status_invalid_size: $m, n, k, \text{nnz}, ldb$ or $ldc$ is invalid.

• rocsparse_status_invalid_pointer: descr, alpha, csr_val, csr_row_ptr, csr_col_ind, $B$, $\beta$ or $C$ pointer is invalid.

• rocsparse_status_arch_mismatch: the device is not supported.

• rocsparse_status_not_implemented: trans_A != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.11 Sparse Extra Functions

This module holds all sparse extra routines.

The sparse extra routines describe operations that manipulate sparse matrices.

2.12.5.11.1 rocsparse_csrgemm_buffer_size()

rocsparse_status rocsparse_csrgemm_buffer_size (rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int $m$, rocsparse_int $n$, rocsparse_int $k$, const float *alpha, const rocsparse_mat_descr descr_A, rocsparse_int nnz_A, const rocsparse_int *csr_row_ptr_A, const rocsparse_int *csr_col_ind_A, const rocsparse_mat_descr descr_B, rocsparse_int $nnz_B$, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const float *beta, const rocsparse_mat_descr descr_D, rocsparse_int $nnz_D$, const rocsparse_int *csr_row_ptr_D, const rocsparse_int *csr_col_ind_D, rocsparse_mat_info info_C, size_t *buffer_size)
rocsparse_status rocsparse_dcsrgemm_buffer_size(rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, const double *alpha, const rocsparse_mat_desc descr_A, rocsparse_int nnz_A, const rocsparse_int *csr_row_ptr_A, const rocsparse_int *csr_col_ind_A, const rocsparse_mat_desc descr_B, rocsparse_int nnz_B, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const double *beta, const rocsparse_mat_desc descr_D, rocsparse_int nnz_D, const rocsparse_int *csr_row_ptr_D, const rocsparse_int *csr_col_ind_D, rocsparse_mat_info info_C, size_t *buffer_size)

Sparse matrix sparse matrix multiplication using CSR storage format. rocsparse_dcsrgemm_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_csr gemm(), rocsparse_scsr gemm(), rocsparse_dcsrgemm(), rocsparse_ccsr gemm() and rocsparse_zcsrgemm(). The temporary storage buffer must be allocated by the user.

Note Please note, that for matrix products with more than 4096 non-zero entries per row, additional temporary storage buffer is allocated by the algorithm.

Note Please note, that for matrix products with more than 8192 intermediate products per row, additional temporary storage buffer is allocated by the algorithm.

Note Currently, only trans_A == trans_B == rocsparse_operation_none is supported.

Note Currently, only rocsparse_matrix_type_general is supported.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix A operation type.
- [in] trans_B: matrix B operation type.
- [in] m: number of rows of the sparse CSR matrix op(A) and C.
- [in] n: number of columns of the sparse CSR matrix op(B) and C.
- [in] k: number of columns of the sparse CSR matrix op(A) and number of rows of the sparse CSR matrix op(B).
- [in] alpha: scalar $\alpha$.
- [in] descr_A: descriptor of the sparse CSR matrix A. Currently, only rocsparse_matrix_type_general is supported.
- [in] nnz_A: number of non-zero entries of the sparse CSR matrix A.
- [in] csr_row_ptr_A: array of m+1 elements (op(A) == A, k+1 otherwise) that point to the start of every row of the sparse CSR matrix op(A).
- [in] csr_col_ind_A: array of nnz_A elements containing the column indices of the sparse CSR matrix A.
- [in] descr_B: descriptor of the sparse CSR matrix B. Currently, only rocsparse_matrix_type_general is supported.
- [in] nnz_B: number of non-zero entries of the sparse CSR matrix B.
• [in] csr_row_ptr_B: array of \(k+1\) elements (\(op(B) == B, m+1\) otherwise) that point to the start of every row of the sparse CSR matrix \(op(B)\).

• [in] csr_col_ind_B: array of \(nnz_B\) elements containing the column indices of the sparse CSR matrix \(B\).

• [in] beta: scalar \(\beta\).

• [in] descr_D: descriptor of the sparse CSR matrix \(D\). Currently, only \texttt{roc-sparse_matrix_type_general} is supported.

• [in] nnz_D: number of non-zero entries of the sparse CSR matrix \(D\).

• [in] csr_row_ptr_D: array of \(m+1\) elements that point to the start of every row of the sparse CSR matrix \(D\).

• [in] csr_col_ind_D: array of \(nnz_D\) elements containing the column indices of the sparse CSR matrix \(D\).

• [inout] info_C: structure that holds meta data for the sparse CSR matrix \(C\).

• [out] buffer_size: number of bytes of the temporary storage buffer required by \texttt{rocsparse_ccsrgemm\_nnz()}, \texttt{rocsparse_scsrgemm()}, \texttt{rocsparse_dcsrgemm()}, \texttt{rocsparse_ccsrgemm() and rocsparse_zcsrgemm()}.  

Return Value

• \texttt{rocsparse\_status\_success}: the operation completed successfully.

• \texttt{rocsparse\_status\_invalid\_handle}: the library context was not initialized.

• \texttt{rocsparse\_status\_invalid\_size}: \(m, n, k, nnz_A, nnz_B\) or \(nnz_D\) is invalid.

• \texttt{rocsparse\_status\_invalid\_pointer}: \(alpha\) and \(beta\) are invalid, \texttt{descr\_A}, \texttt{csr\_row\_ptr\_A}, \texttt{csr\_col\_ind\_A}, \texttt{descr\_B}, \texttt{csr\_row\_ptr\_B} or \texttt{csr\_col\_ind\_B} are invalid if \(alpha\) is valid, \texttt{descr\_D}, \texttt{csr\_row\_ptr\_D} or \texttt{csr\_col\_ind\_D} is invalid if \(beta\) is valid, \texttt{info\_C} or \texttt{buffer\_size} is invalid.

• \texttt{rocsparse\_status\_not\_implemented}: \(trans\_A \neq \texttt{rocsparse\_operation\_none}\), \(trans\_B \neq \texttt{rocsparse\_operation\_none}\), or \texttt{rocsparse\_matrix\_type} \(\neq \texttt{rocsparse\_matrix\_type\_general}\).

\texttt{rocsparse\_status\_rocsparse\_ccsrgemm\_buffer\_size} (\texttt{rocsparse\_handle handle}, \texttt{rocsparse\_operation trans\_A, rocsparse\_operation trans\_B}, \texttt{rocsparse\_int m, rocsparse\_int n, rocsparse\_int k}, \texttt{const rocsparse\_float\_complex \*alpha}, \texttt{const rocsparse\_mat\_descr descr\_A}, \texttt{rocsparse\_int nnz\_A, const rocsparse\_int \*csr\_row\_ptr\_A, const rocsparse\_int \*csr\_col\_ind\_A, const rocsparse\_mat\_descr descr\_B}, \texttt{rocsparse\_int nnz\_B, const rocsparse\_int \*csr\_row\_ptr\_B, const rocsparse\_int \*csr\_col\_ind\_B, const rocsparse\_float\_complex \*beta}, \texttt{const rocsparse\_mat\_descr descr\_D}, \texttt{rocsparse\_int nnz\_D, const rocsparse\_int \*csr\_row\_ptr\_D, const rocsparse\_int \*csr\_col\_ind\_D, rocsparse\_mat\_info info\_C, size\_t \*buffer\_size})

Sparse matrix sparse matrix multiplication using CSR storage format.
rocsparse_csrgemm_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_csrgemm_nnz(), rocsparse_scsrgemm(), rocsparse_dcsrgemm(), rocsparse_ccsrgemm() and rocsparse_zcsrgemm(). The temporary storage buffer must be allocated by the user.

**Note** Please note, that for matrix products with more than 4096 non-zero entries per row, additional temporary storage buffer is allocated by the algorithm.

**Note** Please note, that for matrix products with more than 8192 intermediate products per row, additional temporary storage buffer is allocated by the algorithm.

**Note** Currently, only trans_A == trans_B == rocsparse_operation_none is supported.

**Note** Currently, only rocsparse_matrix_type_general is supported.

**Parameters**

- **[in] handle**: handle to the rocsparse library context queue.
- **[in] trans_A**: matrix A operation type.
- **[in] trans_B**: matrix B operation type.
- **[in] m**: number of rows of the sparse CSR matrix op(A) and C.
- **[in] n**: number of columns of the sparse CSR matrix op(B) and C.
- **[in] k**: number of columns of the sparse CSR matrix op(A) and number of rows of the sparse CSR matrix op(B).
- **[in] alpha**: scalar α.
- **[in] descr_A**: descriptor of the sparse CSR matrix A. Currently, only rocsparse_matrix_type_general is supported.
- **[in] nnz_A**: number of non-zero entries of the sparse CSR matrix A.
- **[in] csr_row_ptr_A**: array of m+1 elements (op(A) == A, k+1 otherwise) that point to the start of every row of the sparse CSR matrix op(A).
- **[in] csr_col_ind_A**: array of nnz_A elements containing the column indices of the sparse CSR matrix A.
- **[in] descr_B**: descriptor of the sparse CSR matrix B. Currently, only rocsparse_matrix_type_general is supported.
- **[in] nnz_B**: number of non-zero entries of the sparse CSR matrix B.
- **[in] csr_row_ptr_B**: array of k+1 elements (op(B) == B, m+1 otherwise) that point to the start of every row of the sparse CSR matrix op(B).
- **[in] csr_col_ind_B**: array of nnz_B elements containing the column indices of the sparse CSR matrix B.
- **[in] beta**: scalar β.
- **[in] descr_D**: descriptor of the sparse CSR matrix D. Currently, only rocsparse_matrix_type_general is supported.
- **[in] nnz_D**: number of non-zero entries of the sparse CSR matrix D.
- **[in] csr_row_ptr_D**: array of m+1 elements that point to the start of every row of the sparse CSR matrix D.
- **[in] csr_col_ind_D**: array of nnz_D elements containing the column indices of the sparse CSR matrix D.
• [inout] info_C: structure that holds meta data for the sparse CSR matrix C.

• [out] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_csrgemm_nnz(), rocsparse_scsrgemm(), rocsparse_dcsrgemm(), rocsparse_ccsrgemm() and rocsparse_zcsrgemm().

Return Value

• rocsparse_status_success: the operation completed successfully.

• rocsparse_status_invalid_handle: the library context was not initialized.

• rocsparse_status_invalid_size: m, n, k, nnz_A, nnz_B or nnz_D is invalid.

• rocsparse_status_invalid_pointer: alpha and beta are invalid, descr_A, csr_row_ptr_A, csr_col_ind_A, descr_B, csr_row_ptr_B or csr_col_ind_B are invalid if alpha is valid, descr_D, csr_row_ptr_D or csr_col_ind_D is invalid if beta is valid, info_C or buffer_size is invalid.

• rocsparse_status_not_implemented: trans_A != rocsparse_operation_none, trans_B != rocsparse_operation_none, or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_zcsrgemm_buffer_size (rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, const rocsparse_double_complex *alpha, const rocsparse_mat_descr descr_A, rocsparse_int nnz_A, const rocsparse_int *csr_row_ptr_A, const rocsparse_int *csr_col_ind_A, const rocsparse_mat_descr descr_B, rocsparse_int nnz_B, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const rocsparse_double_complex *beta, const rocsparse_mat_descr descr_D, rocsparse_int nnz_D, const rocsparse_int *csr_row_ptr_D, const rocsparse_int *csr_col_ind_D, rocsparse_mat_info info_C, size_t *buffer_size)

Sparse matrix sparse matrix multiplication using CSR storage format.

rocsparse_zcsrgemm_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_csrgemm_nnz(), rocsparse_scsrgemm(), rocsparse_dcsrgemm(), rocsparse_ccsrgemm() and rocsparse_zcsrgemm(). The temporary storage buffer must be allocated by the user.

Note Please note, that for matrix products with more than 4096 non-zero entries per row, additional temporary storage buffer is allocated by the algorithm.

Note Please note, that for matrix products with more than 8192 intermediate products per row, additional temporary storage buffer is allocated by the algorithm.

Note Currently, only trans_A == trans_B == rocsparse_operation_none is supported.

Note Currently, only rocsparse_matrix_type_general is supported.

Parameters

• [in] handle: handle to the rocsparse library context queue.

• [in] trans_A: matrix A operation type.

• [in] trans_B: matrix B operation type.
• [in] m: number of rows of the sparse CSR matrix \( op(A) \) and \( C \).
• [in] n: number of columns of the sparse CSR matrix \( op(B) \) and \( C \).
• [in] k: number of columns of the sparse CSR matrix \( op(A) \) and number of rows of the sparse CSR matrix \( op(B) \).
• [in] alpha: scalar \( \alpha \).
• [in] descr_A: descriptor of the sparse CSR matrix \( A \). Currently, only \textit{roc-sparse\_matrix\_type\_general} is supported.
• [in] nnz_A: number of non-zero entries of the sparse CSR matrix \( A \).
• [in] csr_row_ptr_A: array of \( m+1 \) elements (\( op(A) == A \), \( k+1 \) otherwise) that point to the start of every row of the sparse CSR matrix \( op(A) \).
• [in] csr_col_ind_A: array of \( nnz_A \) elements containing the column indices of the sparse CSR matrix \( A \).
• [in] descr_B: descriptor of the sparse CSR matrix \( B \). Currently, only \textit{roc-sparse\_matrix\_type\_general} is supported.
• [in] nnz_B: number of non-zero entries of the sparse CSR matrix \( B \).
• [in] csr_row_ptr_B: array of \( k+1 \) elements (\( op(B) == B \), \( m+1 \) otherwise) that point to the start of every row of the sparse CSR matrix \( op(B) \).
• [in] csr_col_ind_B: array of \( nnz_B \) elements containing the column indices of the sparse CSR matrix \( B \).
• [in] beta: scalar \( \beta \).
• [in] descr_D: descriptor of the sparse CSR matrix \( D \). Currently, only \textit{roc-sparse\_matrix\_type\_general} is supported.
• [in] nnz_D: number of non-zero entries of the sparse CSR matrix \( D \).
• [in] csr_row_ptr_D: array of \( m+1 \) elements that point to the start of every row of the sparse CSR matrix \( D \).
• [in] csr_col_ind_D: array of \( nnz_D \) elements containing the column indices of the sparse CSR matrix \( D \).
• [inout] info_C: structure that holds meta data for the sparse CSR matrix \( C \).
• [out] buffer_size: number of bytes of the temporary storage buffer required by \textit{roc-sparse\_csrgemm\_nnz()}, \textit{roc-sparse\_scsrgemm()}, \textit{roc-sparse\_dcsgemm()}, \textit{roc-sparse\_ccsrgemm()} and \textit{roc-sparse\_zcsrgemm()}.

Return Value

• rocsparse\_status\_success: the operation completed successfully.
• rocsparse\_status\_invalid\_handle: the library context was not initialized.
• rocsparse\_status\_invalid\_size: \( m, n, k, nnz_A, nnz_B \) or \( nnz_D \) is invalid.
• rocsparse\_status\_invalid\_pointer: \( \alpha \) and \( \beta \) are invalid, descr_A, csr_row_ptr_A, csr_col_ind_A, descr_B, csr_row_ptr_B or csr_col_ind_B are invalid if \( \alpha \) is valid, descr_D, csr_row_ptr_D or csr_col_ind_D is invalid if \( \beta \) is valid, info_C or buffer_size is invalid.
• rocsparse\_status\_not\_implemented: \( \text{trans}_A \neq \text{rocsparse\_operation\_none} \), \( \text{trans}_B \neq \text{rocsparse\_operation\_none} \), or rocsparse\_matrix\_type \( \neq \text{rocsparse\_matrix\_type\_general} \).
2.12.5.11.2 rocsparse_csrgemm_nnz()

```
rocsparse_status rocsparse_csrgemm_nnz(rocsparse_handle handle,
rocsparse_operation trans_A,
rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, const rocsparse_mat_descr descr_A, rocsparse_int nnz_A, const rocsparse_row_ptr_t csr_row_ptr_A, const rocsparse_int *csr_col_ind_A,
const rocsparse_mat_descr descr_B, rocsparse_int nnz_B, const rocsparse_row_ptr_t csr_row_ptr_B, const rocsparse_int *csr_col_ind_B,
const rocsparse_mat_descr descr_D, rocsparse_int nnz_D, const rocsparse_row_ptr_t csr_row_ptr_D, const rocsparse_int *csr_col_ind_D,
const rocsparse_mat_descr descr_C, rocsparse_int *nnz_C, const rocsparse_row_ptr_t csr_row_ptr_C, rocsparse_int *nnz_C, void *temp_buffer)
```

Sparse matrix sparse matrix multiplication using CSR storage format.

rocsparse_csrgemm_nnz computes the total CSR non-zero elements and the CSR row offsets, that point to the start of every row of the sparse CSR matrix, of the resulting multiplied matrix C. It is assumed that csr_row_ptr_C has been allocated with size m + 1. The required buffer size can be obtained by rocsparse_scsrgemm_buffer_size(), rocsparse_dcsrgemm_buffer_size(), rocsparse_ccsrgemm_buffer_size() and rocsparse_zcsrgemm_buffer_size(), respectively.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Note** Please note, that for matrix products with more than 8192 intermediate products per row, additional temporary storage buffer is allocated by the algorithm.

**Note** Currently, only `trans_A == trans_B == rocsparse_operation_none` is supported.

**Note** Currently, only `rocsparse_matrix_type_general` is supported.

**Parameters**

- `[in]` handle: handle to the rocsparse library context queue.
- `[in]` trans_A: matrix A operation type.
- `[in]` trans_B: matrix B operation type.
- `[in]` m: number of rows of the sparse CSR matrix `op(A)` and C.
- `[in]` n: number of columns of the sparse CSR matrix `op(B)` and C.
- `[in]` k: number of columns of the sparse CSR matrix `op(A)` and number of rows of the sparse CSR matrix `op(B)`.
- `[in]` descr_A: descriptor of the sparse CSR matrix A. Currently, only `rocsparse_matrix_type_general` is supported.
- `[in]` nnz_A: number of non-zero entries of the sparse CSR matrix A.
- `[in]` csr_row_ptr_A: array of m+1 elements ( `op(A) == A`, k+1 otherwise) that point to the start of every row of the sparse CSR matrix `op(A)`.
- `[in]` csr_col_ind_A: array of nnz_A elements containing the column indices of the sparse CSR matrix A.
- `[in]` descr_B: descriptor of the sparse CSR matrix B. Currently, only `rocsparse_matrix_type_general` is supported.
- [in] `nnz_B`: number of non-zero entries of the sparse CSR matrix $B$.
- [in] `csr_row_ptr_B`: array of $k+1$ elements ($op(B) == B, m+1$ otherwise) that point to the start of every row of the sparse CSR matrix $op(B)$.
- [in] `csr_col_ind_B`: array of $nnz_B$ elements containing the column indices of the sparse CSR matrix $B$.
- [in] `descr_D`: descriptor of the sparse CSR matrix $D$. Currently, only `roc-sparse_matrix_type_general` is supported.
- [in] `nnz_D`: number of non-zero entries of the sparse CSR matrix $D$.
- [in] `csr_row_ptr_D`: array of $m+1$ elements that point to the start of every row of the sparse CSR matrix $D$.
- [in] `csr_col_ind_D`: array of $nnz_D$ elements containing the column indices of the sparse CSR matrix $D$.
- [in] `descr_C`: descriptor of the sparse CSR matrix $C$. Currently, only `roc-sparse_matrix_type_general` is supported.
- [out] `csr_row_ptr_C`: array of $m+1$ elements that point to the start of every row of the sparse CSR matrix $C$.
- [out] `nnz_C`: pointer to the number of non-zero entries of the sparse CSR matrix $C$.
- [in] `info_C`: structure that holds meta data for the sparse CSR matrix $C$.
- [in] `temp_buffer`: temporary storage buffer allocated by the user, size is returned by `rocsparse_scsrgemm_buffer_size()`, `rocsparse_dcsrgemm_buffer_size()`, `rocsparse_ccsrgemm_buffer_size()` or `rocsparse_zcsrgemm_buffer_size()`.

**Return Value**

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: $m, n, k, nnz_A, nnz_B$ or $nnz_D$ is invalid.
- `rocsparse_status_memory_error`: additional buffer for long rows could not be allocated.
- `rocsparse_status_not_implemented`: $trans_A != rocsparse_operation_none, trans_B != rocsparse_operation_none$, or `rocsparse_matrix_type != rocsparse_matrix_type_general`. 

---

2.12. ROCm Libraries
2.12.5.11.3 rocsparse_csrgemm()

rocsparse_status rocsparse_csrgemm(rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, const float *alpha, const rocsparse_mat_descr descr_A, rocsparse_int nnz_A, const float *csr_val_A, const rocsparse_int *csr_row_ptr_A, const rocsparse_int *csr_col_ind_A, const rocsparse_mat_descr descr_B, rocsparse_int nnz_B, const float *csr_val_B, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const float *beta, const rocsparse_mat_descr descr_D, rocsparse_int nnz_D, const float *csr_val_D, const rocsparse_int *csr_row_ptr_D, const rocsparse_int *csr_col_ind_D, const float *csr_val_C, const rocsparse_int *csr_row_ptr_C, rocsparse_int *csr_col_ind_C, const rocsparse_mat_info info_C, void *temp_buffer)

Sparse matrix sparse matrix multiplication using CSR storage format.

rocsparse_csrgemm multiplies the scalar $\alpha$ with the sparse $m \times k$ matrix $A$, defined in CSR storage format, and the sparse $k \times n$ matrix $B$, defined in CSR storage format, and adds the result to the sparse $m \times n$ matrix $D$ that is multiplied by $\beta$. The final result is stored in the sparse $m \times n$ matrix $C$, defined in CSR storage format, such that

$$C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot D,$$

with

$$\text{op}(A) = \begin{cases} A, & \text{if } \text{trans}_A == \text{rocsparse_operation_none} \\ A^T, & \text{if } \text{trans}_A == \text{rocsparse_operation_transpose} \\ A^H, & \text{if } \text{trans}_A == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

and

$$\text{op}(B) = \begin{cases} B, & \text{if } \text{trans}_B == \text{rocsparse_operation_none} \\ B^T, & \text{if } \text{trans}_B == \text{rocsparse_operation_transpose} \\ B^H, & \text{if } \text{trans}_B == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

It is assumed that csr_row_ptr_C has already been filled and that csr_val_C and csr_col_ind_C are allocated by the user. csr_row_ptr_C and allocation size of csr_col_ind_C and csr_val_C is defined by the number of non-zero elements of the sparse CSR matrix C. Both can be obtained.
by \texttt{rocsparse_csrgemm\_nnz()}. The required buffer size for the computation can be obtained by \texttt{rocsparse\_csrgemm\_buffer\_size()}, \texttt{rocsparse\_desrgemm\_buffer\_size()}, \texttt{rocsparse\_csrsrgemm\_buffer\_size()} and \texttt{rocsparse\_zcsrgemm\_buffer\_size()}, respectively.

\textbf{Note} If $\alpha = 0$, then $C = \beta \cdot D$ will be computed.

\textbf{Note} If $\beta = 0$, then $C = \alpha \cdot \text{op}(A) \cdot \text{op}(B)$ will be computed.

\textbf{Note} $\alpha = \beta = 0$ is invalid.

\textbf{Note} Currently, only \texttt{trans\_A == rocsparse\_operation\_none} is supported.

\textbf{Note} Currently, only \texttt{trans\_B == rocsparse\_operation\_none} is supported.

\textbf{Note} Currently, only \texttt{rocsparse\_matrix\_type\_general} is supported.

\textbf{Note} This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

\textbf{Note} Please note, that for matrix products with more than 4096 non-zero entries per row, additional temporary storage buffer is allocated by the algorithm.

\textbf{Example} This example multiplies two CSR matrices with a scalar alpha and adds the result to another CSR matrix.

\begin{verbatim}
// Initialize scalar multipliers
float alpha = 2.0f;
float beta = 1.0f;

// Create matrix descriptors
rocsparse_mat_descr descr_A;
rocsparse_mat_descr descr_B;
rocsparse_mat_descr descr_C;
rocsparse_mat_descr descr_D;
rocsparse_create_mat_descr(&descr_A);
rocsparse_create_mat_descr(&descr_B);
rocsparse_create_mat_descr(&descr_C);
rocsparse_create_mat_descr(&descr_D);

// Create matrix info structure
rocsparse_mat_info info_C;
rocsparse_create_mat_info(&info_C);

// Set pointer mode
rocsparse_set_pointer_mode(handle, rocsparse_pointer_mode_host);

// Query rocsparse for the required buffer size
size_t buffer_size;
rocsparse_scsrgemm_buffer_size(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    &alpha,
    descr_A,
    nnz_A,
    csr_row_ptr_A,
    (continues on next page)
\end{verbatim}
(continued from previous page)

csr_col_ind_A,
descr_B,
nnz_B,
csr_row_ptr_B,
csr_col_ind_B,
\beta,
descr_D,
nnz_D,
csr_row_ptr_D,
csr_col_ind_D,
info_C,
\&buffer_size);

// Allocate buffer
void* buffer;
hipMalloc(&buffer, buffer_size);

// Obtain number of total non-zero entries in C and row pointers of C
rocsparse_int nnz_C;
hipMalloc((void**)&csr_row_ptr_C, sizeof(rocsparse_int) * (m + 1));

rocsparse_csrgemm_nnz(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    descr_A,
    nnz_A,
    csr_row_ptr_A,
    csr_col_ind_A,
    descr_B,
    nnz_B,
    csr_row_ptr_B,
    csr_col_ind_B,
    descr_D,
    nnz_D,
    csr_row_ptr_D,
    csr_col_ind_D,
    descr_C,
    csr_row_ptr_C,
    \&nnz_C,
    info_C,
    buffer);

// Compute column indices and values of C
hipMalloc((void**)&csr_col_ind_C, sizeof(rocsparse_int) * nnz_C);
hipMalloc((void**)&csr_val_C, sizeof(float) * nnz_C);

rocsparse_scsrgemm(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    &alpha,
    descr_A,
    (continues on next page)
nnz_A,
csr_val_A,
csr_row_ptr_A,
csr_col_ind_A,
descr_B,
nnz_B,
csr_val_B,
csr_row_ptr_B,
csr_col_ind_B,
\beta,
descr_D,
nnz_D,
csr_val_D,
csr_row_ptr_D,
csr_col_ind_D,
descr_C,
csr_val_C,
csr_row_ptr_C,
csr_col_ind_C,
info_C,
buffer);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix $A$ operation type.
- [in] trans_B: matrix $B$ operation type.
- [in] m: number of rows of the sparse CSR matrix $op(A)$ and $C$.
- [in] n: number of columns of the sparse CSR matrix $op(B)$ and $C$.
- [in] k: number of columns of the sparse CSR matrix $op(A)$ and number of rows of the sparse CSR matrix $op(B)$.
- [in] alpha: scalar $\alpha$.
- [in] descr_A: descriptor of the sparse CSR matrix $A$. Currently, only \texttt{rocsparse_matrix_type_general} is supported.
- [in] nnz_A: number of non-zero entries of the sparse CSR matrix $A$.
- [in] csr_val_A: array of $nnz_A$ elements of the sparse CSR matrix $A$.
- [in] csr_row_ptr_A: array of $m+1$ elements ($op(A) == A, k+1$ otherwise) that point to the start of every row of the sparse CSR matrix $op(A)$.
- [in] csr_col_ind_A: array of $nnz_A$ elements containing the column indices of the sparse CSR matrix $A$.
- [in] descr_B: descriptor of the sparse CSR matrix $B$. Currently, only \texttt{rocsparse_matrix_type_general} is supported.
- [in] nnz_B: number of non-zero entries of the sparse CSR matrix $B$.
- [in] csr_val_B: array of $nnz_B$ elements of the sparse CSR matrix $B$.
- [in] csr_row_ptr_B: array of $k+1$ elements ($op(B) == B, m+1$ otherwise) that point to the start of every row of the sparse CSR matrix $op(B)$.
• [in] csr_col_ind_B: array of nnz_B elements containing the column indices of the sparse CSR matrix B.
• [in] beta: scalar $\beta$.
• [in] descr_D: descriptor of the sparse CSR matrix D. Currently, only `roc-sparse_matrix_type_general` is supported.
• [in] nnz_D: number of non-zero entries of the sparse CSR matrix D.
• [in] csr_val_D: array of nnz_D elements of the sparse CSR matrix D.
• [in] csr_row_ptr_D: array of m+1 elements that point to the start of every row of the sparse CSR matrix D.
• [in] csr_col_ind_D: array of nnz_D elements containing the column indices of the sparse CSR matrix D.
• [in] descr_C: descriptor of the sparse CSR matrix C. Currently, only `roc-sparse_matrix_type_general` is supported.
• [in] csr_val_C: array of nnz_C elements of the sparse CSR matrix C.
• [in] csr_row_ptr_C: array of m+1 elements that point to the start of every row of the sparse CSR matrix C.
• [out] csr_col_ind_C: array of nnz_C elements containing the column indices of the sparse CSR matrix C.
• [in] info_C: structure that holds meta data for the sparse CSR matrix C.
• [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by `rocsparse_scsrgemm_buffer_size()`, `rocsparse_dcsrgemm_buffer_size()`, `rocsparse_ccsrgemm_buffer_size()` or `rocsparse_zcsrgemm_buffer_size()`.

Return Value

• `rocsparse_status_success`: the operation completed successfully.
• `rocsparse_status_invalid_handle`: the library context was not initialized.
• `rocsparse_status_invalid_size`: m, n, k, nnz_A, nnz_B or nnz_D is invalid.
• `rocsparse_status_invalid_pointer`: alpha and beta are invalid, descr_A, csr_val_A, csr_row_ptr_A, csr_col_ind_A, descr_B, csr_val_B, csr_row_ptr_B or csr_col_ind_B are invalid if alpha is valid, descr_D, csr_val_D, csr_row_ptr_D or csr_col_ind_D is invalid if beta is valid, csr_val_C, csr_row_ptr_C, csr_col_ind_C, info_C or temp_buffer is invalid.
• `rocsparse_status_memory_error`: additional buffer for long rows could not be allocated.
• `rocsparse_status_not_implemented`: trans_A $\neq$ `rocsparse_operation_none`, trans_B $\neq$ `rocsparse_operation_none`, or `rocsparse_matrix_type $\neq$ `roc-sparse_matrix_type_general`. 

348 Chapter 2. Solid Compilation Foundation and Language Support
rocsparse_status rocsparse_ccsrgemm (rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, const rocsparse_float_complex *alpha, const rocsparse_mat_descr descr_A, rocsparse_int nnz_A, const rocsparse_float_complex *csr_val_A, const rocsparse_int *csr_row_ptr_A, const rocsparse_int *csr_col_ind_A, const rocsparse_float_complex *csr_val_B, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const rocsparse_float_complex *beta, const rocsparse_mat_descr descr_B, rocsparse_int nnz_B, const rocsparse_float_complex *csr_val_B, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const rocsparse_float_complex *csr_val_D, const rocsparse_int *csr_row_ptr_D, const rocsparse_int *csr_col_ind_D, const rocsparse_float_complex *csr_val_C, const rocsparse_int *csr_row_ptr_C, const rocsparse_int *csr_col_ind_C, const rocsparse_mat_info info_C, void *temp_buffer)

Sparse matrix sparse matrix multiplication using CSR storage format.

rocsparse_ccsrgemm multiplies the scalar $\alpha$ with the sparse $m \times k$ matrix $A$, defined in CSR storage format, and the sparse $k \times n$ matrix $B$, defined in CSR storage format, and adds the result to the sparse $m \times n$ matrix $D$ that is multiplied by $\beta$. The final result is stored in the sparse $m \times n$ matrix $C$, defined in CSR storage format, such that

$$C := \alpha \cdot op(A) \cdot op(B) + \beta \cdot D,$$

with

$$op(A) = \begin{cases} A, & \text{if} \ trans_A == \text{rocsparse_operation_none} \\ A^T, & \text{if} \ trans_A == \text{rocsparse_operation_transpose} \\ A^H, & \text{if} \ trans_A == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

and

$$op(B) = \begin{cases} B, & \text{if} \ trans_B == \text{rocsparse_operation_none} \\ B^T, & \text{if} \ trans_B == \text{rocsparse_operation_transpose} \\ B^H, & \text{if} \ trans_B == \text{rocsparse_operation_conjugate_transpose} \end{cases}$$

It is assumed that $csr\_row\_ptr\_C$ has already been filled and that $csr\_val\_C$ and $csr\_col\_ind\_C$ are allocated by the user. $csr\_row\_ptr\_C$ and allocation size of $csr\_col\_ind\_C$ and $csr\_val\_C$ is defined by the number of non-zero elements of the sparse CSR matrix C. Both can be obtained by $rocsparse\_ccsrgemm\_nnz()$. The required buffer size for the computation can be obtained by $rocsparse\_ccsrgemm\_buffer\_size()$, $rocsparse\_dcsrgemm\_buffer\_size()$, $rocsparse\_ccsrgemm\_buffer\_size()$ and $rocsparse\_zcsrgemm\_buffer\_size()$, respectively.

**Note** If $\alpha == 0$, then $C = \beta \cdot D$ will be computed.

**Note** If $\beta == 0$, then $C = \alpha \cdot op(A) \cdot op(B)$ will be computed.

**Note** $\alpha == beta == 0$ is invalid.

**Note** Currently, only $trans\_A == \text{rocsparse\_operation\_none}$ is supported.

**Note** Currently, only $trans\_B == \text{rocsparse\_operation\_none}$ is supported.

**Note** Currently, only $\text{rocsparse\_matrix\_type\_general}$ is supported.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.
Note Please note, that for matrix products with more than 4096 non-zero entries per row, additional temporary storage buffer is allocated by the algorithm.

Example This example multiplies two CSR matrices with a scalar alpha and adds the result to another CSR matrix.

```c
// Initialize scalar multipliers
float alpha = 2.0f;
float beta = 1.0f;

// Create matrix descriptors
rocsp_mat_descr descr_A;
rocsp_mat_descr descr_B;
rocsp_mat_descr descr_C;
rocsp_mat_descr descr_D;

rocsp_create_mat_descr(&descr_A);
rocsp_create_mat_descr(&descr_B);
rocsp_create_mat_descr(&descr_C);
rocsp_create_mat_descr(&descr_D);

// Create matrix info structure
rocsp_mat_info info_C;
rocsp_create_mat_info(&info_C);

// Set pointer mode
rocsp_set_pointer_mode(handle, rocsp_pointer_mode_host);

// Query rocsp for the required buffer size
size_t buffer_size;
rocsp_scsrgemm_buffer_size(handle,
    rocsp_operation_none,
    rocsp_operation_none,
    m,
    n,
    k,
    &alpha,
    descr_A,
    nnz_A,
    csr_row_ptr_A,
    csr_col_ind_A,
    descr_B,
    nnz_B,
    csr_row_ptr_B,
    csr_col_ind_B,
    &beta,
    descr_D,
    nnz_D,
    csr_row_ptr_D,
    csr_col_ind_D,
    info_C,
    &buffer_size);

// Allocate buffer
void* buffer;
hipMalloc(&buffer, buffer_size);
```

(continues on next page)
// Obtain number of total non-zero entries in C and row pointers of C
rocsparse_int nnz_C;
hipMalloc((void**)&csr_row_ptr_C, sizeof(rocsparse_int) * (m + 1));

rocsparse_cscrgemm_nnz(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    descr_A,
    nnz_A,
    csr_row_ptr_A,
    csr_col_ind_A,
    descr_B,
    nnz_B,
    csr_row_ptr_B,
    csr_col_ind_B,
    descr_D,
    nnz_D,
    csr_row_ptr_D,
    csr_col_ind_D,
    descr_C,
    csr_row_ptr_C,
    &nnz_C,
    info_C,
    buffer);

// Compute column indices and values of C
hipMalloc((void**)&csr_col_ind_C, sizeof(rocsparse_int) * nnz_C);
hipMalloc((void**)&csr_val_C, sizeof(float) * nnz_C);

rocsparse_scsrgemm(handle,
    rocsparse_operation_none,
    rocsparse_operation_none,
    m,
    n,
    k,
    &alpha,
    descr_A,
    nnz_A,
    csr_val_A,
    csr_row_ptr_A,
    csr_col_ind_A,
    descr_B,
    nnz_B,
    csr_val_B,
    csr_row_ptr_B,
    csr_col_ind_B,
    &beta,
    descr_D,
    nnz_D,
    csr_val_D,
    csr_row_ptr_D,
    csr_col_ind_D,
    descr_C,
    csr_val_C,
Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix $A$ operation type.
- [in] trans_B: matrix $B$ operation type.
- [in] m: number of rows of the sparse CSR matrix $op(A)$ and $C$.
- [in] n: number of columns of the sparse CSR matrix $op(B)$ and $C$.
- [in] k: number of columns of the sparse CSR matrix $op(A)$ and number of rows of the sparse CSR matrix $op(B)$.
- [in] alpha: scalar $\alpha$.
- [in] descr_A: descriptor of the sparse CSR matrix $A$. Currently, only rocsparse_matrix_type_general is supported.
- [in] nnz_A: number of non-zero entries of the sparse CSR matrix $A$.
- [in] csr_val_A: array of nnz_A elements of the sparse CSR matrix $A$.
- [in] csr_row_ptr_A: array of m+1 elements ($op(A) == A, k+1$ otherwise) that point to the start of every row of the sparse CSR matrix $op(A)$.
- [in] csr_col_ind_A: array of nnz_A elements containing the column indices of the sparse CSR matrix $A$.
- [in] descr_B: descriptor of the sparse CSR matrix $B$. Currently, only rocsparse_matrix_type_general is supported.
- [in] nnz_B: number of non-zero entries of the sparse CSR matrix $B$.
- [in] csr_val_B: array of nnz_B elements of the sparse CSR matrix $B$.
- [in] csr_row_ptr_B: array of k+1 elements ($op(B) == B, m+1$ otherwise) that point to the start of every row of the sparse CSR matrix $op(B)$.
- [in] csr_col_ind_B: array of nnz_B elements containing the column indices of the sparse CSR matrix $B$.
- [in] beta: scalar $\beta$.
- [in] descr_D: descriptor of the sparse CSR matrix $D$. Currently, only rocsparse_matrix_type_general is supported.
- [in] nnz_D: number of non-zero entries of the sparse CSR matrix $D$.
- [in] csr_row_ptr_D: array of m+1 elements that point to the start of every row of the sparse CSR matrix $D$.
- [in] csr_col_ind_D: array of nnz_D elements containing the column indices of the sparse CSR matrix $D$.
- [in] descr_C: descriptor of the sparse CSR matrix $C$. Currently, only rocsparse_matrix_type_general is supported.
• [out] csr_val_C: array of nnz_C elements of the sparse CSR matrix C.
• [in] csr_row_ptr_C: array of m+1 elements that point to the start of every row of the sparse CSR matrix C.
• [out] csr_col_ind_C: array of nnz_C elements containing the column indices of the sparse CSR matrix C.
• [in] info_C: structure that holds meta data for the sparse CSR matrix C.
• [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by rocsparse_scsrgemm_buffer_size(), rocsparse_dcsrgemm_buffer_size(), rocsparse_ccsrgemm_buffer_size() or rocsparse_zcsrgemm_buffer_size().

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n, k, nnz_A, nnz_B or nnz_D is invalid.
• rocsparse_status_invalid_pointer: alpha and beta are invalid, descr_A, csr_val_A, csr_row_ptr_A, csr_col_ind_A, descr_B, csr_val_B, csr_row_ptr_B or csr_col_ind_B are invalid if alpha is valid, descr_D, csr_val_D, csr_row_ptr_D or csr_col_ind_D is invalid if beta is valid, csr_val_C, csr_row_ptr_C, csr_col_ind_C, info_C or temp_buffer is invalid.
• rocsparse_status_memory_error: additional buffer for long rows could not be allocated.
• rocsparse_status_not_implemented: trans_A != rocsparse_operation_none, trans_B != rocsparse_operation_none, or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_zcsrgemm(rocsparse_handle handle, rocsparse_operation trans_A, rocsparse_operation trans_B, rocsparse_int m, rocsparse_int n, rocsparse_int k, const rocsparse_double_complex *alpha, const rocsparse_mat_descr descr_A, rocsparse_int nnz_A, const rocsparse_double_complex *csr_val_A, const rocsparse_int *csr_row_ptr_A, const rocsparse_int *csr_col_ind_A, const rocsparse_mat_descr descr_B, rocsparse_int nnz_B, const rocsparse_double_complex *csr_val_B, const rocsparse_int *csr_row_ptr_B, const rocsparse_int *csr_col_ind_B, const rocsparse_double_complex *beta, const rocsparse_mat_descr descr_D, rocsparse_int nnz_D, const rocsparse_double_complex *csr_val_D, const rocsparse_int *csr_row_ptr_D, const rocsparse_int *csr_col_ind_D, const rocsparse_mat_descr descr_C, rocsparse_double_complex *csr_val_C, const rocsparse_int *csr_row_ptr_C, const rocsparse_int *csr_col_ind_C, const rocsparse_mat_info info_C, void *temp_buffer)

Sparse matrix sparse matrix multiplication using CSR storage format.

rocsparse_csrgemm multiplies the scalar \( \alpha \) with the sparse \( m \times k \) matrix \( A \), defined in CSR storage format, and the sparse \( k \times n \) matrix \( B \), defined in CSR storage format, and adds the result to the sparse \( m \times n \) matrix \( D \) that is multiplied by \( \beta \). The final result is stored in the sparse \( m \times n \) matrix \( C \), defined in CSR storage format, such that

\[
C := \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot D,
\]
with

\[ op(A) = \begin{cases} 
A, & \text{if trans}_A == \text{rocsparse\_operation\_none} \\
A^T, & \text{if trans}_A == \text{rocsparse\_operation\_transpose} \\
A^H, & \text{if trans}_A == \text{rocsparse\_operation\_conjugate\_transpose} 
\end{cases} \]

and

\[ op(B) = \begin{cases} 
B, & \text{if trans}_B == \text{rocsparse\_operation\_none} \\
B^T, & \text{if trans}_B == \text{rocsparse\_operation\_transpose} \\
B^H, & \text{if trans}_B == \text{rocsparse\_operation\_conjugate\_transpose} 
\end{cases} \]

It is assumed that \texttt{csr\_row\_ptr\_C} has already been filled and that \texttt{csr\_val\_C} and \texttt{csr\_col\_ind\_C} are allocated by the user. \texttt{csr\_row\_ptr\_C} and allocation size of \texttt{csr\_col\_ind\_C} and \texttt{csr\_val\_C} is defined by the number of non-zero elements of the sparse CSR matrix \( C \). Both can be obtained by \texttt{rocsparse\_cscrgemm\_nnz()}\). The required buffer size for the computation can be obtained by \texttt{rocsparse\_dcsrgemm\_buffer\_size()}, \texttt{rocsparse\_ccsrgemm\_buffer\_size()}, \texttt{rocsparse\_ccsrgemm\_buffer\_size()} and \texttt{rocsparse\_ccsrgemm\_buffer\_size()} respectively.

\textbf{Note} If \( \alpha == 0 \), then \( C = \beta \cdot D \) will be computed.

\textbf{Note} If \( \beta == 0 \), then \( C = \alpha \cdot op(A) \cdot op(B) \) will be computed.

\textbf{Note} \( \alpha == beta == 0 \) is invalid.

\textbf{Note} Currently, only \texttt{trans\_A == rocsparse\_operation\_none} is supported.

\textbf{Note} Currently, only \texttt{trans\_B == rocsparse\_operation\_none} is supported.

\textbf{Note} Currently, only \texttt{rocsparse\_matrix\_type\_general} is supported.

\textbf{Note} This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

\textbf{Note} Please note, that for matrix products with more than 4096 non-zero entries per row, additional temporary storage buffer is allocated by the algorithm.

\textbf{Example} This example multiplies two CSR matrices with a scalar alpha and adds the result to another CSR matrix.

```c
// Initialize scalar multipliers
float alpha = 2.0f;
float beta  = 1.0f;

// Create matrix descriptors
rocsparse_mat_descr descr_A;
rocsparse_mat_descr descr_B;
rocsparse_mat_descr descr_C;
rocsparse_mat_descr descr_D;
rocsparse_create_mat_descr(&descr_A);
rocsparse_create_mat_descr(&descr_B);
rocsparse_create_mat_descr(&descr_C);
rocsparse_create_mat_descr(&descr_D);

// Create matrix info structure
rocsparse_mat_info info_C;
rocsparse_create_mat_info(&info_C);

// Set pointer mode
```

(continues on next page)
rocspure_set_pointer_mode(handle, rocspure_pointer_mode_host);

// Query rocspure for the required buffer size
size_t buffer_size;

rocspure_scsrgemm_buffer_size(handle,
    rocspure_operation_none,
    rocspure_operation_none,
    m, n, k, &alpha,
    descr_A, nnz_A,
    csr_row_ptr_A, csr_col_ind_A,
    descr_B, nnz_B,
    csr_row_ptr_B, csr_col_ind_B,
    &beta, descr_D, nnz_D,
    csr_row_ptr_D, csr_col_ind_D,
    info_C, &buffer_size);

// Allocate buffer
void* buffer;
hipMalloc(&buffer, buffer_size);

// Obtain number of total non-zero entries in C and row pointers of C
rocspure_int nnz_C;
hipMalloc((void**)&csr_row_ptr_C, sizeof(rocspure_int) * (m + 1));

rocspure_scsrgemm_nnz(handle,
    rocspure_operation_none,
    rocspure_operation_none,
    m, n, k,
    descr_A, nnz_A,
    csr_row_ptr_A, csr_col_ind_A,
    descr_B, nnz_B,
    csr_row_ptr_B, csr_col_ind_B,
    descr_D, nnz_D,
    csr_row_ptr_D, csr_col_ind_D,
    descr_C, csr_row_ptr_C, &nnz_C,
    &nnz_C, (continues on next page)
Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] trans_A: matrix A operation type.
- [in] trans_B: matrix B operation type.
- [in] m: number of rows of the sparse CSR matrix \(op(A)\) and \(C\).
- [in] n: number of columns of the sparse CSR matrix \(op(B)\) and \(C\).
- [in] k: number of columns of the sparse CSR matrix \(op(A)\) and number of rows of the sparse CSR matrix \(op(B)\).
- [in] alpha: scalar \(\alpha\).
- [in] descr_A: descriptor of the sparse CSR matrix A. Currently, only rocsparse_matrix_type_general is supported.
- [in] nnz_A: number of non-zero entries of the sparse CSR matrix A.
- [in] csr_val_A: array of nnz_A elements of the sparse CSR matrix A.
• [in] csr_row_ptr_A: array of $m+1$ elements (\( op(A) == A, k+1 \) otherwise) that point to the start of every row of the sparse CSR matrix \( op(A) \).

• [in] csr_col_ind_A: array of \( nnz_A \) elements containing the column indices of the sparse CSR matrix \( A \).

• [in] descr_B: descriptor of the sparse CSR matrix \( B \). Currently, only \texttt{roc-sparse_matrix_type_general} is supported.

• [in] \( nnz_B \): number of non-zero entries of the sparse CSR matrix \( B \).

• [in] csr_val_B: array of \( nnz_B \) elements of the sparse CSR matrix \( B \).

• [in] csr_row_ptr_B: array of \( k+1 \) elements (\( op(B) == B, m+1 \) otherwise) that point to the start of every row of the sparse CSR matrix \( op(B) \).

• [in] csr_col_ind_B: array of \( nnz_B \) elements containing the column indices of the sparse CSR matrix \( B \).

• [in] beta: scalar \( \beta \).

• [in] descr_D: descriptor of the sparse CSR matrix \( D \). Currently, only \texttt{roc-sparse_matrix_type_general} is supported.

• [in] \( nnz_D \): number of non-zero entries of the sparse CSR matrix \( D \).

• [in] csr_val_D: array of \( nnz_D \) elements of the sparse CSR matrix \( D \).

• [in] csr_row_ptr_D: array of \( m+1 \) elements that point to the start of every row of the sparse CSR matrix \( D \).

• [in] csr_col_ind_D: array of \( nnz_D \) elements containing the column indices of the sparse CSR matrix \( D \).

• [in] descr_C: descriptor of the sparse CSR matrix \( C \). Currently, only \texttt{roc-sparse_matrix_type_general} is supported.

• [out] csr_val_C: array of \( nnz_C \) elements of the sparse CSR matrix \( C \).

• [in] csr_row_ptr_C: array of \( m+1 \) elements that point to the start of every row of the sparse CSR matrix \( C \).

• [out] csr_col_ind_C: array of \( nnz_C \) elements containing the column indices of the sparse CSR matrix \( C \).

• [in] info_C: structure that holds meta data for the sparse CSR matrix \( C \).

• [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by \texttt{rocsparse_scsrgemm_buffer_size()}, \texttt{rocsparse_dcsrgemm_buffer_size()}, \texttt{rocsparse_ccsrgemm_buffer_size()} or \texttt{rocsparse_zcsrgemm_buffer_size()}.

Return Value

• \texttt{rocsparse_status_success}: the operation completed successfully.

• \texttt{rocsparse_status_invalid_handle}: the library context was not initialized.

• \texttt{rocsparse_status_invalid_size}: \( m, n, k, nnz_A, nnz_B \) or \( nnz_D \) is invalid.

• \texttt{rocsparse_status_invalid_pointer}: alpha and beta are invalid, descr_A, csr_val_A, csr_row_ptr_A, csr_col_ind_A, descr_B, csr_val_B, csr_row_ptr_B or csr_col_ind_B are invalid if alpha is valid, descr_D, csr_val_D, csr_row_ptr_D or csr_col_ind_D is invalid if beta is valid, csr_val_C, csr_row_ptr_C, csr_col_ind_C, info_C or temp_buffer is invalid.

• \texttt{rocsparse_status_memory_error}: additional buffer for long rows could not be allocated.
• rocsparse_status_not_implemented: trans_A != rocsparse_operation_none, trans_B != rocsparse_operation_none, or rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.12 Preconditioner Functions

This module holds all sparse preconditioners.

The sparse preconditioners describe manipulations on a matrix in sparse format to obtain a sparse preconditioner matrix.

2.12.5.12.1 rocsparse_csrilu0_zero_pivot()

rocsparse_status rocsparse_csrilu0_zero_pivot (rocsparse_handle handle, rocsparse_mat_info info, rocsparse_int *position)

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrilu0_zero_pivot returns rocsparse_status_zero_pivot, if either a structural or numerical zero has been found during rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_ccsrilu0() or rocsparse_zcsrilu0() computation. The first zero pivot \( j \) at \( A_{j,j} \) is stored in position, using same index base as the CSR matrix.

position can be in host or device memory. If no zero pivot has been found, position is set to -1 and rocsparse_status_success is returned instead.

Note rocsparse_csrilu0_zero_pivot is a blocking function. It might influence performance negatively.

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] info: structure that holds the information collected during the analysis step.
• [inout] position: pointer to zero pivot \( j \), can be in host or device memory.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_pointer: info or position pointer is invalid.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_zero_pivot: zero pivot has been found.

2.12.5.12.2 rocsparse_csrilu0_buffer_size()

rocsparse_status rocsparse_scscrilu0_buffer_size (rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, size_t *buffer_size)
Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrilu0_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis(), rocsparse_zcsrilu0_analysis(), rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_ccsrilu0() and rocsparse_zcsrilu0(). The temporary storage buffer must be allocated by the user. The size of the temporary storage buffer is identical to the size returned by rocsparse_scsrsv_buffer_size(), rocsparse_dcsrsv_buffer_size(), rocsparse_ccsrsv_buffer_size() and rocsparse_zcsrsv_buffer_size() if the matrix sparsity pattern is identical. The user allocated buffer can thus be shared between subsequent calls to those functions.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.
- [in] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis(), rocsparse_zcsrilu0_analysis(), rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_ccsrilu0() and rocsparse_zcsrilu0().

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind, info or buffer_size pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.
Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrilu0_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis(), rocsparse_zcsrilu0_analysis(), rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_ccsrilu0() and rocsparse_zcsrilu0(). The temporary storage buffer must be allocated by the user. The size of the temporary storage buffer is identical to the size returned by rocsparse_scsrsv_buffer_size(), rocsparse_dcsrsv_buffer_size(), rocsparse_ccsrsv_buffer_size() and rocsparse_zcsrsv_buffer_size() if the matrix sparsity pattern is identical. The user allocated buffer can thus be shared between subsequent calls to those functions.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.
- [in] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis(), rocsparse_zcsrilu0_analysis(), rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_ccsrilu0() and rocsparse_zcsrilu0().

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind, info or buffer_size pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_zcsrilu0_buffer_size (rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, size_t *buffer_size)

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrilu0_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis(), rocsparse_zcsrilu0_analysis(), rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_ccsrilu0() and rocsparse_zcsrilu0(). The temporary storage buffer must be allocated by the user. The size of the temporary storage...
buffer is identical to the size returned by `rocsparse_scsrsrv_buffer_size()`, `rocsparse_dcsrsrv_buffer_size()`, `rocsparse_cscrsrv_buffer_size()` and `rocsparse_zcsrsrv_buffer_size()` if the matrix sparsity pattern is identical. The user allocated buffer can thus be shared between subsequent calls to those functions.

**Parameters**

- **[in]** `handle`: handle to the rocsparse library context queue.
- **[in]** `m`: number of rows of the sparse CSR matrix.
- **[in]** `nnz`: number of non-zero entries of the sparse CSR matrix.
- **[in]** `descr`: descriptor of the sparse CSR matrix.
- **[in]** `csr_val`: array of `nnz` elements of the sparse CSR matrix.
- **[in]** `csr_row_ptr`: array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
- **[in]** `csr_col_ind`: array of `nnz` elements containing the column indices of the sparse CSR matrix.
- **[out]** `info`: structure that holds the information collected during the analysis step.
- **[in]** `buffer_size`: number of bytes of the temporary storage buffer required by `rocsparse_scsrilu0_analysis()`, `rocsparse_dcsrilu0_analysis()`, `rocsparse_cscsrilu0_analysis()`, `rocsparse_zcsrilu0_analysis()`, `rocsparse_scsrilu0()`, `rocsparse_dcsrilu0()`, `rocsparse_cscsrilu0()` and `rocsparse_zcsrilu0()`.

**Return Value**

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: `m` or `nnz` is invalid.
- `rocsparse_status_invalid_pointer`: `descr`, `csr_val`, `csr_row_ptr`, `csr_col_ind`, `info` or `buffer_size` pointer is invalid.
- `rocsparse_status_internal_error`: an internal error occurred.
- `rocsparse_status_not_implemented`: `trans` != `rocsparse_operation_none` or `rocsparse_matrix_type` != `rocsparse_matrix_type_general`.

### 2.12.5.12.3 rocsparse_cscsrilu0_analysis()

```c
rocsparse_status rocsparse_scsrilu0_analysis(rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, rocsparse_analysis_policy analysis, rocsparse_solve_policy solve, void *temp_buffer)
```

```c
rocsparse_status rocsparse_dcsrilu0_analysis(rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const double *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, rocsparse_analysis_policy analysis, rocsparse_solve_policy solve, void *temp_buffer)
```
Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

*rocsparse_csrilu0_analysis* performs the analysis step for *rocsparse_scsrilu0()*,**rocsparse_dcsrilu0()**, *rocsparse_ccsrilu0()*, and *rocsparse_zcsrilu0()*. It is expected that this function will be executed only once for a given matrix and particular operation type. The analysis meta data can be cleared by *rocsparse_csrilu0_clear()*. 

*rocsparse_csrilu0_analysis* can share its meta data with *rocsparse_scsrsrv_analysis()*, *rocsparse_dcsrsrv_analysis()*, *rocsparse_ccscrsrv_analysis()*, and *rocsparse_zcsrsrv_analysis()*. Selecting *rocsparse_analysis_policy_force* policy can greatly improve computation performance of meta data. However, the user need to make sure that the sparsity pattern remains unchanged. If this cannot be assured, *rocsparse_analysis_policy_reuse* has to be used.

**Note** If the matrix sparsity pattern changes, the gathered information will become invalid.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Parameters**

- **[in]** `handle`: handle to the rocsparse library context queue.
- **[in]** `m`: number of rows of the sparse CSR matrix.
- **[in]** `nnz`: number of non-zero entries of the sparse CSR matrix.
- **[in]** `descr`: descriptor of the sparse CSR matrix.
- **[in]** `csr_val`: array of `nnz` elements of the sparse CSR matrix.
- **[in]** `csr_row_ptr`: array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
- **[in]** `csr_col_ind`: array of `nnz` elements containing the column indices of the sparse CSR matrix.
- **[out]** `info`: structure that holds the information collected during the analysis step.
- **[in]** `analysis`: `rocsparse_analysis_policy_reuse` or `rocsparse_analysis_policy_force`.
- **[in]** `solve`: `rocsparse_solve_policy_auto`.
- **[in]** `temp_buffer`: temporary storage buffer allocated by the user.

**Return Value**

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: `m` or `nnz` is invalid.
- `rocsparse_status_invalid_pointer`: `descr`, `csr_val`, `csr_row_ptr`, `csr_col_ind`, `info` or `temp_buffer` pointer is invalid.
- `rocsparse_status_internal_error`: an internal error occurred.
- `rocsparse_status_not_implemented`: `trans` != `rocsparse_operation_none` or `rocsparse_matrix_type` != `rocsparse_matrix_type_general`.
In incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrilu0_analysis performs the analysis step for rocsparse_scsrilu0(), rocsparse_dcsrilu0(), rocsparse_cscsrlu0(), and rocsparse_zcsrilu0(). It is expected that this function will be executed only once for a given matrix and particular operation type. The analysis meta data can be cleared by rocsparse_csrilu0_clear().

rocsparse_csrilu0_analysis can share its meta data with rocsparse_scsrsv_analysis(), rocsparse_dcsrsv_analysis(), rocsparse_ccsrsv_analysis(), and rocsparse_zcsrsv_analysis(). Selecting rocsparse_analysis_policy_reuse policy can greatly improve computation performance of meta data. However, the user need to make sure that the sparsity pattern remains unchanged. If this cannot be assured, rocsparse_analysis_policy_force has to be used.

Note If the matrix sparsity pattern changes, the gathered information will become invalid.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** m: number of rows of the sparse CSR matrix.
- **[in]** nnz: number of non-zero entries of the sparse CSR matrix.
- **[in]** descr: descriptor of the sparse CSR matrix.
- **[in]** csr_val: array of nnz elements of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[in]** csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- **[out]** info: structure that holds the information collected during the analysis step.
- **[in]** analysis: rocsparse_analysis_policy_reuse or rocsparse_analysis_policy_force.
- **[in]** solve: rocsparse_solve_policy_auto.
- **[in]** temp_buffer: temporary storage buffer allocated by the user.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind, info or temp_buffer pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.
rocsparse_status rocsparse_zcsrilu0_analysis (rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, rocsparse_analysis_policy analysis, rocsparse_solve_policy solve, void *temp_buffer)

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_zcsrilu0_analysis performs the analysis step for rocsparse_scsrilu0(), rocsparse_dcsrilu0() and rocsparse_zcsrilu0(). It is expected that this function will be executed only once for a given matrix and particular operation type. The analysis meta data can be cleared by rocsparse_csrilu0_clear().

rocsparse_zcsrilu0_analysis can share its meta data with rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis() and rocsparse_zcsrilu0_analysis(). Selecting rocsparse_analysis_policy_reuse policy can greatly improve computation performance of meta data. However, the user need to make sure that the sparsity pattern remains unchanged. If this cannot be assured, rocsparse_analysis_policy_force has to be used.

Note If the matrix sparsity pattern changes, the gathered information will become invalid.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] info: structure that holds the information collected during the analysis step.
- [in] analysis: rocsparse_analysis_policy_reuse or rocsparse_analysis_policy_force.
- [in] temp_buffer: temporary storage buffer allocated by the user.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr, csr_col_ind, info or temp_buffer pointer is invalid.
• `rocsparse_status_internal_error`: an internal error occurred.
• `rocsparse_status_not_implemented`: `trans` != `rocsparse_operation_none` or `rocsparse_matrix_type` != `rocsparse_matrix_type_general`.

2.12.5.12.4 `rocsparse_csrilu0()`

```
rocsparse_status rocsparse_csrilu0 (rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, const rocsparse_mat_descr descr, float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_mat_info info, rocsparse_solve_policy policy, void *temp_buffer)
```

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

`rocsparse_csrilu0` computes the incomplete LU factorization with 0 fill-ins and no pivoting of a sparse `m x m` CSR matrix `A`, such that

\[ A \approx LU \]

`rocsparse_csrilu0` requires a user allocated temporary buffer. Its size is returned by `rocsparse_csrilu0_buffer_size()`, `rocsparse_dcsrilu0_buffer_size()`, `rocsparse_ccsrilu0_buffer_size()` or `rocsparse_zcsrilu0_buffer_size()`. Furthermore, analysis meta data is required. It can be obtained by `rocsparse_csrilu0_analysis()`, `rocsparse_dcsrilu0_analysis()`, `rocsparse_ccsrilu0_analysis()` or `rocsparse_zcsrilu0_analysis()`. `rocsparse_csrilu0` reports the first zero pivot (either numerical or structural zero). The zero pivot status can be obtained by calling `rocsparse_csrilu0_zero_pivot()`.

**Note** The sparse CSR matrix has to be sorted. This can be achieved by calling `rocsparse_csrsort()`.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** Consider the sparse `m x m` matrix `A`, stored in CSR storage format. The following example computes the incomplete LU factorization `M \approx LU` and solves the preconditioned system `My = x`.

```
// Create rocSPARSE handle
rocsparse_handle handle;
rocsparse_create_handle(&handle);

// Create matrix descriptor for M
rocsparse_mat_descr descr_M;
rocsparse_create_mat_descr(&descr_M);

// Create matrix descriptor for L
rocsparse_mat_descr descr_L;
rocsparse_create_mat_descr(&descr_L);
rocsparse_set_mat_fill_mode(descr_L, rocsparse_fill_mode_lower);
rocsparse_set_mat_diag_type(descr_L, rocsparse_diag_type_unit);

// Create matrix descriptor for U
rocsparse_mat_descr descr_U;
```

(continues on next page)
rocspare_create_mat_descri(&descr_U);
rocspare_set_mat_fill_mode(descr_U, rocspare_fill_mode_upper);
rocspare_set_mat_diag_type(descr_U, rocspare_diag_type_non_unit);

// Create matrix info structure
rocspare_mat_info info;
rocspare_create_mat_info(&info);

// Obtain required buffer size
size_t buffer_size_M;
size_t buffer_size_L;
size_t buffer_size_U;
rocspare_dcsrilu0_buffer_size(handle,
    m,
    nnz,
    descr_M,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size_M);
rocspare_dcsrsrv_buffer_size(handle,
    rocspare_operation_none,
    m,
    nnz,
    descr_L,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size_L);
rocspare_dcsrsrv_buffer_size(handle,
    rocspare_operation_none,
    m,
    nnz,
    descr_U,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size_U);

size_t buffer_size = max(buffer_size_M, max(buffer_size_L, buffer_size_U));

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);

// Perform analysis steps, using rocspare_analysis_policy_reuse to improve
// computation performance
rocspare_dcsrilu0_analysis(handle,
    m,
    nnz,
    descr_M,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
info,
rocsparse_analysis_policy_reuse,
rocsparse_solve_policy_auto,
temp_buffer);

rocsparse_dcsrsrv_analysis(handle,
    rocsparse_operation_none,
    m,
nnz,
descr_L,
csr_val,
csr_row_ptr,
csr_col_ind,
    info,
rocsparse_analysis_policy_reuse,
rocsparse_solve_policy_auto,
temp_buffer);

rocsparse_dcsrsrv_analysis(handle,
    rocsparse_operation_none,
    m,
nnz,
descr_U,
csr_val,
csr_row_ptr,
csr_col_ind,
    info,
rocsparse_analysis_policy_reuse,
rocsparse_solve_policy_auto,
temp_buffer);

// Check for zero pivot
rocsparse_int position;
if (rocsparse_status_zero_pivot == rocsparse_csrilu0_zero_pivot(handle,
      info,
      &position))
{
    printf("A has structural zero at A(%d,%d)\n", position, position);
}

// Compute incomplete LU factorization
rocsparse_dcsrilu0(handle,
    m,
nnz,
descr_M,
csr_val,
csr_row_ptr,
csr_col_ind,
    info,
rocsparse_solve_policy_auto,
temp_buffer);

// Check for zero pivot
if (rocsparse_status_zero_pivot == rocsparse_csrilu0_zero_pivot(handle,
      info,
      &position))
{
    printf("U has structural and/or numerical zero at U(%d,%d)\n", position,
    position,
(continues on next page)
position);
}

// Solve Lz = x
rocspase_dcsrsolve(handle,
    rocspase_operation_none,
    m,
    nnz,
    &alpha,
    descr_L,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    x,
    z,
    rocspase_solve_policy_auto,
    temp_buffer);

// Solve Uy = z
rocspase_dcsrsolve(handle,
    rocspase_operation_none,
    m,
    nnz,
    &alpha,
    descr_U,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    z,
    y,
    rocspase_solve_policy_auto,
    temp_buffer);

// Clean up
hipFree(temp_buffer);
rocsparse_destroy_mat_info(info);
rocsparse_destroy_mat_descr(descr_M);
rocsparse_destroy_mat_descr(descr_L);
rocsparse_destroy_mat_descr(descr_U);
rocsparse_destroy_handle(handle);

Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** m: number of rows of the sparse CSR matrix.
- **[in]** nnz: number of non-zero entries of the sparse CSR matrix.
- **[in]** descr: descriptor of the sparse CSR matrix.
- **[inout]** csr_val: array of nnz elements of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[in]** csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
• [in] info: structure that holds the information collected during the analysis step.
• [in] policy: \texttt{rocsparse\_solve\_policy\_auto}.
• [in] temp\_buffer: temporary storage buffer allocated by the user.

\textbf{Return Value}

• rocsparse\_status\_success: the operation completed successfully.
• rocsparse\_status\_invalid\_handle: the library context was not initialized.
• rocsparse\_status\_invalid\_size: \( m \) or \( nnz \) is invalid.
• rocsparse\_status\_invalid\_pointer: \( descr, csr\_val, csr\_row\_ptr \) or \( csr\_col\_ind \) pointer is invalid.
• rocsparse\_status\_arch\_mismatch: the device is not supported.
• rocsparse\_status\_internal\_error: an internal error occurred.
• rocsparse\_status\_not\_implemented: \( trans \neq \) \texttt{rocsparse\_operation\_none} or \texttt{rocsparse\_matrix\_type \_general}.

\begin{verbatim}
rocsparse\_status rocsparse\_ccsrilu0 (rocsparse\_handle handle, rocsparse\_int m, rocsparse\_int nnz,
 const rocsparse\_mat\_descr descr, rocsparse\_float\_complex *csr\_val, const rocsparse\_int *csr\_row\_ptr,
 const rocsparse\_int *csr\_col\_ind, rocsparse\_mat\_info info, rocsparse\_solve\_policy policy,
 void *temp\_buffer)
\end{verbatim}

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

\texttt{rocsparse\_ccsrilu0} computes the incomplete LU factorization with 0 fill-ins and no pivoting of a sparse \( m \times m \) CSR matrix \( A \), such that

\[ A \approx LU \]

\texttt{rocsparse\_ccsrilu0} requires a user allocated temporary buffer. Its size is returned by \texttt{rocsparse\_ccsrilu0\_buffer\_size()}.

Furthermore, analysis meta data is required. It can be obtained by \texttt{rocsparse\_ccsrilu0\_analysis()}, \texttt{rocsparse\_ccsrilu0\_analysis()} or \texttt{rocsparse\_ccsrilu0\_analysis()}.

\texttt{rocsparse\_ccsrilu0} reports the first zero pivot (either numerical or structural zero). The zero pivot status can be obtained by calling \texttt{rocsparse\_ccsrilu0\_zero\_pivot()}.

\textbf{Note} The sparse CSR matrix has to be sorted. This can be achieved by calling \texttt{rocsparse\_csrsort()}.

\textbf{Note} This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

\textbf{Example} Consider the sparse \( m \times m \) matrix \( A \), stored in CSR storage format. The following example computes the incomplete LU factorization \( M \approx LU \) and solves the preconditioned system \( My = x \).

\begin{verbatim}
// Create rocSPARSE handle
rocsparse\_handle handle;
rocsparse\_create\_handle(&handle);

// Create matrix descriptor for M
rocsparse\_mat\_descr descr\_M;
rocsparse\_create\_mat\_descr(descr\_M);

// Create matrix descriptor for L
rocsparse\_mat\_descr descr\_L;
\end{verbatim}

(continues on next page)
rocsparse_create_mat_descr(&descr_L);
rocsparse_set_mat_fill_mode(descr_L, rocsparse_fill_mode_lower);
rocsparse_set_mat_diag_type(descr_L, rocsparse_diag_type_unit);

// Create matrix descriptor for U
rocsparse_mat_descr descr_U;
rocsparse_create_mat_descr(&descr_U);
rocsparse_set_mat_fill_mode(descr_U, rocsparse_fill_mode_upper);
rocsparse_set_mat_diag_type(descr_U, rocsparse_diag_type_non_unit);

// Create matrix info structure
rocsparse_mat_info info;
rocsparse_create_mat_info(&info);

// Obtain required buffer size
size_t buffer_size_M;
size_t buffer_size_L;
size_t buffer_size_U;
rocsparse_dcsrilu0_buffer_size(handle,
    m,
    nnz,
    descr_M,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size_M);
rocsparse_dcsrsv_buffer_size(handle,
    rocsparse_operation_none,
    m,
    nnz,
    descr_L,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size_L);
rocsparse_dcsrsv_buffer_size(handle,
    rocsparse_operation_none,
    m,
    nnz,
    descr_U,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    &buffer_size_U);

size_t buffer_size = max(buffer_size_M, max(buffer_size_L, buffer_size_U));

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);

// Perform analysis steps, using rocsparse_analysis_policy_reuse to improve
// computation performance
rocsparse_dcsrilu0_analysis(handle,
rocsparse_dcsrsv_analysis(handle, rocsparse_operation_none, m, nnz, descr_M, csr_val, csr_row_ptr, csr_col_ind, info, rocsparse_analysis_policy_reuse, rocsparse_solve_policy_auto, temp_buffer);

rocsparse_dcsrsv_analysis(handle, rocsparse_operation_none, m, nnz, descr_L, csr_val, csr_row_ptr, csr_col_ind, info, rocsparse_analysis_policy_reuse, rocsparse_solve_policy_auto, temp_buffer);

rocsparse_dcsrsv_analysis(handle, rocsparse_operation_none, m, nnz, descr_U, csr_val, csr_row_ptr, csr_col_ind, info, rocsparse_analysis_policy_reuse, rocsparse_solve_policy_auto, temp_buffer);

// Check for zero pivot
rocsparse_int position;
if (rocsparse_status_zero_pivot == rocsparse_csrilu0_zero_pivot(handle, info, &position)) {
  printf("A has structural zero at A(\%d,\%d)\n", position, position);
}

// Compute incomplete LU factorization
rocsparse_dcsrilu0(handle, m, nnz, descr_M, csr_val, csr_row_ptr, csr_col_ind, info, rocsparse_solve_policy_auto, temp_buffer);

// Check for zero pivot
if(rocsparse_status_zero_pivot == rocsparse_csrilu0_zero_pivot(handle, info, &position))
{
    printf("U has structural and/or numerical zero at U(%d,%d)\n", position, position);
}

// Solve Lz = x
rocsparse_dcsrsolve(handle, rocsparse_operation_none, m, nnz, &alpha, descr_L, csr_val, csr_row_ptr, csr_col_ind, info, x, z, rocsparse_solve_policy_auto, temp_buffer);

// Solve Uy = z
rocsparse_dcsrsolve(handle, rocsparse_operation_none, m, nnz, &alpha, descr_U, csr_val, csr_row_ptr, csr_col_ind, info, z, y, rocsparse_solve_policy_auto, temp_buffer);

// Clean up
hipFree(temp_buffer);
rocsparse_destroy_mat_info(info);
rocsparse_destroy_mat_descr(descr_M);
rocsparse_destroy_mat_descr(descr_L);
rocsparse_destroy_mat_descr(descr_U);
rocsparse_destroy_handle(handle);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
• [inout] csr_val: array of nnz elements of the sparse CSR matrix.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
• [in] info: structure that holds the information collected during the analysis step.
• [in] temp_buffer: temporary storage buffer allocated by the user.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m or nnz is invalid.
• rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr or csr_col_ind pointer is invalid.
• rocsparse_status_arch_mismatch: the device is not supported.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_zcsrILU0 (rocsparse_handle handle, rocsparse_int m, rocsparse_int nnz, 
const rocsparse_mat_desc descr, rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const 
rocsparse_int *csr_col_ind, rocsparse_mat_info info, rocsparse_solve_policy policy, void *temp_buffer)

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrILU0 computes the incomplete LU factorization with 0 fill-ins and no pivoting of a sparse m × m CSR matrix A, such that

\[ A \approx LU \]

rocsparse_csrILU0 requires a user allocated temporary buffer. Its size is returned by rocsparse_csrILU0_buffer_size(), rocsparse_dcsrILU0_buffer_size(), rocsparse_ccsrILU0_buffer_size() or rocsparse_zcsrILU0_buffer_size(). Furthermore, analysis meta data is required. It can be obtained by rocsparse_csrILU0_analysis(), rocsparse_dcsrILU0_analysis(), rocsparse_ccsrILU0_analysis() or rocsparse_zcsrILU0_analysis(). rocsparse_csrILU0 reports the first zero pivot (either numerical or structural zero). The zero pivot status can be obtained by calling rocsparse_csrILU0_zero_pivot().

Note The sparse CSR matrix has to be sorted. This can be achieved by calling rocsparse_csrSort().

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example Consider the sparse m × m matrix A, stored in CSR storage format. The following example computes the incomplete LU factorization M \approx LU and solves the preconditioned system My = x.
// Create rocSPARSE handle
crorsparse_handle handle;
crorsparse_create_handle(&handle);

// Create matrix descriptor for M
crorsparse_mat_descr descr_M;
crorsparse_create_mat_descr(&descr_M);

// Create matrix descriptor for L
crorsparse_mat_descr descr_L;
crorsparse_create_mat_descr(&descr_L);
crorsparse_set_mat_fill_mode(descr_L, rocsparse_fill_mode_lower);
crorsparse_set_mat_diag_type(descr_L, rocsparse_diag_type_unit);

// Create matrix descriptor for U
crorsparse_mat_descr descr_U;
crorsparse_create_mat_descr(&descr_U);
crorsparse_set_mat_fill_mode(descr_U, rocsparse_fill_mode_upper);
crorsparse_set_mat_diag_type(descr_U, rocsparse_diag_type_non_unit);

// Create matrix info structure
crorsparse_mat_info info;
crorsparse_create_mat_info(&info);

// Obtain required buffer size
size_t buffer_size_M;
size_t buffer_size_L;
size_t buffer_size_U;
crorsparse_dcsrilu0_buffer_size(handle,
    m,
nnz,
descr_M,
csr_val,
csr_row_ptr,
csr_col_ind,
info,
&buffer_size_M);
crorsparse_dcsrsv_buffer_size(handle,
    rocsparse_operation_none,
    m,
nnz,
descr_L,
csr_val,
csr_row_ptr,
csr_col_ind,
info,
&buffer_size_L);
crorsparse_dcsrsv_buffer_size(handle,
    rocsparse_operation_none,
    m,
nnz,
descr_U,
csr_val,
csr_row_ptr,
csr_col_ind,
info,
&buffer_size_U);
size_t buffer_size = max(buffer_size_M, max(buffer_size_L, buffer_size_U));

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);

// Perform analysis steps, using rocsparse_analysis_policy_reuse to improve
// computation performance
rocsparse_dcsrilu0_analysis(handle,
  m,
  nnz,
  descr_M,
  csr_val,
  csr_row_ptr,
  csr_col_ind,
  info,
  rocsparse_analysis_policy_reuse,
  rocsparse_solve_policy_auto,
  temp_buffer);

rocsparse_dcsrsv_analysis(handle,
  rocsparse_operation_none,
  m,
  nnz,
  descr_L,
  csr_val,
  csr_row_ptr,
  csr_col_ind,
  info,
  rocsparse_analysis_policy_reuse,
  rocsparse_solve_policy_auto,
  temp_buffer);

rocsparse_dcsrsv_analysis(handle,
  rocsparse_operation_none,
  m,
  nnz,
  descr_U,
  csr_val,
  csr_row_ptr,
  csr_col_ind,
  info,
  rocsparse_analysis_policy_reuse,
  rocsparse_solve_policy_auto,
  temp_buffer);

// Check for zero pivot
rocsparse_int position;
if(rocsparse_status_zero_pivot == rocsparse_csrilu0_zero_pivot(handle,
  info,
  &position))
{
  printf("A has structural zero at A(%d,%d)\n", position, position);
}

// Compute incomplete LU factorization
rocsparse_dcsrilu0(handle,
  m,
  (continues on next page)
nnz,
dscr_M,
csr_val,
csr_row_ptr,
csr_col_ind,
info,
rocsparse_solve_policy_auto,
temp_buffer);

// Check for zero pivot
if (rocsparse_status_zero_pivot == rocsparse_csrilu0_zero_pivot(handle,
    info,
    &position))
{
    printf("U has structural and/or numerical zero at U(\%d,\%d)\n",
        position,
        position);
}

// Solve Lz = x
rocsparse_dcsrsolve(handle,
    rocsparse_operation_none,
    m,
    nnz,
    &alpha,
    dscr_L,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    x,
    z,
    rocsparse_solve_policy_auto,
    temp_buffer);

// Solve Uy = z
rocsparse_dcsrsolve(handle,
    rocsparse_operation_none,
    m,
    nnz,
    &alpha,
    dscr_U,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    info,
    z,
    y,
    rocsparse_solve_policy_auto,
    temp_buffer);

// Clean up
hipFree(temp_buffer);
rocsparse_destroy_mat_info(info);
rocsparse_destroy_mat_descr(dscr_M);
rocsparse_destroy_mat_descr(dscr_L);
rocsparse_destroy_mat_descr(dscr_U);
rocsparse_destroy_handle(handle);

Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** m: number of rows of the sparse CSR matrix.
- **[in]** nnz: number of non-zero entries of the sparse CSR matrix.
- **[in]** descr: descriptor of the sparse CSR matrix.
- **[inout]** csr_val: array of nnz elements of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[in]** csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- **[in]** info: structure that holds the information collected during the analysis step.
- **[in]** policy: rocsparse_solve_policy_auto.
- **[in]** temp_buffer: temporary storage buffer allocated by the user.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: descr, csr_val, csr_row_ptr or csr_col_ind pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_internal_error: an internal error occurred.
- rocsparse_status_not_implemented: trans != rocsparse_operation_none or rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.12.5 rocsparse_csrilu0_clear()

rocsparse_status rocsparse_csrilu0_clear (rocsparse_handle handle, rocsparse_mat_info info)

Incomplete LU factorization with 0 fill-ins and no pivoting using CSR storage format.

rocsparse_csrilu0_clear deallocates all memory that was allocated by rocsparse_scsrilu0_analysis(), rocsparse_dcsrilu0_analysis(), rocsparse_ccsrilu0_analysis() or rocsparse_zcsrilu0_analysis(). This is especially useful, if memory is an issue and the analysis data is not required for further computation.

Note Calling rocsparse_csrilu0_clear is optional. All allocated resources will be cleared, when the opaque rocsparse_mat_info struct is destroyed using rocsparse_destroy_mat_info().

Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[inout]** info: structure that holds the information collected during the analysis step.
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_pointer: info pointer is invalid.
• rocsparse_status_memory_error: the buffer holding the meta data could not be deallocated.
• rocsparse_status_internal_error: an internal error occurred.

### 2.12.5.13 Sparse Conversion Functions

This module holds all sparse conversion routines.

The sparse conversion routines describe operations on a matrix in sparse format to obtain a matrix in a different sparse format.

#### 2.12.5.13.1 rocsparse_csr2coo()

```c
rocsparse_status rocsparse_csr2coo (rocsparse_handle handle, const rocsparse_int *csr_row_ptr, rocsparse_int nnz, rocsparse_int m, rocsparse_int *coo_row_ind, rocsparse_index_base idx_base)
```

Convert a sparse CSR matrix into a sparse COO matrix.

rocsparse_csr2coo converts the CSR array containing the row offsets, that point to the start of every row, into a COO array of row indices.

**Note** It can also be used to convert a CSC array containing the column offsets into a COO array of column indices.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example converts a CSR matrix into a COO matrix.

```c
  // Allocate CSR matrix arrays
  rocsparse_int m = 3;
  rocsparse_int n = 5;
  rocsparse_int nnz = 8;
  csr_row_ptr[m+1] = {0, 3, 5, 8}; // device memory
  csr_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
  csr_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

  // Allocate COO matrix arrays
  rocsparse_int* coo_row_ind;
  rocsparse_int* coo_col_ind;
  float* coo_val;
  hipMalloc((void**)&coo_row_ind, sizeof(rocsparse_int) * nnz);
  hipMalloc((void**)&coo_col_ind, sizeof(rocsparse_int) * nnz);
  hipMalloc((void**)&coo_val, sizeof(float) * nnz);

  // Convert the csr row offsets into coo row indices
```

(continues on next page)
rocsparse_csr2coo(handle,  
    csr_row_ptr,  
    nnz,  
    m,  
    coo_row_ind,  
    rocsparse_index_base_zero);

// Copy the column and value arrays  
hipMemcpy(coo_col_ind,  
    csr_col_ind,  
    sizeof(rocsparse_int) * nnz,  
    hipMemcpyDeviceToDevice);

hipMemcpy(coo_val,  
    csr_val,  
    sizeof(float) * nnz,  
    hipMemcpyDeviceToDevice);

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] nnz: number of non-zero entries of the sparse CSR matrix.
• [in] m: number of rows of the sparse CSR matrix.
• [out] coo_row_ind: array of nnz elements containing the row indices of the sparse COO matrix.
• [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m or nnz is invalid.
• rocsparse_status_invalid_pointer: csr_row_ptr or coo_row_ind pointer is invalid.
• rocsparse_status_arch_mismatch: the device is not supported.

2.12.5.13.2 rocsparse_coo2csr()

rocsparse_status rocsparse_coo2csr(rocsparse_handle handle, const rocsparse_int *coo_row_ind,  
    rocsparse_int nnz, rocsparse_int m, rocsparse_int *csr_row_ptr,  
    rocsparse_index_base idx_base)

Convert a sparse COO matrix into a sparse CSR matrix.

rocsparse_coo2csr converts the COO array containing the row indices into a CSR array of row offsets,  
that point to the start of every row. It is assumed that the COO row index array is sorted.

Note It can also be used, to convert a COO array containing the column indices into a CSC array of column  
offsets, that point to the start of every column. Then, it is assumed that the COO column index array is  
sorted, instead.
Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example converts a COO matrix into a CSR matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

rocspars_int m = 3;
rocspars_int n = 5;
rocspars_int nnz = 8;

coo_row_ind[nnz] = {0, 0, 0, 1, 1, 2, 2, 2}; // device memory
coo_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
coo_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Allocate CSR matrix arrays
rocspars_int* csr_row_ptr;
rocspars_int* csr_col_ind;
float* csr_val;

hipMalloc((void**)&csr_row_ptr, sizeof(rocspars_int) * (m + 1));
hipMalloc((void**)&csr_col_ind, sizeof(rocspars_int) * nnz);
hipMalloc((void**)&csr_val, sizeof(float) * nnz);

// Convert the coo row indices into csr row offsets
rocspars_coo2csr(handle,
    coo_row_ind,
    nnz,
    m,
    csr_row_ptr,
    rocsparse_index_base_zero);

// Copy the column and value arrays
hipMemcpby(csr_col_ind,
    coo_col_ind,
    sizeof(rocspars_int) * nnz,
    hipMemcpbyDeviceToDevice);

hipMemcpby(csr_val,
    coo_val,
    sizeof(float) * nnz,
    hipMemcpbyDeviceToDevice);
```

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] coo_row_ind: array of nnz elements containing the row indices of the sparse COO matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] m: number of rows of the sparse CSR matrix.
- [out] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.

Return Value
2.12.5.13.3 rocsparse_csr2csc_buffer_size()

rocsparse_status rocsparse_csr2csc_buffer_size (rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_action copy_values, size_t *buffer_size)

Convert a sparse CSR matrix into a sparse CSC matrix.

rocsparse_csr2csc_buffer_size returns the size of the temporary storage buffer required by rocsparse_scsr2csc(), rocsparse_dcsr2csc(), rocsparse_ccsr2csc() and rocsparse_zcsr2csc(). The temporary storage buffer must be allocated by the user.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [in] copy_values: rocsparse_action_symbolic or rocsparse_action_numeric.
- [out] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_scsr2csc(), rocsparse_dcsr2csc(), rocsparse_ccsr2csc() and rocsparse_zcsr2csc().

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or nnz is invalid.
- rocsparse_status_invalid_pointer: coo_row_ind or csr_row_ptr pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.
2.12.5.13.4 rocsparse_csr2csc()

rocsparse_status rocsparse_csr2csc(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, float *csc_val, rocsparse_int *csc_row_ind, rocsparse_int *csc_col_ptr, rocsparse_action copy_values, rocsparse_index_base idx_base, void *temp_buffer)

rocsparse_status rocsparse_dcsr2csc(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const double *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, double *csc_val, rocsparse_int *csc_row_ind, rocsparse_int *csc_col_ptr, rocsparse_action copy_values, rocsparse_index_base idx_base, void *temp_buffer)

Convert a sparse CSR matrix into a sparse CSC matrix.

rocsparse_csr2csc converts a CSR matrix into a CSC matrix. rocsparse_csr2csc can also be used to convert a CSC matrix into a CSR matrix. copy_values decides whether csc_val is being filled during conversion (rocsparse_action_numeric) or not (rocsparse_action_symbolic).

rocsparse_csr2csc requires extra temporary storage buffer that has to be allocated by the user. Storage buffer size can be determined by rocsparse_csr2csc_buffer_size().

Note The resulting matrix can also be seen as the transpose of the input matrix.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example computes the transpose of a CSR matrix.

```
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

rocsparse_int m_A = 3;
rocsparse_int n_A = 5;
rocsparse_int nnz_A = 8;

csr_row_ptr_A[m+1] = {0, 3, 5, 8}; // device memory
csr_col_ind_A[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val_A[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Allocate memory for transposed CSR matrix
rocsparse_int m_T = n_A;
rocsparse_int n_T = m_A;
rocsparse_int nnz_T = nnz_A;

rocsparse_int * csr_row_ptr_T;
rocsparse_int * csr_col_ind_T;
float * csr_val_T;
hipMalloc((void**)&csr_row_ptr_T, sizeof(rocsparse_int) * (m_T + 1));
hipMalloc((void**)&csr_col_ind_T, sizeof(rocsparse_int) * nnz_T);
hipMalloc((void**)&csr_val_T, sizeof(float) * nnz_T);

// Obtain the temporary buffer size
size_t buffer_size;
```
rocsparse_csr2csc_buffer_size(handle,
   m_A,
   n_A,
   nnz_A,
   csr_row_ptr_A,
   csr_col_ind_A,
   rocsparse_action_numeric,
   &buffer_size);

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);

rocsparse_scsr2csc(handle,
   m_A,
   n_A,
   nnz_A,
   csr_val_A,
   csr_row_ptr_A,
   csr_col_ind_A,
   csr_val_T,
   csr_col_ind_T,
   csr_row_ptr_T,
   rocsparse_action_numeric,
   rocsparse_index_base_zero,
   temp_buffer);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] csc_val: array of nnz elements of the sparse CSC matrix.
- [out] csc_row_ind: array of nnz elements containing the row indices of the sparse CSC matrix.
- [out] csc_col_ptr: array of n+1 elements that point to the start of every column of the sparse CSC matrix.
- [in] copy_values: rocsparse_action_symbolic or rocsparse_action_numeric.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.
- [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by rocsparse_csr2csc_buffer_size().

Return Value

- rocsparse_status_success: the operation completed successfully.
**rocsparse_status** rocsparse_csr2csc

```
rocsparse_status rocsparse_csr2csc (rocsparse_handle handle, rocsparse_int m, rocsparse_int n, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_float_complex *csc_val, rocsparse_int *csc_row_ind, rocsparse_int *csc_col_ptr, rocsparse_action copy_values, rocsparse_index_base idx_base, void *temp_buffer)
```

Convert a sparse CSR matrix into a sparse CSC matrix.

**rocsparse_csr2csc** converts a CSR matrix into a CSC matrix. **rocsparse_csr2csc** can also be used to convert a CSC matrix into a CSR matrix. **copy_values** decides whether **csc_val** is being filled during conversion (**rocsparse_action_numeric**) or not (**rocsparse_action_symbolic**).

**rocsparse_csr2csc** requires extra temporary storage buffer that has to be allocated by the user. Storage buffer size can be determined by **rocsparse_csr2csc_buffer_size**().

**Note** The resulting matrix can also be seen as the transpose of the input matrix.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example computes the transpose of a CSR matrix.

```
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

rocsparse_int m_A = 3;
rocsparse_int n_A = 5;
rocsparse_int nnz_A = 8;

csr_row_ptr_A[m+1] = {0, 3, 5, 8}; // device memory
csr_col_ind_A[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val_A[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Allocate memory for transposed CSR matrix
rocsparse_int m_T = n_A;
rocsparse_int n_T = m_A;
rocsparse_int nnz_T = nnz_A;

rocsparse_int* csr_row_ptr_T;
rocsparse_int* csr_col_ind_T;
float* csr_val_T;

hipMalloc((void**)&csr_row_ptr_T, sizeof(rocsparse_int) * (m_T + 1));
hipMalloc((void**)&csr_col_ind_T, sizeof(rocsparse_int) * nnz_T);
hipMalloc((void**)&csr_val_T, sizeof(float) * nnz_T);

// Obtain the temporary buffer size
```

(continues on next page)
size_t buffer_size;
rocspars csr2csc_buffer_size(handle,
    m_A,
    n_A,
    nnz_A,
    csr_row_ptr_A,
    csr_col_ind_A,
    rocsparse_action_numeric,
    &buffer_size);

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(temp_buffer, buffer_size);

rocspars csr2csc(handle,
    m_A,
    n_A,
    nnz_A,
    csr_val_A,
    csr_row_ptr_A,
    csr_col_ind_A,
    csr_val_T,
    csr_col_ind_T,
    csr_row_ptr_T,
    rocsparse_action_numeric,
    rocsparse_index_base_zero,
    temp_buffer);

Parameters

- [in] handle: handle to the rocspars library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] csc_val: array of nnz elements of the sparse CSC matrix.
- [out] csc_row_ind: array of nnz elements containing the row indices of the sparse CSC matrix.
- [out] csc_col_ptr: array of n+1 elements that point to the start of every column of the sparse CSC matrix.
- [in] copy_values: rocsparse_action_symbolic or rocsparse_action_numeric.
- [in] idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.
- [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by rocspars csr2csc_buffer_size().

Return Value
- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: \( m, n \) or \( nnz \) is invalid.
- rocsparse_status_invalid_pointer: \( csr\_val, csr\_row\_ptr, csr\_col\_ind, csc\_val, csc\_row\_ind, csc\_col\_ptr \) or temp buffer pointer is invalid.
- rocsparse_status_arch_mismatch: the device is not supported.
- rocsparse_status_internal_error: an internal error occurred.

\[
\text{rocsparse\_status rocsparse\_zcsr2csc (rocsparse\_handle handle, rocsparse\_int } m, \text{ rocsparse\_int } n, \text{ rocsparse\_double\_complex } *\text{csr\_val, const rocsparse\_int } *\text{csr\_row\_ptr, const rocsparse\_int } *\text{csr\_col\_ind, rocsparse\_double\_complex } *\text{csc\_val, rocsparse\_int } *\text{csc\_row\_ind, rocsparse\_int } *\text{csc\_col\_ptr, rocsparse\_action copy\_values, rocsparse\_index\_base idx\_base, void } *\text{temp\_buffer})}
\]

Convert a sparse CSR matrix into a sparse CSC matrix.

rocsparse_csr2csc converts a CSR matrix into a CSC matrix. rocsparse_csr2csc can also be used to convert a CSC matrix into a CSR matrix. \text{copy\_values} decides whether \text{csc\_val} is being filled during conversion (rocsparse\_action_numeric) or not (rocsparse\_action_symbolic).

rocsparse_csr2csc requires extra temporary storage buffer that has to be allocated by the user. Storage buffer size can be determined by rocsparse\_csr2csc\_buffer\_size().

Note The resulting matrix can also be seen as the transpose of the input matrix.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example computes the transpose of a CSR matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

rocsparse\_int m\_A = 3;
rocsparse\_int n\_A = 5;
rocsparse\_int nnz\_A = 8;

csr\_row\_ptr\_A[m+1] = {0, 3, 5, 8}; // device memory
csr\_col\_ind\_A[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr\_val\_A[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Allocate memory for transposed CSR matrix
rocsparse\_int m\_T = n\_A;
rocsparse\_int n\_T = m\_A;
rocsparse\_int nnz\_T = nnz\_A;

rocsparse\_int* csr\_row\_ptr\_T;
rocsparse\_int* csr\_col\_ind\_T;
float* csr\_val\_T;

hip\_Malloc((void**)csr\_row\_ptr\_T, size\_of(rocsparse\_int) \* (m\_T + 1));
hip\_Malloc((void**)csr\_col\_ind\_T, size\_of(rocsparse\_int) \* nnz\_T);
hip\_Malloc((void**)csr\_val\_T, size\_of(float) \* nnz\_T);
```

(continues on next page)
// Obtain the temporary buffer size
size_t buffer_size;
rocsparse csr2csc_buffer_size(handle,
m_A,
n_A,
nnz_A,
csr_row_ptr_A,
csr_col_ind_A,
rocsparse_action_numeric,
&buffer_size);

// Allocate temporary buffer
void* temp_buffer;
hipMalloc(&temp_buffer, buffer_size);
rocsparse_scsr2csc(handle,
m_A,
n_A,
nnz_A,
csr_val_A,
csr_row_ptr_A,
csr_col_ind_A,
csr_val_T,
csr_col_ind_T,
csr_row_ptr_T,
rocsparse_action_numeric,
rocsparse_index_base_zero,
temp_buffer);

Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** m: number of rows of the sparse CSR matrix.
- **[in]** n: number of columns of the sparse CSR matrix.
- **[in]** nnz: number of non-zero entries of the sparse CSR matrix.
- **[in]** csr_val: array of nnz elements of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[in]** csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- **[out]** csc_val: array of nnz elements of the sparse CSC matrix.
- **[out]** csc_row_ind: array of nnz elements containing the row indices of the sparse CSC matrix.
- **[out]** csc_col_ptr: array of n+1 elements that point to the start of every column of the sparse CSC matrix.
- **[in]** copy_values: rocsparse_action_symbolic or rocsparse_action_numeric.
- **[in]** idx_base: rocsparse_index_base_zero or rocsparse_index_base_one.
- **[in]** temp_buffer: temporary storage buffer allocated by the user, size is returned by rocsparse csr2csc_buffer_size().
Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: \(m\), \(n\) or \(nnz\) is invalid.
- `rocsparse_status_invalid_pointer`: `csr_val`, `csr_row_ptr`, `csr_col_ind`, `csc_val`, `csc_row_ind`, `csc_col_ptr` or `temp_buffer` pointer is invalid.
- `rocsparse_status_arch_mismatch`: the device is not supported.
- `rocsparse_status_arch_mismatch`: the device is not supported.
- `rocsparse_status_not_implemented`: \(roc\)-sparse_matrix_type\_general\ != \(roc\)-sparse_matrix_type\_general.

2.12.5.13.5 rocsparse_csr2ell_width()

```c
rocsparse_status rocsparse_csr2ell_width(rocsparse_handle handle, rocsparse_int m, const rocsparse_mat_descr csr_descr, const rocsparse_int *csr_row_ptr, const rocsparse_mat_descr ell_descr, rocsparse_int *ell_width)
```

Convert a sparse CSR matrix into a sparse ELL matrix.

`rocsparse_csr2ell_width` computes the maximum of the per row non-zero elements over all rows, the ELL width, for a given CSR matrix.

**Note**  This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- `[in] handle`: handle to the rocsparse library context queue.
- `[in] m`: number of rows of the sparse CSR matrix.
- `[in] csr_descr`: descriptor of the sparse CSR matrix. Currently, only `rocsparse_matrix_type_general` is supported.
- `[in] csr_row_ptr`: array of \(m+1\) elements that point to the start of every row of the sparse CSR matrix.
- `[in] ell_descr`: descriptor of the sparse ELL matrix. Currently, only `rocsparse_matrix_type_general` is supported.
- `[out] ell_width`: pointer to the number of non-zero elements per row in ELL storage format.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: \(m\) is invalid.
- `rocsparse_status_invalid_pointer`: `csr_descr`, `csr_row_ptr`, or `ell_width` pointer is invalid.
- `rocsparse_status_not_implemented`: \(roc\)-sparse_matrix_type\_general\ != \(roc\)-sparse_matrix_type\_general.
2.12.5.13.6 rocsparse_csr2ell()

rocsparse_status rocsparse_csr2ell(rocsparse_handle handle, rocsparse_int m, const rocsparse_mat_descr csr_descr, const float* csr_val, const rocsparse_int* csr_row_ptr, const rocsparse_int* csr_col_ind, const rocsparse_mat_descr ell_descr, rocsparse_int ell_width, float* ell_val, rocsparse_int* ell_col_ind)

rocsparse_status rocsparse_dcsr2ell(rocsparse_handle handle, rocsparse_int m, const rocsparse_mat_descr csr_descr, const double* csr_val, const rocsparse_int* csr_row_ptr, const rocsparse_int* csr_col_ind, const rocsparse_mat_descr ell_descr, rocsparse_int ell_width, double* ell_val, rocsparse_int* ell_col_ind)

Convert a sparse CSR matrix into a sparse ELL matrix.

rocsparse_csr2ell converts a CSR matrix into an ELL matrix. It is assumed, that ell_val and ell_col_ind are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that nnz_{ELL} = m \cdot ell_width. The number of ELL non-zero elements per row is obtained by rocsparse_csr2ell_width().

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example converts a CSR matrix into an ELL matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

t = 3;
t = 5;

CSR_row_ptr[m+1] = {0, 3, 5, 8}; // device memory
CSR_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
CSR_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Create ELL matrix descriptor
rocsparse_mat_descr ell_descr;
rocsparse_create_mat_descr(&ell_descr);

// Obtain the ELL width
rocsparse_int ell_width;
rocsparse_csr2ell_width(handle, 
    m, 
    csr_descr, 
    csr_row_ptr, 
    ell_descr, 
    &ell_width);

// Compute ELL non-zero entries
rocsparse_int ell_nnz = m \cdot ell_width;

// Allocate ELL column and value arrays
rocsparse_int* ell_col_ind;
hipMalloc((void**)&ell_col_ind, sizeof(rocsparse_int) \* ell_nnz);

float* ell_val;
```

(continues on next page)
hipMalloc((void**)&ell_val, sizeof(float) * ell_nnz);

// Format conversion
rocsparse_scsr2ell(handle, 
    m, 
    csr_descr, 
    csr_val, 
    csr_row_ptr, 
    csr_col_ind, 
    ell_descr, 
    ell_width, 
    ell_val, 
    ell_col_ind);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] csr_descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] csr_val: array containing the values of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array containing the column indices of the sparse CSR matrix.
- [in] ell_descr: descriptor of the sparse ELL matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] ell_width: number of non-zero elements per row in ELL storage format.
- [out] ell_val: array of m times ell_width elements of the sparse ELL matrix.
- [out] ell_col_ind: array of m times ell_width elements containing the column indices of the sparse ELL matrix.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m or ell_width is invalid.
- rocsparse_status_invalid_pointer: csr_descr, csr_val, csr_row_ptr, csr_col_ind, ell_descr, ell_val or ell_col_ind pointer is invalid.
- rocsparse_status_not_implemented: rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_ccsr2ell (rocsparse_handle handle, rocsparse_int m, const rocsparse_mat_descr csr_descr, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, const rocsparse_mat_descr ell_descr, rocsparse_int ell_width, rocsparse_float_complex *ell_val, rocsparse_int *ell_col_ind)

Convert a sparse CSR matrix into a sparse ELL matrix.
rocsparse_csr2ell converts a CSR matrix into an ELL matrix. It is assumed, that ell_val and ell_col_ind are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that \( \text{nnz_{ELL}} = m \cdot \text{ell_width} \). The number of ELL non-zero elements per row is obtained by \text{rocsparse_csr2ell_width()}. 

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example converts a CSR matrix into an ELL matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

rocsparse_int m = 3;
rocsparse_int n = 5;
rocsparse_int nnz = 8;

csr_row_ptr[m+1] = {0, 3, 5, 8}; // device memory
csr_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Create ELL matrix descriptor
rocsparse_mat_descr ell_descr;
rocsparse_create_mat_descr(&ell_descr);

// Obtain the ELL width
rocsparse_int ell_width;
rocsparse_csr2ell_width(handle,
    m,
    csr_descr,
    csr_row_ptr,
    ell_descr,
    ell_width);

// Compute ELL non-zero entries
rocsparse_int ell_nnz = m * ell_width;

// Allocate ELL column and value arrays
rocsparse_int* ell_col_ind;
hipMalloc((void**)&ell_col_ind, sizeof(rocsparse_int) * ell_nnz);

float* ell_val;
hipMalloc((void**)&ell_val, sizeof(float) * ell_nnz);

// Format conversion
rocsparse_scsr2ell(handle,
    m,
    csr_descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    ell_descr,
    ell_width,
    ell_val,
    ell_col_ind);
```

**Parameters**
**rocsparses**

### rocsparse_zcsr2ell

#### Signature

```c
rocsparses (rocsparses_handle handle, rocsparse_int m, const rocsparse_mat_descr csr_descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, const rocsparse_mat_descr ell_descr, rocsparse_int ell_width, rocsparse_double_complex *ell_val, rocsparse_int *ell_col_ind)
```

#### Description

Convert a sparse CSR matrix into a sparse ELL matrix.

**rocsparses csr2ell** converts a CSR matrix into an ELL matrix. It is assumed, that ell_val and ell_col_ind are allocated. Allocation size is computed by the number of rows times the number of ELL non-zero elements per row, such that `nnzEll = m \cdot ell_width`. The number of ELL non-zero elements per row is obtained by `rocsparses csr2ell_width()`.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example converts a CSR matrix into an ELL matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8

rocsparses_int m = 3;
rocsparses_int n = 5;
rocsparses_int nnz = 8;
```

(continues on next page)
csr_row_ptr[m+1] = {0, 3, 5, 8}; // device memory
csr_col_ind[nnz] = {0, 1, 3, 1, 2, 0, 3, 4}; // device memory
csr_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8}; // device memory

// Create ELL matrix descriptor
rocsparse_mat_descr ell_descr;
rocsparse_create_mat_descr(&ell_descr);

// Obtain the ELL width
rocsparse_int ell_width;
rocsparse_csr2ell_width(handle,
m,
    csr_descr,
    csr_row_ptr,
    ell_descr,
    sell_width);

// Compute ELL non-zero entries
rocsparse_int ell_nnz = m * ell_width;

// Allocate ELL column and value arrays
rocsparse_int* ell_col_ind;
hipMalloc((void**)&ell_col_ind, sizeof(rocsparse_int) * ell_nnz);
float* ell_val;
hipMalloc((void**)&ell_val, sizeof(float) * ell_nnz);

// Format conversion
rocsparse_scsr2ell(handle,
m,
    csr_descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind,
    ell_descr,
    ell_width,
    ell_val,
    ell_col_ind);

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] m: number of rows of the sparse CSR matrix.
• [in] csr_descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
• [in] csr_val: array containing the values of the sparse CSR matrix.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] csr_col_ind: array containing the column indices of the sparse CSR matrix.
• [in] ell_descr: descriptor of the sparse ELL matrix. Currently, only rocsparse_matrix_type_general is supported.
• [in] ell_width: number of non-zero elements per row in ELL storage format.
ReadTheDocs-Breathe Documentation, Release 1.0.0

- [out] ell_val: array of \( m \) times ell_width elements of the sparse ELL matrix.
- [out] ell_col_ind: array of \( m \) times ell_width elements containing the column indices of the sparse ELL matrix.

Return Value

- rocspmeal_status_success: the operation completed successfully.
- rocspmeal_status_invalid_handle: the library context was not initialized.
- rocspmeal_status_invalid_size: \( m \) or ell_width is invalid.
- rocspmeal_status_invalid_pointer: csr_descr, csr_val, csr_row_ptr, csr_col_ind, ell_descr, ell_val or ell_col_ind pointer is invalid.
- rocspmeal_status_not_implemented: rocspmeal_matrix_type != rocspmeal_matrix_type_general.

2.12.5.13.7 rocspmeal_ell2csr_nnz()

rocspmeal_status rocspmeal_ell2csr_nnz(rocspmeal_handle handle, rocspmeal_int m, rocspmeal_int n, const rocspmeal_mat_descr ell_descr, rocspmeal_int ell_width, const rocspmeal_int *ell_col_ind, const rocspmeal_mat_descr csr_descr, rocspmeal_int *csr_row_ptr, rocspmeal_int *csr_nnz)

Convert a sparse ELL matrix into a sparse CSR matrix.

rocspmeal_ell2csr_nnz computes the total CSR non-zero elements and the CSR row offsets, that point to the start of every row of the sparse CSR matrix, for a given ELL matrix. It is assumed that csr_row_ptr has been allocated with size \( m + 1 \).

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Parameters

- [in] handle: handle to the rocspmeal library context queue.
- [in] m: number of rows of the sparse ELL matrix.
- [in] n: number of columns of the sparse ELL matrix.
- [in] ell_descr: descriptor of the sparse ELL matrix. Currently, only rocspmeal_matrix_type_general is supported.
- [in] ell_width: number of non-zero elements per row in ELL storage format.
- [in] ell_col_ind: array of \( m \) times ell_width elements containing the column indices of the sparse ELL matrix.
- [in] csr_descr: descriptor of the sparse CSR matrix. Currently, only rocspmeal_matrix_type_general is supported.
- [out] csr_row_ptr: array of \( m+1 \) elements that point to the start of every row of the sparse CSR matrix.
- [out] csr_nnz: pointer to the total number of non-zero elements in CSR storage format.

Return Value

- rocspmeal_status_success: the operation completed successfully.
- rocspmeal_status_invalid_handle: the library context was not initialized.
• **rocsparse_status_invalid_size**: `m`, `n` or `ell_width` is invalid.

• **rocsparse_status_invalid_pointer**: `ell_descr`, `ell_col_ind`, `csr_descr`, `csr_row_ptr` or `csr_nnz` pointer is invalid.

• **rocsparse_status_notImplemented**: `rocsparse_matrix_type` != `rocsparse_matrix_type_general`.

### 2.12.5.13.8 rocsparse_ell2csr()

```c
rocsparse_status rocsparse_ell2csr(rocsparse_handle handle, rocsparse_int m, rocsparse_int n,
const rocsparse_mat_descr ell_descr, rocsparse_int ell_width,
const float *ell_val, const rocsparse_int *ell_col_ind, const rocsparse_mat_descr csr_descr, float *csr_val, const rocsparse_int *csr_row_ptr, rocsparse_int *csr_col_ind)
```

Convert a sparse ELL matrix into a sparse CSR matrix.

**rocsparse_ell2csr** converts an ELL matrix into a CSR matrix. It is assumed that `csr_row_ptr` has already been filled and that `csr_val` and `csr_col_ind` are allocated by the user. `csr_row_ptr` and allocation size of `csr_col_ind` and `csr_val` is defined by the number of CSR non-zero elements. Both can be obtained by **rocsparse_ell2csr_nnz**.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example converts an ELL matrix into a CSR matrix.

```c
// A = 1 2 0 3 0
// 0 4 5 0 0
// 6 0 0 7 8

rocsparse_int m = 3;
rocsparse_int n = 5;
rocsparse_int nnz = 9;
rocsparse_int ell_width = 3;

ell_col_ind[nnz] = {0, 1, 0, 1, 2, 3, 3, -1, 4}; // device memory
ell_val[nnz] = {1, 4, 6, 2, 5, 7, 3, 0, 8}; // device memory

// Create CSR matrix descriptor
rocsparse_mat_descr csr_descr;
rocsparse_create_mat_descr(&csr_descr);

// Allocate csr_row_ptr array for row offsets
rocsparse_int *csr_row_ptr;
hipMalloc((void **)&csr_row_ptr, sizeof(rocsparse_int) * (m + 1));

// Obtain the number of CSR non-zero entries
// and fill csr_row_ptr array with row offsets
rocsparse_int csr_nnz;
rocsparse_ell2csr_nnz(handle,
```
m,
  n,
  ell_descr,
  ell_width,
  ell_col_ind,
  csr_descr,
  csr_row_ptr,
  csr_nnz);

// Allocate CSR column and value arrays
rocsparse_int* csr_col_ind;
hpMalloc((void**)&csr_col_ind, sizeof(rocsparse_int) * csr_nnz);

float* csr_val;
hpMalloc((void**)&csr_val, sizeof(float) * csr_nnz);

// Format conversion
rocsparse_sell2csr(handle,
  m,
  n,
  ell_descr,
  ell_width,
  ell_val,
  ell_col_ind,
  csr_descr,
  csr_val,
  csr_row_ptr,
  csr_col_ind);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse ELL matrix.
- [in] n: number of columns of the sparse ELL matrix.
- [in] ell_descr: descriptor of the sparse ELL matrix. Currently, only roc-
sparse_matrix_type_general is supported.
- [in] ell_width: number of non-zero elements per row in ELL storage format.
- [in] ell_val: array of m times ell_width elements of the sparse ELL matrix.
- [in] ell_col_ind: array of m times ell_width elements containing the column indices of
  the sparse ELL matrix.
- [in] csr_descr: descriptor of the sparse CSR matrix. Currently, only roc-
sparse_matrix_type_general is supported.
- [out] csr_val: array containing the values of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR
  matrix.
- [out] csr_col_ind: array containing the column indices of the sparse CSR matrix.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: $m$, $n$ or $ell\_width$ is invalid.
• rocsparse_status_invalid_pointer: $csr\_descr$, $csr\_val$, $csr\_row\_ptr$, $csr\_col\_ind$, $ell\_descr$, $ell\_val$ or $ell\_col\_ind$ pointer is invalid.
• rocsparse_status_not_implemented: $rocsparse\_matrix\_type \neq rocsparse\_matrix\_type\_general$.

rocsparse_status rocsparse_cell2csr(rocsparse_handle handle, rocsparse_int $m$, const rocsparse_mat_descr ell_descr, rocsparse_int $ell\_width$, const rocsparse_float_complex *ell_val, const rocsparse_int *ell_col_ind, const rocsparse_mat_descr csr_descr, rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, rocsparse_int *csr_col_ind)

Convert a sparse ELL matrix into a sparse CSR matrix.

rocsparse_ell2csr converts an ELL matrix into a CSR matrix. It is assumed that $csr\_row\_ptr$ has already been filled and that $csr\_val$ and $csr\_col\_ind$ are allocated by the user. $csr\_row\_ptr$ and allocation size of $csr\_col\_ind$ and $csr\_val$ is defined by the number of CSR non-zero elements. Both can be obtained by rocsparse_ell2csr_nnz().

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example converts an ELL matrix into a CSR matrix.

```
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8
rocsparse_int m = 3;
rocsparse_int n = 5;
rocsparse_int nnz = 9;
rocsparse_int ell_width = 3;
ell_col_ind[nnz] = {0, 1, 0, 1, 2, 3, -1, 4}; // device memory
ell_val[nnz] = {1, 4, 6, 2, 5, 7, 3, 0, 8}; // device memory

// Create CSR matrix descriptor
rocsparse_mat_descr csr_descr;
rocsparse_create_mat_descr(&csr_descr);

// Allocate csr_row_ptr array for row offsets
rocsparse_int* csr_row_ptr;
hipMalloc((void**)&csr_row_ptr, sizeof(rocsparse_int) * (m + 1));

// Obtain the number of CSR non-zero entries
// and fill csr_row_ptr array with row offsets
rocsparse_int csr_nnz;
rocsparse_ell2csr_nnz(handle,
    m,
    n,
    ell_descr,
    ell_width,
    ell_col_ind,
    csr descr,
    csr_row_ptr,
    &csr_nnz);
```

(continues on next page)
// Allocate CSR column and value arrays
rocsparse_int* csr_col_ind;
hipMalloc((void**)&csr_col_ind, sizeof(rocsparse_int) * csr_nnz);

float* csr_val;
hipMalloc((void**)&csr_val, sizeof(float) * csr_nnz);

// Format conversion
rocsparse_sell2csr(handle,
    m,
    n,
    ell_descr,
    ell_width,
    ell_val,
    ell_col_ind,
    csr_descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse ELL matrix.
- [in] n: number of columns of the sparse ELL matrix.
- [in] ell_descr: descriptor of the sparse ELL matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] ell_width: number of non-zero elements per row in ELL storage format.
- [in] ell_val: array of m times ell_width elements of the sparse ELL matrix.
- [in] ell_col_ind: array of m times ell_width elements containing the column indices of the sparse ELL matrix.
- [in] csr_descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
- [out] csr_val: array containing the values of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [out] csr_col_ind: array containing the column indices of the sparse CSR matrix.

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m, n or ell_width is invalid.
- rocsparse_status_invalid_pointer: csr_descr, csr_val, csr_row_ptr, csr_col_ind, ell_descr, ell_val or ell_col_ind pointer is invalid.
- rocsparse_status_not_implemented: rocsparse_matrix_type != rocsparse_matrix_type_general.
rocsparse_status rocsparse_ell2csr(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, const rocsparse_mat_descr ell_descr, rocsparse_int ell_width, const rocsparse_double_complex *ell_val, const rocsparse_int *ell_col_ind, const rocsparse_mat_descr csr_descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind)

Convert a sparse ELL matrix into a sparse CSR matrix.

rocsparse_ell2csr converts an ELL matrix into a CSR matrix. It is assumed that csr_row_ptr has already been filled and that csr_val and csr_col_ind are allocated by the user. csr_row_ptr and allocation size of csr_col_ind and csr_val is defined by the number of CSR non-zero elements. Both can be obtained by rocsparse_ell2csr_nnz().

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** This example converts an ELL matrix into a CSR matrix.

```c
// 1 2 0 3 0
// A = 0 4 5 0 0
// 6 0 0 7 8
rocsparse_int m = 3;
rocsparse_int n = 5;
rocsparse_int nnz = 9;
rocsparse_int ell_width = 3;

ell_col_ind[nnz] = {0, 1, 0, 1, 2, 3, 3, -1, 4}; // device memory
ell_val[nnz] = {1, 4, 6, 2, 5, 7, 3, 0, 8}; // device memory

// Create CSR matrix descriptor
rocsparse_mat_descr csr descr;
rocsparse_create_mat_descr(&csr descr);

// Allocate csr_row_ptr array for row offsets
rocsparse_int csr_row_ptr;
hipMalloc((void**)&csr_row_ptr, sizeof(rocsparse_int) * (m + 1));

// Obtain the number of CSR non-zero entries
// and fill csr_row_ptr array with row offsets
rocsparse_int csr nnz;
rocsparse_ell2csr nnz(handle,
    m,
    n,
    ell_descr,
    ell_width,
    ell_col_ind,
    csr descr,
    csr_row_ptr,
    &csr nnz);

// Allocate CSR column and value arrays
rocsparse_int* csr_col_ind;
hipMalloc((void**)&csr_col_ind, sizeof(rocsparse_int) * csr nnz);

float* csr_val;
hipMalloc((void**)&csr_val, sizeof(float) * csr nnz);
```

(continues on next page)
// Format conversion
rocspart_sell2csr(handle,
    m,
    n,
    ell_descr,
    ell_width,
    ell_val,
    ell_col_ind,
    csr_descr,
    csr_val,
    csr_row_ptr,
    csr_col_ind);

Parameters

- **[in]** handle: handle to the rocsparse library context queue.
- **[in]** m: number of rows of the sparse ELL matrix.
- **[in]** n: number of columns of the sparse ELL matrix.
- **[in]** ell_descr: descriptor of the sparse ELL matrix. Currently, only `rocspart_matrix_type_general` is supported.
- **[in]** ell_width: number of non-zero elements per row in ELL storage format.
- **[in]** ell_val: array of m times ell_width elements of the sparse ELL matrix.
- **[in]** ell_col_ind: array of m times ell_width elements containing the column indices of the sparse ELL matrix.
- **[in]** csr_descr: descriptor of the sparse CSR matrix. Currently, only `rocspart_matrix_type_general` is supported.
- **[out]** csr_val: array containing the values of the sparse CSR matrix.
- **[in]** csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- **[out]** csr_col_ind: array containing the column indices of the sparse CSR matrix.

Return Value

- **rocspart_status_success**: the operation completed successfully.
- **rocspart_status_invalid_handle**: the library context was not initialized.
- **rocspart_status_invalid_size**: m, n or ell_width is invalid.
- **rocspart_status_invalid_pointer**: csr_descr, csr_val, csr_row_ptr, csr_col_ind, ell_descr, ell_val or ell_col_ind pointer is invalid.
- **rocspart_status_not_implemented**: rocspart_matrix_type != rocspart_matrix_type_general.
2.12.5.13.9 rocsparse_csr2hyb()

rocsparse_status rocsparse_scsr2hyb(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, const rocsparse_mat_descr descr, const float *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_hyb_mat hyb, rocsparse_int user_ell_width, rocsparse_hyb_partition partition_type)

rocsparse_status rocsparse_ddcsr2hyb(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, const rocsparse_mat_descr descr, const double *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_hyb_mat hyb, rocsparse_int user_ell_width, rocsparse_hyb_partition partition_type)

Convert a sparse CSR matrix into a sparse HYB matrix.

rocsparse_csr2hyb converts a CSR matrix into a HYB matrix. It is assumed that hyb has been initialized with rocsparse_create_hyb_mat().

Note This function requires a significant amount of storage for the HYB matrix, depending on the matrix structure.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example converts a CSR matrix into a HYB matrix using user defined partitioning.

```c
// Create HYB matrix structure
rocsparse_hyb_mat hyb;
rocsparse_create_hyb_mat(&hyb);

// User defined ell width
rocsparse_int user_ell_width = 5;

// Perform the conversion
rocsparse_scsr2hyb(handle, m, n, descr, csr_val, csr_row_ptr, csr_col_ind, hyb, user_ell_width, rocsparse_hyb_partition_user);

// Do some work

// Clean up
rocsparse_destroy_hyb_mat(hyb);
```

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
• [in] csr_val: array containing the values of the sparse CSR matrix.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [in] csr_col_ind: array containing the column indices of the sparse CSR matrix.
• [out] hyb: sparse matrix in HYB format.
• [in] user_ell_width: width of the ELL part of the HYB matrix (only required if partition_type == rocsparse_hyb_partition_user).
• [in] partition_type: rocsparse_hyb_partition_auto (recommended), rocsparse_hyb_partition_user or rocsparse_hyb_partition_max.

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n or user_ell_width is invalid.
• rocsparse_status_invalid_value: partition_type is invalid.
• rocsparse_status_invalid_pointer: descr, hyb, csr_val, csr_row_ptr or csr_col_ind pointer is invalid.
• rocsparse_status_memory_error: the buffer for the HYB matrix could not be allocated.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: rocsparse_matrix_type != rocsparse_matrix_type_general.

rocsparse_status rocsparse_csr2hyb (rocsparse_handle handle, rocsparse_int m, rocsparse_int n, const rocsparse_mat_descr descr, const rocsparse_float_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_hyb_mat hyb, rocsparse_int user_ell_width, rocsparse_hyb_partition partition_type)

Convert a sparse CSR matrix into a sparse HYB matrix.

rocsparse_csr2hyb converts a CSR matrix into a HYB matrix. It is assumed that hyb has been initialized with rocsparse_create_hyb_mat().

Note This function requires a significant amount of storage for the HYB matrix, depending on the matrix structure.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example converts a CSR matrix into a HYB matrix using user defined partitioning.

```c
// Create HYB matrix structure
rocsparse_hyb_mat hyb;
rocsparse_create_hyb_mat(&hyb);

// User defined ell width
rocsparse_int user_ell_width = 5;

// Perform the conversion
rocsparse_scsr2hyb(handle,
```
```c
m,
\ n,
\ descr,
\ csr_val,
\ csr_row_ptr,
\ csr_col_ind,
\ hyb,
\ user_ell_width,
\ rocsparse_hyb_partition_user);

// Do some work
// Clean up
rocsparse_destroy_hyb_mat(hyb);
```

### Parameters

- **[in]** `handle`: handle to the rocsparse library context queue.
- **[in]** `m`: number of rows of the sparse CSR matrix.
- **[in]** `n`: number of columns of the sparse CSR matrix.
- **[in]** `descr`: descriptor of the sparse CSR matrix. Currently, only `rocsparse_matrix_type_general` is supported.
- **[in]** `csr_val`: array containing the values of the sparse CSR matrix.
- **[in]** `csr_row_ptr`: array of `m+1` elements that point to the start of every row of the sparse CSR matrix.
- **[in]** `csr_col_ind`: array containing the column indices of the sparse CSR matrix.
- **[out]** `hyb`: sparse matrix in HYB format.
- **[in]** `user_ell_width`: width of the ELL part of the HYB matrix (only required if `partition_type == rocsparse_hyb_partition_user`).
- **[in]** `partition_type`: `rocsparse_hyb_partition_auto` (recommended), `rocsparse_hyb_partition_user` or `rocsparse_hyb_partition_max`.

### Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: `m`, `n` or `user_ell_width` is invalid.
- `rocsparse_status_invalid_value`: `partition_type` is invalid.
- `rocsparse_status_invalid_pointer`: `descr`, `hyb`, `csr_val`, `csr_row_ptr` or `csr_col_ind` pointer is invalid.
- `rocsparse_status_memory_error`: the buffer for the HYB matrix could not be allocated.
- `rocsparse_status_internal_error`: an internal error occurred.
- `rocsparse_status_not_implemented`: `rocsparse_matrix_type` != `rocsparse_matrix_type_general`. 

---

### 2.12. ROCm Libraries

403
rocsparse_status rocsparse_zcsr2hyb(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, const rocsparse_mat_descr descr, const rocsparse_double_complex *csr_val, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, rocsparse_hyb_mat hyb, rocsparse_int user_ell_width, rocsparse_hyb_partition partition_type)

Convert a sparse CSR matrix into a sparse HYB matrix.

rocsparse_zcsr2hyb converts a CSR matrix into a HYB matrix. It is assumed that hyb has been initialized with rocsparse_create_hyb_mat().

Note This function requires a significant amount of storage for the HYB matrix, depending on the matrix structure.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example This example converts a CSR matrix into a HYB matrix using user defined partitioning.

```c
// Create HYB matrix structure
rocsparse_hyb_mat hyb;
rocsparse_create_hyb_mat(&hyb);

// User defined ell width
rocsparse_int user_ell_width = 5;

// Perform the conversion
rocsparse_zcsr2hyb(handle, m, n, descr, csr_val, csr_row_ptr, csr_col_ind, hyb, user_ell_width, rocsparse_hyb_partition_user);

// Do some work

// Clean up
rocsparse_destroy_hyb_mat(hyb);
```

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
- [in] csr_val: array containing the values of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array containing the column indices of the sparse CSR matrix.
- [out] hyb: sparse matrix in HYB format.
• \[\text{in}\] user_ell_width: width of the ELL part of the HYB matrix (only required if partition_type == rocsparse_hyb_partition_user).

• \[\text{in}\] partition_type: rocsparse_hyb_partition_auto (recommended), rocsparse_hyb_partition_user or rocsparse_hyb_partition_max.

Return Value

• rocsparse_status_success: the operation completed successfully.

• rocsparse_status_invalid_handle: the library context was not initialized.

• rocsparse_status_invalid_size: \(m, n\) or user_ell_width is invalid.

• rocsparse_status_invalid_value: partition_type is invalid.

• rocsparse_status_invalid_pointer: descr, hyb, csr_val, csr_row_ptr or csr_col_ind pointer is invalid.

• rocsparse_status_memory_error: the buffer for the HYB matrix could not be allocated.

• rocsparse_status_internal_error: an internal error occurred.

• rocsparse_status_not_implemented: rocsparse_matrix_type != rocsparse_matrix_type_general.

### 2.12.5.13.10 rocsparse_create_identity_permutation()

**rocsparse_status rocsparse_create_identity_permutation**(rocsparse_handle handle, rocsparse_int n, rocsparse_int *p)

Create the identity map.

rocsparse_create_identity_permutation stores the identity map in p, such that \(p = 0 : 1 : (n - 1)\).

```c
for(i = 0; i < n; ++i)
{
    p[i] = i;
}
```

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** The following example creates an identity permutation.

```c
rocsparse_int size = 200;

// Allocate memory to hold the identity map
rocsparse_int* perm;
hipMalloc((void**)&perm, sizeof(rocsparse_int) * size);

// Fill perm with the identity permutation
rocsparse_create_identity_permutation(handle, size, perm);
```

**Parameters**

• \[\text{in}\] handle: handle to the rocsparse library context queue.

• \[\text{in}\] n: size of the map p.

• \[\text{out}\] p: array of n integers containing the map.
Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: n is invalid.
- `rocsparse_status_invalid_pointer`: pointer is invalid.

2.12.5.13.11 `rocsparse_csrsort_buffer_size()`

```c
rocsparse_status rocsparse_csrsort_buffer_size (rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_int *csr_row_ptr, const rocsparse_int *csr_col_ind, size_t *buffer_size)
```

Sort a sparse CSR matrix.

`rocsparse_csrsort_buffer_size` returns the size of the temporary storage buffer required by `rocsparse_csrsort()`. The temporary storage buffer must be allocated by the user.

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse CSR matrix.
- [in] n: number of columns of the sparse CSR matrix.
- [in] nnz: number of non-zero entries of the sparse CSR matrix.
- [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
- [in] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
- [out] buffer_size: number of bytes of the temporary storage buffer required by `rocsparse_csrsort()`.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: the library context was not initialized.
- `rocsparse_status_invalid_size`: m, n or nnz is invalid.
- `rocsparse_status_invalid_pointer`: csr_row_ptr, csr_col_ind or buffer_size pointer is invalid.

2.12.5.13.12 `rocsparse_csrsort()`

```c
rocsparse_status rocsparse_csrsort (rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_mat_descr descr, const rocsparse_int *csr_row_ptr, rocsparse_int *csr_col_ind, rocsparse_int *perm, void *temp_buffer)
```

Sort a sparse CSR matrix.
rocsparse_csrsort sorts a matrix in CSR format. The sorted permutation vector perm can be used to obtain sorted csr_val array. In this case, perm must be initialized as the identity permutation, see rocsparse_create_identity_permutation().

rocsparse_csrsort requires extra temporary storage buffer that has to be allocated by the user. Storage buffer size can be determined by rocsparse_csrsort_buffer_size().

**Note** perm can be NULL if a sorted permutation vector is not required.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** The following example sorts a $3 \times 3$ CSR matrix.

```c
// A =
// 1 2 3
// 4 5 6
// 7 8 9
rocsparse_int m = 3;
rocsparse_int n = 3;
rocsparse_int nnz = 9;
csr_row_ptr[m + 1] = {0, 3, 6, 9}; // device memory
csr_col_ind[nnz] = {2, 0, 1, 0, 1, 2, 0, 2, 1}; // device memory
csr_val[nnz] = {3, 1, 2, 4, 5, 6, 7, 9, 8}; // device memory

// Create permutation vector perm as the identity map
rocsparse_int* perm;
hipMalloc((void**)&perm, sizeof(rocsparse_int) * nnz);
rocsparse_create_identity_permutation(handle, nnz, perm);

// Allocate temporary buffer
size_t buffer_size;
void* temp_buffer;
rocsparse_csrsort_buffer_size(handle, m, n, nnz, csr_row_ptr, csr_col_ind, &buffer_size);
hipMalloc(&temp_buffer, buffer_size);

// Sort the CSR matrix
rocsparse_csrsort(handle, m, n, nnz, descr, csr_row_ptr, csr_col_ind, perm, temp_buffer);

// Gather sorted csr_val array
float* csr_val_sorted;
hipMalloc((void**)&csr_val_sorted, sizeof(float) * nnz);
rocsparse_sgthr(handle, nnz, csr_val, csr_val_sorted, perm, rocsparse_index_base_zero);

// Clean up
hipFree(temp_buffer);
hipFree(perm);
hipFree(csr_val);
```

**Parameters**

- `[in]` handle: handle to the rocsparse library context queue.
- `[in]` m: number of rows of the sparse CSR matrix.
- `[in]` n: number of columns of the sparse CSR matrix.
- `[in]` nnz: number of non-zero entries of the sparse CSR matrix.
• [in] descr: descriptor of the sparse CSR matrix. Currently, only rocsparse_matrix_type_general is supported.
• [in] csr_row_ptr: array of m+1 elements that point to the start of every row of the sparse CSR matrix.
• [inout] csr_col_ind: array of nnz elements containing the column indices of the sparse CSR matrix.
• [inout] perm: array of nnz integers containing the unsorted map indices, can be NULL.
• [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by rocsparse csrsort_buffer_size().

Return Value
• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n or nnz is invalid.
• rocsparse_status_invalid_pointer: descr, csr_row_ptr, csr_col_ind or temp_buffer pointer is invalid.
• rocsparse_status_internal_error: an internal error occurred.
• rocsparse_status_not_implemented: rocsparse_matrix_type != rocsparse_matrix_type_general.

2.12.5.13.13 rocsparse_coosort_buffer_size()

rocsparse_status rocsparse_coosort_buffer_size (rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, const rocsparse_int *coo_row_ind, const rocsparse_int *coo_col_ind, size_t *buffer_size)

Sort a sparse COO matrix.

coirsort_buffer_size returns the size of the temporary storage buffer that is required by rocsparse_coosort_by_row() and rocsparse_coosort_by_column(). The temporary storage buffer has to be allocated by the user.

Parameters
• [in] handle: handle to the rocsparse library context queue.
• [in] m: number of rows of the sparse COO matrix.
• [in] n: number of columns of the sparse COO matrix.
• [in] nnz: number of non-zero entries of the sparse COO matrix.
• [in] coo_row_ind: array of nnz elements containing the row indices of the sparse COO matrix.
• [in] coo_col_ind: array of nnz elements containing the column indices of the sparse COO matrix.
• [out] buffer_size: number of bytes of the temporary storage buffer required by rocsparse_coosort_by_row() and rocsparse_coosort_by_column().

Return Value
• rocsparse_status_success: the operation completed successfully.
• `rocspase_status_invalid_handle`: the library context was not initialized.
• `rocspase_status_invalid_size`: `m`, `n` or `nnz` is invalid.
• `rocspase_status_invalid_pointer`: `coo_row_ind`, `coo_col_ind` or buffer_size pointer is invalid.
• `rocspase_status_internal_error`: an internal error occurred.

2.12.5.13.14 `rocspase_coosort_by_row()`

```
rocspase_status rocspase_coosort_by_row (rocspase_handle handle, rocspase_int m, rocspase_int n, rocspase_int nnz, rocspase_int *coo_row_ind, rocspase_int *coo_col_ind, rocspase_int *perm, void *temp_buffer)
```

Sort a sparse COO matrix by row.

`rocspase_coosort_by_row` sorts a matrix in COO format by row. The sorted permutation vector `perm` can be used to obtain sorted `coo_val` array. In this case, `perm` must be initialized as the identity permutation, see `rocspase_create_identity_permutation()`.

`rocspase_coosort_by_row` requires extra temporary storage buffer that has to be allocated by the user. Storage buffer size can be determined by `rocspase_coosort_buffer_size()`.

**Note** `perm` can be `NULL` if a sorted permutation vector is not required.

**Note** This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

**Example** The following example sorts a $3 \times 3$ COO matrix by row indices.

```c
// 1 2 3
// A = 4 5 6
// 7 8 9
rocspase_int m = 3;
rocspase_int n = 3;
rocspase_int nnz = 9;

coo_row_ind[nnz] = {0, 1, 2, 0, 1, 2, 0, 1, 2}; // device memory
coo_col_ind[nnz] = {0, 0, 0, 1, 1, 1, 2, 2, 2}; // device memory
coo_val[nnz] = {1, 4, 7, 2, 5, 8, 3, 6, 9}; // device memory

// Create permutation vector perm as the identity map
rocspase_int* perm;
hipMalloc((void**)&perm, sizeof(rocspase_int) * nnz);
rocspase_create_identity_permutation(handle, nnz, perm);

// Allocate temporary buffer
size_t buffer_size;
void* temp_buffer;
rocspase_coosort_buffer_size(handle, m, n, nnz, coo_row_ind, coo_col_ind, &buffer_size);
hipMalloc(&temp_buffer, buffer_size);
```

(continues on next page)
// Sort the COO matrix
rocspare_coosort_by_row(handle,
m, 
n, 
ngen, 
coo_row_ind, 
coo_col_ind, 
perm, 
temp_buffer);

// Gather sorted coo_val array
float* coo_val_sorted;
hipMalloc((void**)&coo_val_sorted, sizeof(float) * nnn);
rocspare_sgthr(handle, nnn, coo_val, coo_val_sorted, perm, rocsparse_index_ 
—base_zero);

// Clean up
hipFree(temp_buffer);
hipFree(perm);
hipFree(coo_val);

Parameters

• [in] handle: handle to the rocsparse library context queue.
• [in] m: number of rows of the sparse COO matrix.
• [in] n: number of columns of the sparse COO matrix.
• [in] nnn: number of non-zero entries of the sparse COO matrix.
• [inout] coo_row_ind: array of nnn elements containing the row indices of the sparse COO matrix.
• [inout] coo_col_ind: array of nnn elements containing the column indices of the sparse COO matrix.
• [inout] perm: array of nnn integers containing the unsorted map indices, can be NULL.
• [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by rocsparse_coosort_buffer_size().

Return Value

• rocsparse_status_success: the operation completed successfully.
• rocsparse_status_invalid_handle: the library context was not initialized.
• rocsparse_status_invalid_size: m, n or nnn is invalid.
• rocsparse_status_invalid_pointer: coo_row_ind, coo_col_ind or temp_buffer pointer is invalid.
• rocsparse_status_internal_error: an internal error occurred.
2.12.5.13.15 rocsparse_coosort_by_column()

```
rocsparse_status rocsparse_coosort_by_column(rocsparse_handle handle, rocsparse_int m, rocsparse_int n, rocsparse_int nnz, rocsparse_int *coo_row_ind, rocsparse_int *coo_col_ind, rocsparse_int *perm, void *temp_buffer)
```

Sort a sparse COO matrix by column.

rocsparse_coosort_by_column sorts a matrix in COO format by column. The sorted permutation vector `perm` can be used to obtain sorted `coo_val` array. In this case, `perm` must be initialized as the identity permutation, see `rocsparse_create_identity_permutation()`.

rocsparse_coosort_by_column requires extra temporary storage buffer that has to be allocated by the user. Storage buffer size can be determined by `rocsparse_coosort_buffer_size()`.

Note perm can be NULL if a sorted permutation vector is not required.

Note This function is non blocking and executed asynchronously with respect to the host. It may return before the actual computation has finished.

Example The following example sorts a $3 \times 3$ COO matrix by column indices.

```c
// 1 2 3
// A = 4 5 6
// 7 8 9
rocsparse_int m = 3;
rocsparse_int n = 3;
rocsparse_int nnz = 9;

coo_row_ind[nnz] = {0, 0, 0, 1, 1, 1, 2, 2, 2}; // device memory
coo_col_ind[nnz] = {0, 1, 2, 0, 1, 2, 0, 1, 2}; // device memory
coo_val[nnz] = {1, 2, 3, 4, 5, 6, 7, 8, 9}; // device memory

// Create permutation vector perm as the identity map
rocsparse_int* perm;
hipMalloc((void**)&perm, sizeof(rocsparse_int) * nnz);
rocsparse_create_identity_permutation(handle, nnz, perm);

// Allocate temporary buffer
size_t buffer_size;
void* temp_buffer;
rocsparse_coosort_buffer_size(handle,
    m,
    n,
    nnz,
    coo_row_ind,
    coo_col_ind,
    &buffer_size);
hipMalloc(&temp_buffer, buffer_size);

// Sort the COO matrix
rocsparse_coosort_by_column(handle,
    m,
    n,
    nnz,
    coo_row_ind,
    coo_col_ind,
    perm,
    (continues on next page)
temp_buffer);

// Gather sorted coo_val array
float* coo_val_sorted;
hipMalloc((void**)&coo_val_sorted, sizeof(float) * nnz);
rocsparse_sgthr(handle, nnz, coo_val, coo_val_sorted, perm, rocsparse_index_base_zero);

// Clean up
hipFree(temp_buffer);
hipFree(perm);
hipFree(coo_val);

Parameters

- [in] handle: handle to the rocsparse library context queue.
- [in] m: number of rows of the sparse COO matrix.
- [in] n: number of columns of the sparse COO matrix.
- [in] nnz: number of non-zero entries of the sparse COO matrix.
- [inout] coo_row_ind: array of nnz elements containing the row indices of the sparse COO matrix.
- [inout] coo_col_ind: array of nnz elements containing the column indices of the sparse COO matrix.
- [inout] perm: array of nnz integers containing the unsorted map indices, can be NULL.
- [in] temp_buffer: temporary storage buffer allocated by the user, size is returned by rocsparse_coosort_buffer_size().

Return Value

- rocsparse_status_success: the operation completed successfully.
- rocsparse_status_invalid_handle: the library context was not initialized.
- rocsparse_status_invalid_size: m, n or nnz is invalid.
- rocsparse_status_invalid_pointer: coo_row_ind, coo_col_ind or temp_buffer pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.

2.12.6 rocSOLVER

2.12.6.1 Introduction

An implementation of Lapack routines on top of AMD’s Radeon Open Compute Platform (ROCm) runtime and toolchains. rocSOLVER is implemented in the HIP programming language; it is based on rocBLAS, an optimized BLAS implementation for AMD’s latest discrete GPUs. More information about rocBLAS can be found here.
rocSOLVER requires cmake and ROCm, including hip and rocBLAS, to be installed.

Once these requirements are satisfied, the following instructions will build and install rocSOLVER:

```
mkdir build && cd build
CXX=/opt/rocm/bin/hcc cmake ..
make
make install
```

rocSolver Library is in early stages of active development. New features and functionality is being continuously added. New functionality is documented at each release of the ROCm platform.

The following table summarizes the LAPACK functionality implemented in rocSOLVER’s last release.

<table>
<thead>
<tr>
<th>Lapack Auxiliary Function</th>
<th>single</th>
<th>double</th>
<th>single complex</th>
<th>double complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>rocsolver_laswp</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>rocsolver_larfkg</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_larft</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_larf</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_larftb</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_org2r</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_orgqru</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_orgl2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_orglq</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_orgbr</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_orm2r</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_ormqr</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Lapack Function

<table>
<thead>
<tr>
<th>Lapack Function</th>
<th>single</th>
<th>double</th>
<th>single complex</th>
<th>double complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>rocsolver_potf2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_potf2_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_potrf</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_potrf_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_getf2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>rocsolver_getf2_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_getrf</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_getrf_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_geqr2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_geqr2_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_geqrf</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_geqrf_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_gelq2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_gelq2_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_gelqf</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_gelqf_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_getrs</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>rocsolver_getrs_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rocsolver_getrs_strided_batched</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2.12.6.4 Benchmarking and Testing

Additionally, rocSOLVER has a basic/preliminary infrastructure for testing and benchmarking similar to that of rocBLAS.

On a normal installation, clients should be located in the directory `<rocsolverDIR>/build/clients/staging`. `rocsolver-test` executes a suite of Google tests (`gtest`) that verifies the correct functioning of the library; the results computed by rocSOLVER, for random input data, are compared with the results computed by NETLib LAPACK on the CPU.

Calling the rocSOLVER gtest client with the –help flag

```bash
./rocsolver-test --help
```

returns information on different flags that control the behavior of the gtests.

`rocsolver-bench` allows to run any rocSOLVER function with random data of the specified dimensions; it compares the computed results, and provides basic performance information (as for now, execution times).

Similarly,

```bash
./rocsolver-bench --help
```

returns information on how to use the rocSOLVER benchmark client.
2.12.6.5 rocSOLVER

This section provides details of the rocSOLVER library API as in release ROCm 2.10. Most rocSOLVER types are aliases of rocBLAS types. See rocBLAS types here.

2.12.6.5.1 Definitions

2.12.6.5.1.1 rocsolver_int

typedef rocblas_int rocsolver_int
Used to specify int32 or int64.
rocsolver_int is a rocblas_int

2.12.6.5.2 Enums

2.12.6.5.2.1 rocsolver_handle

typedef rocblas_handle rocsolver_handle
A structure holding the rocsolver library context.
It must be initialized using rocsolver_create_handle() and the returned handle must be passed to all subsequent library function calls. It should be destroyed at the end using rocsolver_destroy_handle(). rocsolver_handle is a rocblas_handle.

2.12.6.5.2.2 rocsolver_operation

typedef rocblas_operation rocsolver_operation
Used to specify whether the matrix is to be transposed.
rocsolver_operation is a rocblas_operation

2.12.6.5.2.3 rocsolver_fill

typedef rocblas_fill rocsolver_fill
Used to specify whether the upper or lower triangle in a matrix is referenced.
rocsolver_fill is a rocblas_fill

2.12.6.5.2.4 rocsolver_diagonal

typedef rocblas_diagonal rocsolver_diagonal
Used to specify whether a matrix has ones along the diagonal.
rocsolver_diagonal is a rocblas_diagonal
2.12.6.5.2.5 rocsolver_side

typedef rocblas_side rocsolver_side
   Used to specify whether matrix multiplication is done by the right or left.
   rocsolver_side is a rocblas_side

2.12.6.5.2.6 rocsolver_direct

enum rocsolver_direct
   Used to specify the order in which multiple elementary matrices are applied together.

   Values:

   enumerator rocsolver_forward_direction = 171
      Elementary matrices applied from the right.

   enumerator rocsolver_backward_direction = 172
      Elementary matrices applied from the left.

2.12.6.5.2.7 rocsolver_storev

enum rocsolver_storev
   Used to specify how householder vectors are stored in a matrix of vectors.

   Values:

   enumerator rocsolver_column_wise = 181
      Householder vectors are stored in the columns of a matrix.

   enumerator rocsolver_row_wise = 182
      Householder vectors are stored in the rows of a matrix.

2.12.6.5.2.8 rocsolver_status

typedef rocblas_status rocsolver_status
   The rocSOLVER status code definition.

   rocsolver_status is a rocblas_status

   These are functions that support more advanced Lapack routines.

2.12.6.5.3 Matrix permutations and manipulations

2.12.6.5.3.1 rocsolver_<type>laswp()

rocsolver_status rocsolver_zlaswp (rocsolver_handle handle, const rocsolver_int n, 
rocblas_double_complex *A, const rocblas_int lda, const 
rocblas_int k1, const rocsolver_int k2, const rocsolver_int *
ipiv, const rocsolver_int incx)

rocsolver_status rocsolver_claswp (rocsolver_handle handle, const rocsolver_int n, 
rocblas_float_complex *A, const rocsolver_int lda, const 
rocblas_int k1, const rocsolver_int k2, const rocsolver_int *
ipiv, const rocsolver_int incx)


Laswp performs a series of row interchanges on the matrix A.

It interchanges row I with row IPIV[k1 + (I - k1) * abs(inx)], for each of rows K1 through K2 of A. k1 and k2 are 1-based indices.

Parameters

- `[in] handle: rocsolver_handle`
- `[in] n: rocsolver_int. n >= 0. The number of columns of the matrix A.`
- `[inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the matrix of column dimension n to which the row interchanges will be applied. On exit, the permuted matrix.`
- `[in] lda: rocsolver_int. lda > 0. The leading dimension of the array A.`
- `[in] k1: rocsolver_int. k1 > 0. The first element of IPIV for which a row interchange will be done. This is a 1-based index.`
- `[in] k2: rocsolver_int. k2 > k1 > 0. (K2-K1+1) is the number of elements of IPIV for which a row interchange will be done. This is a 1-based index.`
- `[in] ipiv: pointer to rocsolver_int. Array on the GPU of dimension at least k1 + (k2 - k1) * abs(inx). The vector of pivot indices. Only the elements in positions k1 through (k1 + (k2 - k1) * abs(inx)) of IPIV are accessed. Elements of ipiv are considered 1-based.`
- `[in] incx: rocsolver_int. incx != 0. The increment between successive values of IPIV. If IPIV is negative, the pivots are applied in reverse order."

2.12.5.4 Householder reflexions

2.12.5.4.1 rocsolver_<type>larfg()  

Larfg generates an orthogonal Householder reflector H of order n.

Householder reflector H is such that

\[
H \times \begin{bmatrix} \alpha \\ x \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \end{bmatrix}
\]

where x is an n-1 vector and alpha and beta are scalars. Matrix H can be generated as

\[
H = I - \tau \times \begin{bmatrix} 1 \\ v \end{bmatrix} \times \begin{bmatrix} 1 & v' \\ v \end{bmatrix}
\]

with v an n-1 vector and tau a scalar.

Parameters
• [in] handle: rocsolver_handle
• [in] n: rocsolver_int. n >= 0. The order (size) of reflector H.
• [inout] alpha: pointer to type. A scalar on the GPU. On input the scalar alpha, on output it is overwritten with beta.
• [inout] x: pointer to type. Array on the GPU of size at least n-1. On input it is the vector x, on output it is overwritten with vector v.
• [in] incx: rocsolver_int. incx > 0. The increment between consecutive elements of x.
• [out] tau: pointer to type. A scalar on the GPU. The scalar tau.

2.12.6.5.4.2 rocsolver_<type>larft()

rocsolver_status rocsolver_dlarft (rocsolver_handle handle, const rocsolver_direct direct, const rocsolver_storev storev, const rocsolver_int n, const rocsolver_int k, double *V, const rocsolver_int ldv, double *tau, double *T, const rocsolver_int ldt)

rocsolver_status rocsolver_slarft (rocsolver_handle handle, const rocsolver_direct direct, const rocsolver_storev storev, const rocsolver_int n, const rocsolver_int k, float *V, const rocsolver_int ldv, float *tau, float *T, const rocsolver_int ldt)

LARFT Generates the triangular factor T of a block reflector H of order n.

The block reflector H is defined as the product of k Householder matrices as

\[ H = H(1) \times H(2) \times \ldots \times H(k) \] (forward direction), or
\[ H = H(k) \times \ldots \times H(2) \times H(1) \] (backward direction)

depending on the value of direct.

The triangular matrix T is upper triangular in forward direction and lower triangular in backward direction. If storev is column-wise, then

\[ H = I - V \times T \times V' \]

where the i-th column of matrix V contains the Householder vector associated to H(i). If storev is row-wise, then

\[ H = I - V' \times T \times V \]

where the i-th row of matrix V contains the Householder vector associated to H(i).

Parameters

• [in] handle: rocsolver_handle.
• [in] direct: rocsolver_direct. Specifies the direction in which the Householder matrices are applied.
• [in] storev: rocsolver_storev. Specifies how the Householder vectors are stored in matrix V.
• [in] n: rocsolver_int. n >= 0. The order (size) of the block reflector.
• [in] k: rocsolver_int. k >= 1. The number of Householder matrices.
• [in] V: pointer to type. Array on the GPU of size ldv*k if column-wise, or ldv*n if row-wise. The matrix of Householder vectors.
• [in] ldv: rocsolver_int. ldv >= n if column-wise, or ldv >= k if row-wise. Leading dimension of V.

• [in] tau: pointer to type. Array of k scalars on the GPU. The vector of all the scalars associated to the Householder matrices.

• [out] T: pointer to type. Array on the GPU of dimension ldt*k. The triangular factor. T is upper triangular if forward operation, otherwise it is lower triangular. The rest of the array is not used.

• [in] ldt: rocsolver_int. ldt >= k. The leading dimension of T.

2.12.6.5.4.3 rocsolver_<type>_larf()

rocsolver_status rocsolver_dlarf (rocsolver_handle handle, const rocsolver_side side, const rocsolver_int m, const rocsolver_int n, double *x, const rocsolver_int incx, const double *alpha, double *A, const rocsolver_int lda)

rocsolver_status rocsolver_larf (rocsolver_handle handle, const rocsolver_side side, const rocsolver_int m, const rocsolver_int n, float *x, const rocsolver_int incx, const float *alpha, float *A, const rocsolver_int lda)

LARF applies a Householder reflector H to a general matrix A.

The Householder reflector H, of order m (or n), is to be applied to a m-by-n matrix A from the left (or the right). H is given by

\[ H = I - \alpha \cdot x \cdot x' \]

where alpha is a scalar and x a Householder vector. H is never actually computed.

Parameters

• [in] handle: rocsolver_handle.

• [in] side: rocsolver_side. If side = rocsolver_side_left, then compute H*A If side = rocsolver_side_right, then compute A*H

• [in] m: rocsolver_int. m >= 0. Number of rows of A.

• [in] n: rocsolver_int. n >= 0. Number of columns of A.

• [in] x: pointer to type. Array on the GPU of size at least (1 + (m-1)*abs(incx)) if left side, or at least (1 + (n-1)*abs(incx)) if right side. The Householder vector x.

• [in] incx: rocsolver_int. incx != 0. Increment between to consecutive elements of x. If incx < 0, the elements of x are used in reverse order.

• [in] alpha: pointer to type. A scalar on the GPU. If alpha = 0, then H = I (A will remain the same, x is never used)

• [inout] A: pointer to type. Array on the GPU of size lda*n. On input, the matrix A. On output it is overwritten with H*A (or A*H).

• [in] lda: rocsolver_int. lda >= m. Leading dimension of A.
2.12.6.5.4 rocsolver_<type>larfb()

rocsolver_status rocsolver_dlarfb (rocsolver_handle handle, const rocsolver_side side, const rocsolver_operation trans, const rocsolver_direct direct, const rocsolver_storev storev, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *V, const rocsolver_int ldv, double *T, const rocsolver_int ldt, double *A, const rocsolver_int lda)

rocsolver_status rocsolver_slarfb (rocsolver_handle handle, const rocsolver_side side, const rocsolver_operation trans, const rocsolver_direct direct, const rocsolver_storev storev, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *V, const rocsolver_int ldv, float *T, const rocsolver_int ldt, float *A, const rocsolver_int lda)

LARFB applies a block reflector $H$ to a general $m$-by-$n$ matrix $A$.

The block reflector $H$ is applied in one of the following forms, depending on the values of side and trans:

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H \times A$</td>
<td>(No transpose from the left)</td>
</tr>
<tr>
<td>$H' \times A$</td>
<td>(Transpose from the left)</td>
</tr>
<tr>
<td>$A \times H'$</td>
<td>(No transpose from the right), and</td>
</tr>
<tr>
<td>$A \times H$</td>
<td>(Transpose from the right)</td>
</tr>
</tbody>
</table>

The block reflector $H$ is defined as the product of $k$ Householder matrices as

- $H = H(1) \times H(2) \times \ldots \times H(k)$ (forward direction), or
- $H = H(k) \times \ldots \times H(2) \times H(1)$ (backward direction)

depending on the value of direct. $H$ is never stored. It is calculated as

$$H = I - V \times T \times V'$$

where the $i$-th column of matrix $V$ contains the Householder vector associated to $H(i)$, if storev is column-wise; or

$$H = I - V' \times T \times V$$

where the $i$-th row of matrix $V$ contains the Householder vector associated to $H(i)$, if storev is row-wise. $T$ is the associated triangular factor as computed by LARFT.

Parameters

-  [in] handle: rocsolver_handle.
-  [in] side: rocsolver_side. Specifies from which side to apply $H$.
-  [in] trans: rocsolver_operation. Specifies whether the block reflector or its transpose is to be applied.
-  [in] direct: rocsolver_direct. Specifies the direction in which the Householder matrices were to be applied to generate $H$.
-  [in] m: rocsolver_int. $m \geq 0$. Number of rows of matrix $A$.
-  [in] n: rocsolver_int. $n \geq 0$. Number of columns of matrix $A$.
-  [in] k: rocsolver_int. $k \geq 1$. The number of Householder matrices.
• \( \text{[in]} \ V: \) pointer to type. Array on the GPU of size \( \text{ldv} \times k \) if column-wise, \( \text{ldv} \times n \) if row-wise and applying from the right, or \( \text{ldv} \times m \) if row-wise and applying from the left. The matrix of Householder vectors.

• \( \text{[in]} \ \text{ldv}: \) rocsolver_int. \( \text{ldv} \geq k \) if row-wise, \( \text{ldv} \geq m \) if column-wise and applying from the left, or \( \text{ldv} \geq n \) if column-wise and applying from the right. Leading dimension of \( V \).

• \( \text{[in]} \ T: \) pointer to type. Array on the GPU of dimension \( \text{ldt} \times k \). The triangular factor of the block reflector.

• \( \text{[in]} \ \text{ldt}: \) rocsolver_int. \( \text{ldt} \geq k \). The leading dimension of \( T \).

• \( \text{[inout]} \ A: \) pointer to type. Array on the GPU of dimension \( \text{lda} \times n \). On input, the matrix \( A \). On output it is overwritten with \( H \times A, A \times H, H^* \times A, \) or \( A \times H^* \).

• \( \text{[in]} \ \text{lda}: \) rocsolver_int. \( \text{lda} \geq m \). Leading dimension of \( A \).

2.12.6.5.5 Orthonormal matrices

2.12.6.5.5.1 rocsolver_<type>_org2r()  

rocsolver_status rocsolver_dorg2r(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv)  

rocsolver_status rocsolver_sorg2r(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv)

ORG2R generates a \( m \)-by-\( n \) Matrix \( Q \) with orthonormal columns. (This is the unblocked version of the algorithm).

The matrix \( Q \) is defined as the first \( n \) columns of the product of \( k \) Householder reflectors of order \( m \)

\[
Q = H(1) \times H(2) \times \ldots \times H(k)
\]

Householder matrices \( H(i) \) are never stored, they are computed from its corresponding Householder vector \( v(i) \) and scalar \( \text{ipiv}_i \) as returned by GEQRF.

Parameters

• \( \text{[in]} \ \text{handle}: \) rocsolver_handle.

• \( \text{[in]} \ m: \) rocsolver_int. \( m \geq 0 \). The number of rows of the matrix \( Q \).

• \( \text{[in]} \ n: \) rocsolver_int. \( 0 \leq n \leq m \). The number of columns of the matrix \( Q \).

• \( \text{[in]} \ k: \) rocsolver_int. \( 0 \leq k \leq n \). The number of Householder reflectors.

• \( \text{[inout]} \ A: \) pointer to type. Array on the GPU of dimension \( \text{lda} \times n \). On entry, the \( i \)-th column has Householder vector \( v(i) \), for \( i = 1,2,\ldots,k \) as returned in the first \( k \) columns of matrix \( A \) of GEQRF. On exit, the computed matrix \( Q \).

• \( \text{[in]} \ \text{lda}: \) rocsolver_int. \( \text{lda} \geq m \). Specifies the leading dimension of \( A \).

• \( \text{[in]} \ \text{ipiv}: \) pointer to type. Array on the GPU of dimension at least \( k \). The scalar factors of the Householder matrices \( H(i) \) as returned by GEQRF.
2.12.6.5.5.2 rocsolver_<type>orgqr()

```
rocsolver_status rocsolver_dorgqr (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv)
rocsolver_status rocsolver_sorgqr (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv)
```

ORGQR generates a m-by-n Matrix Q with orthonormal columns.

(This is the blocked version of the algorithm).

The matrix Q is defined as the first n columns of the product of k Householder reflectors of order m

\[ Q = H(k) \times H(k-1) \times \ldots \times H(1) \]

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv_i as returned by GEQRF.

**Parameters**

- [in] handle: rocsolver_handle.
- [in] m: rocsolver_int. m >= 0. The number of rows of the matrix Q.
- [in] n: rocsolver_int. 0 <= n <= m. The number of columns of the matrix Q.
- [in] k: rocsolver_int. 0 <= k <= n. The number of Householder reflectors.
- [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the i-th column has Householder vector v(i), for i = 1,2,\ldots,k as returned in the first k columns of matrix A of GEQRF. On exit, the computed matrix Q.
- [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
- [in] ipiv: pointer to type. Array on the GPU of dimension at least k. The scalar factors of the Householder matrices H(i) as returned by GEQRF.

2.12.6.5.5.3 rocsolver_<type>orgl2()

```
rocsolver_status rocsolver_dorgl2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv)
rocsolver_status rocsolver_sorgl2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv)
```

ORGL2 generates a m-by-n Matrix Q with orthonormal rows.

(This is the unblocked version of the algorithm).

The matrix Q is defined as the first m rows of the product of k Householder reflectors of order n

\[ Q = H(k) \times H(k-1) \times \ldots \times H(1) \]

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv_i as returned by GELQF.

**Parameters**
• [in] handle: rocsolver_handle.
• [in] m: rocsolver_int. 0 <= m <= n. The number of rows of the matrix Q.
• [in] n: rocsolver_int. n >= 0. The number of columns of the matrix Q.
• [in] k: rocsolver_int. 0 <= k <= m. The number of Householder reflectors.
• [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the i-th row has Householder vector v(i), for i = 1,2,...,k as returned in the first k rows of matrix A of GELQF. On exit, the computed matrix Q.
• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
• [in] ipiv: pointer to type. Array on the GPU of dimension at least k. The scalar factors of the Householder matrices H(i) as returned by GELQF.

2.12.6.5.5.4 rocsolver_<type>orglq()

rocsolver_status rocsolver_dorglq(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv)

rocsolver_status rocsolver_sorglq(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv)

ORGLQ generates a m-by-n Matrix Q with orthonormal rows.

(This is the blocked version of the algorithm).

The matrix Q is defined as the first m rows of the product of k Householder reflectors of order n

\[ Q = H(k) \times H(k-1) \times \ldots \times H(1) \]

Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv_i as returned by GELQF.

Parameters

• [in] handle: rocsolver_handle.
• [in] m: rocsolver_int. 0 <= m <= n. The number of rows of the matrix Q.
• [in] n: rocsolver_int. n >= 0. The number of columns of the matrix Q.
• [in] k: rocsolver_int. 0 <= k <= m. The number of Householder reflectors.
• [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the i-th row has Householder vector v(i), for i = 1,2,...,k as returned in the first k rows of matrix A of GELQF. On exit, the computed matrix Q.
• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
• [in] ipiv: pointer to type. Array on the GPU of dimension at least k. The scalar factors of the Householder matrices H(i) as returned by GELQF.
2.12.6.5.5 rocsolver_<type>orgbr()

rocsolver_status rocsolver_dorgbr(rocsolver_handle handle, const rocsolver_storev storev, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv)

rocsolver_status rocsolver_sorgbr(rocsolver_handle handle, const rocsolver_storev storev, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv)

ORGBR generates a m-by-n Matrix Q with orthonormal rows or columns.

If storev is column-wise, then the matrix Q has orthonormal columns. If m >= k, Q is defined as the first n columns of the product of k Householder reflectors of order m

\[ Q = H(1) \times H(2) \times \ldots \times H(k) \]

If m < k, Q is defined as the product of Householder reflectors of order m

\[ Q = H(1) \times H(2) \times \ldots \times H(m-1) \]

On the other hand, if storev is row-wise, then the matrix Q has orthonormal rows. If n > k, Q is defined as the first m rows of the product of k Householder reflectors of order n

\[ Q = H(k) \times H(k-1) \times \ldots \times H(1) \]

If n <= k, Q is defined as the product of Householder reflectors of order n

\[ Q = H(n-1) \times H(n-2) \times \ldots \times H(1) \]

The Householder matrices H(i) are never stored, they are computed from its corresponding Householder vector v(i) and scalar ipiv_i as returned by GEBRD.

Parameters

- [in] handle: rocsolver_handle.
- [in] storev: rocsolver_storev. Specifies whether to work column-wise or row-wise.
- [in] m: rocsolver_int. m >= 0. The number of rows of the matrix Q. If row-wise, then min(n,k) <= m <= n.
- [in] n: rocsolver_int. n >= 0. The number of columns of the matrix Q. If column-wise, then min(m,k) <= n <= m.
- [in] k: rocsolver_int. k >= 0. The number of columns (if storev is column-wise) or rows (if row-wise) of the original matrix reduced by GEBRD.
- [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the i-th column (or row) has the Householder vector v(i) as returned by GEBRD. On exit, the computed matrix Q.
- [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
- [in] ipiv: pointer to type. Array on the GPU of dimension min(m,k) if column-wise, or min(n,k) if row-wise. The scalar factors of the Householder matrices H(i) as returned by GEBRD.
2.12.5.5.6 rocsolver_<type>orm2r()

rocsolver_status rocsolver_dorm2r(rocsolver_handle handle, const rocsolver_side side, const rocsolver_operation trans, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv, double *C, const rocsolver_int ldc)

rocsolver_status rocsolver_sorm2r(rocsolver_handle handle, const rocsolver_side side, const rocsolver_operation trans, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv, float *C, const rocsolver_int ldc)

ORM2R applies a matrix Q with orthonormal columns to a general m-by-n matrix C. (This is the unblocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q * C</td>
<td>(No transpose from the left)</td>
</tr>
<tr>
<td>Q’ * C</td>
<td>(Transpose from the left)</td>
</tr>
<tr>
<td>C * Q</td>
<td>(No transpose from the right), and</td>
</tr>
<tr>
<td>C * Q’</td>
<td>(Transpose from the right)</td>
</tr>
</tbody>
</table>

Q is an orthogonal matrix defined as the product of k Householder reflectors as

\[ Q = H(1) \times H(2) \times ... \times H(k) \]

or order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QR factorization GEQRF.

Parameters

- [in] handle: rocsolver_handle.
- [in] side: rocsolver_side. Specifies from which side to apply Q.
- [in] trans: rocsolver_operation. Specifies whether the matrix Q or its transpose is to be applied.
- [in] m: rocsolver_int. m >= 0. Number of rows of matrix C.
- [in] n: rocsolver_int. n >= 0. Number of columns of matrix C.
- [in] k: rocsolver_int. k >= 0; k <= m if side is left, k <= n if side is right. The number of Householder reflectors that form Q.
- [in] A: pointer to type. Array on the GPU of size lda*k. The i-th column has the Householder vector v(i) associated with H(i) as returned by GEQRF in the first k columns of its argument A.
- [in] lda: rocsolver_int. lda >= m if side is left, or lda >= n if side is right. Leading dimension of A.
- [in] ipiv: pointer to type. Array on the GPU of dimension at least k. The scalar factors of the Householder matrices H(i) as returned by GEQRF.
- [inout] C: pointer to type. Array on the GPU of size ldc*n. On input, the matrix C. On output it is overwritten with Q*C, C*Q, Q’*C, or C*Q’.
- [in] lda: rocsolver_int. ldc >= m. Leading dimension of C.
2.12.6.5.5.7 rocsolver_<type>ormqr()

**rocsolver_status rocsolver_dormqr**

\[
\text{rocsolver_dormqr} \left( \text{rocsolver_handle handle, const rocsolver_side side, const rocsolver_operation trans, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, double *A, const rocsolver_int lda, double *ipiv, double *C, const rocsolver_int ldc} \right)
\]

**rocsolver_status rocsolver_sormqr**

\[
\text{rocsolver_sormqr} \left( \text{rocsolver_handle handle, const rocsolver_side side, const rocsolver_operation trans, const rocsolver_int m, const rocsolver_int n, const rocsolver_int k, float *A, const rocsolver_int lda, float *ipiv, float *C, const rocsolver_int ldc} \right)
\]

ORMQR applies a matrix Q with orthonormal columns to a general m-by-n matrix C.

(This is the blocked version of the algorithm).

The matrix Q is applied in one of the following forms, depending on the values of side and trans:

- \( Q \times C \) (No transpose from the left)
- \( Q' \times C \) (Transpose from the left)
- \( C \times Q \) (No transpose from the right), and
- \( C \times Q' \) (Transpose from the right)

Q is an orthogonal matrix defined as the product of k Householder reflectors as

\[
Q = H(1) \times H(2) \times \ldots \times H(k)
\]

or order m if applying from the left, or n if applying from the right. Q is never stored, it is calculated from the Householder vectors and scalars returned by the QR factorization GEQRF.

**Parameters**

- **[in]** handle: rocsolver_handle.
- **[in]** side: rocsolver_side. Specifies from which side to apply Q.
- **[in]** trans: rocsolver_operation. Specifies whether the matrix Q or its transpose is to be applied.
- **[in]** m: rocsolver_int. m >= 0. Number of rows of matrix C.
- **[in]** n: rocsolver_int. n >= 0. Number of columns of matrix C.
- **[in]** k: rocsolver_int. k >= 0; k <= m if side is left, k <= n if side is right. The number of Householder reflectors that form Q.
- **[in]** A: pointer to type. Array on the GPU of size lda*k. The i-th column has the Householder vector v(i) associated with H(i) as returned by GEQRF in the first k columns of its argument A.
- **[in]** lda: rocsolver_int. lda >= m if side is left, or lda >= n if side is right. Leading dimension of A.
- **[in]** ipiv: pointer to type. Array on the GPU of dimension at least k. The scalar factors of the Householder matrices H(i) as returned by GEQRF.
- **[inout]** C: pointer to type. Array on the GPU of size ldc*n. On input, the matrix C. On output it is overwritten with \( Q^*C, C^*Q, Q^*C, \) or \( C^*Q' \).
- **[in]** lda: rocsolver_int. ldc >= m. Leading dimension of C.

Lapack routines solve complex Numerical Linear Algebra problems.
### 2.12.6.5.6 Special Matrix Factorizations

#### 2.12.6.5.6.1 rocsolver_<type>potf2()

-rocsolver_status rocsolver_dpotf2 (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, double *A, const rocsolver_int lda, rocblas_int *info) -rocsolver_status rocsolverスポットト2 (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, float *A, const rocsolver_int lda, rocblas_int *info)

POTF2 computes the Cholesky factorization of a real symmetric positive definite matrix $A$.

(This is the unblocked version of the algorithm).

The factorization has the form:

$$ A = U' * U, \text{ or } A = L * L' $$

depending on the value of uplo. $U$ is an upper triangular matrix and $L$ is lower triangular.

**Parameters**

- [in] handle: rocsolver_handle.
- [in] uplo: rocsolver_fill. Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of $A$ is not used.
- [in] n: rocsolver_int. n >= 0. The matrix dimensions.
- [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the matrix $A$ to be factored. On exit, the lower or upper triangular factor.
- [in] lda: rocsolver_int. lda >= n. specifies the leading dimension of $A$.
- [out] info: pointer to a rocsolver_int on the GPU. If info = 0, successful factorization of matrix $A$. If info = i > 0, the leading minor of order i of $A$ is not positive definite. The factorization stopped at this point.

#### 2.12.6.5.6.2 rocsolver_<type>potf2_batched()

-rocsolver_status rocsolver_dpotf2_batched (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, double *const A[], const rocsolver_int lda, rocblas_int *info, const rocsolver_int batch_count) -rocsolver_status rocsolverスポットト2_batched (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, float *const A[], const rocsolver_int lda, rocblas_int *info, const rocsolver_int batch_count)

POTF2_BATCHED computes the Cholesky factorization of a batch of real symmetric positive definite matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix $A_{i}$ in the batch has the form:

$$ A_{i} = U_{i}' * U_{i}, \text{ or } A_{i} = L_{i} * L_{i}' $$

depending on the value of uplo. $U_{i}$ is an upper triangular matrix and $L_{i}$ is lower triangular.
Parameters

- [in] handle: rocsolver_handle.
- [in] uplo: rocsolver_fill. Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- [in] n: rocsolver_int. n >= 0. The dimension of matrix A_i.
- [inout] A: array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the matrices A_i to be factored. On exit, the upper or lower triangular factors.
- [in] lda: rocsolver_int. lda >= n. specifies the leading dimension of A_i.
- [out] info: pointer to rocsolver_int. Array of batch_count integers on the GPU. If info_i = 0, successful factorization of matrix A_i. If info_i = j > 0, the leading minor of order j of A_i is not positive definite. The i-th factorization stopped at this point.
- [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.6.5.6.3 rocsolver_<type>_potf2_strided_batched()

roc solver_status rocsolver_dpotf2_strided_batched (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, rocblas_int *info, const rocsolver_int batch_count)

roc solver_status rocsolverスポットf2_strided_batched (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, rocblas_int *info, const rocsolver_int batch_count)

POTF2_STRIDED_BATCHED computes the Cholesky factorization of a batch of real symmetric positive definite matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix A_i in the batch has the form:

\[
A_i = U_i' \times U_i, \text{ or } \quad A_i = L_i \times L_i'
\]

depending on the value of uplo. U_i is an upper triangular matrix and L_i is lower triangular.

Parameters

- [in] handle: rocsolver_handle.
- [in] uplo: rocsolver_FILL. Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- [in] n: rocsolver_int. n >= 0. The dimension of matrix A_i.
- [inout] A: pointer to type. Array on the GPU (the size depends on the value of strideA). On entry, the matrices A_i to be factored. On exit, the upper or lower triangular factors.
- [in] lda: rocsolver_int. lda >= n. specifies the leading dimension of A_i.
- [in] strideA: rocsolver_int. Stride from the start of one matrix A_i and the next one A_(i+1). There is no restriction for the value of strideA. Normal use case is strideA >= lda*n.
• \([\text{out}]\) \(\text{info}\): pointer to \(\text{rocsolver\_int}\). Array of \(\text{batch\_count}\) integers on the GPU. If \(\text{info}_i = 0\), successful factorization of matrix \(A_i\). If \(\text{info}_i = j > 0\), the leading minor of order \(j\) of \(A_i\) is not positive definite. The \(i\)-th factorization stopped at this point.

• \([\text{in}]\) \(\text{batch\_count}\): \(\text{rocsolver\_int}\). \(\text{batch\_count} \geq 0\). Number of matrices in the batch.

### 2.12.6.5.6.4 \(\text{rocsolver\_<type>potrf()}\)

\(\text{rocsolver\_status rocsolver\_dpotrf}(\text{rocsolver\_handle handle, const rocsolver\_fill uplo, const rocsolver\_int n, double *A, const rocsolver\_int lda, rocblas\_int *info})\)

\(\text{rocsolver\_status rocsolver\_spotrf}(\text{rocsolver\_handle handle, const rocsolver\_fill uplo, const rocsolver\_int n, float *A, const rocsolver\_int lda, rocblas\_int *info})\)

\(\text{POTRF}\) computes the Cholesky factorization of a real symmetric positive definite matrix \(A\).

(This is the blocked version of the algorithm).

The factorization has the form:

\[
A = U' \times U, \text{ or } A = L \times L'
\]

depending on the value of \(\text{uplo}\). \(U\) is an upper triangular matrix and \(L\) is lower triangular.

**Parameters**

• \([\text{in}]\) \(\text{handle}\): \(\text{rocsolver\_handle}\).

• \([\text{in}]\) \(\text{uplo}\): \(\text{rocsolver\_fill}\). Specifies whether the factorization is upper or lower triangular. If \(\text{uplo}\) indicates lower (or upper), then the upper (or lower) part of \(A\) is not used.

• \([\text{in}]\) \(\text{n}\): \(\text{rocsolver\_int}\). \(\text{n} \geq 0\). The matrix dimensions.

• \([\text{inout}]\) \(A\): pointer to \(\text{type}\). Array on the GPU of dimension \(\text{lda} \times \text{n}\). On entry, the matrix \(A\) to be factored. On exit, the lower or upper triangular factor.

• \([\text{in}]\) \(\text{lda}\): \(\text{rocsolver\_int}\). \(\text{lda} \geq \text{n}\). specifies the leading dimension of \(A\).

• \([\text{out}]\) \(\text{info}\): pointer to a \(\text{rocsolver\_int}\) on the GPU. If \(\text{info} = 0\), succesful factorization of matrix \(A\). If \(\text{info} = i > 0\), the leading minor of order \(i\) of \(A\) is not positive definite. The factorization stopped at this point.

### 2.12.6.5.6.5 \(\text{rocsolver\_<type>potrf\_batched()}\)

\(\text{rocsolver\_status rocsolver\_dpotrf\_batched}(\text{rocsolver\_handle handle, const rocsolver\_fill uplo, const rocsolver\_int n, double const A[], const rocsolver\_int lda, rocblas\_int *info, const rocsolver\_int batch\_count})\)

\(\text{rocsolver\_status rocsolver\_spotrf\_batched}(\text{rocsolver\_handle handle, const rocsolver\_fill uplo, const rocsolver\_int n, float const A[], const rocsolver\_int lda, rocblas\_int *info, const rocsolver\_int batch\_count})\)

\(\text{POTRF\_BATCHED}\) computes the Cholesky factorization of a batch of real symmetric positive definite matrices.

(This is the blocked version of the algorithm).

The factorization of matrix \(A_i\) in the batch has the form:
A_i = U_i' * U_i, or
A_i = L_i * L_i'

depending on the value of uplo. U_i is an upper triangular matrix and L_i is lower triangular.

Parameters

- [in] handle: rocsolver_handle.
- [in] uplo: rocsolver_fill. Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- [in] n: rocsolver_int. n >= 0. The dimension of matrix A_i.
- [inout] A: array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the matrices A_i to be factored. On exit, the upper or lower triangular factors.
- [in] lda: rocsolver_int. lda >= n. specifies the leading dimension of A_i.
- [out] info: pointer to rocsolver_int. Array of batch_count integers on the GPU. If info_i = 0, successful factorization of matrix A_i. If info_i = j > 0, the leading minor of order j of A_i is not positive definite. The i-th factorization stopped at this point.
- [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.6.5.6.6 rocsolver_<type>potrf_strided_batched()

rocsolver_status rocsolver_dpotrf_strided_batched (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *info, const rocsolver_int batch_count)

rocsolver_status rocsolver_spotrf_strided_batched (rocsolver_handle handle, const rocsolver_fill uplo, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, const rocsolver_int *info, const rocsolver_int batch_count)

POTRF_STRIDED_BATCHED computes the Cholesky factorization of a batch of real symmetric positive definite matrices.

(This is the blocked version of the algorithm).

The factorization of matrix A_i in the batch has the form:

A_i = U_i' * U_i, or
A_i = L_i * L_i'

depending on the value of uplo. U_i is an upper triangular matrix and L_i is lower triangular.

Parameters

- [in] handle: rocsolver_handle.
- [in] uplo: rocsolver_fill. Specifies whether the factorization is upper or lower triangular. If uplo indicates lower (or upper), then the upper (or lower) part of A is not used.
- [in] n: rocsolver_int. n >= 0. The dimension of matrix A_i.
• [inout] A: pointer to type. Array on the GPU (the size depends on the value of strideA). On entry, the matrices A_i to be factored. On exit, the upper or lower triangular factors.

• [in] lda: rocsolver_int. lda >= n. specifies the leading dimension of A_i.

• [in] strideA: rocsolver_int. Stride from the start of one matrix A_i and the next one A_(i+1). There is no restriction for the value of strideA. Normal use case is strideA >= lda*n.

• [out] info: pointer to rocsolver_int. Array of batch_count integers on the GPU. If info_i = 0, succesful factorization of matrix A_i. If info_i = j > 0, the leading minor of order j of A_i is not positive definite. The i-th factorization stopped at this point.

• [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.6.5.7 General Matrix Factorizations

2.12.6.5.7.1 rocsolver_<type>getf2()

rocsolver_status rocsolver_zgetf2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_double_complex *A, const rocsolver_int lda, rocsolver_int *ipiv, rocsolver_int *info)

rocsolver_status rocsolver_cgetf2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_float_complex *A, const rocsolver_int lda, rocsolver_int *ipiv, rocsolver_int *info)

rocsolver_status rocsolver_dgetf2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, rocsolver_int *ipiv, rocsolver_int *info)

rocsolver_status rocsolver_sgetf2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, rocsolver_int *ipiv, rocsolver_int *info)

GETF2 computes the LU factorization of a general m-by-n matrix A using partial pivoting with row interchanges.

(This is the right-looking Level 2 BLAS version of the algorithm).

The factorization has the form

\[ A = P \times L \times U \]

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

Parameters

• [in] handle: rocsolver_handle.

• [in] m: rocsolver_int. m >= 0. The number of rows of the matrix A.

• [in] n: rocsolver_int. n >= 0. The number of columns of the matrix A.

• [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the m-by-n matrix A to be factored. On exit, the factors L and U from the factorization. The unit diagonal elements of L are not stored.

• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
• [out] ipiv: pointer to rocsolver_int. Array on the GPU of dimension min(m,n). The vector of pivot indices. Elements of ipiv are 1-based indices. For 1 <= i <= min(m,n), the row i of the matrix was interchanged with row ipiv[i]. Matrix P of the factorization can be derived from ipiv.

• [out] info: pointer to a rocsolver_int on the GPU. If info = 0, successful exit. If info = i > 0, U is singular. U(i,i) is the first zero pivot.

2.12.6.5.7.2 rocsolver_<type>getf2_batched()

rocsolver_status rocsolver_zgetf2_batched(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_double_complex *const A[], const rocsolver_int lda, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)

rocsolver_status rocsolver_cgetf2_batched(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_float_complex *const A[], const rocsolver_int lda, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)

rocsolver_status rocsolver_dgetf2_batched(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *const A[], const rocsolver_int lda, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)

rocsolver_status rocsolver_sgetf2_batched(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *const A[], const rocsolver_int lda, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)

GETF2_BATCHED computes the LU factorization of a batch of general m-by-n matrices using partial pivoting with row interchanges.

(This is the right-looking Level 2 BLAS version of the algorithm).

The factorization of matrix A_i in the batch has the form

\[ A_i = P_i \cdot L_i \cdot U_i \]

where P_i is a permutation matrix, L_i is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U_i is upper triangular (upper trapezoidal if m < n).

Parameters

• [in] handle: rocsolver_handle.

• [in] m: rocsolver_int. m >= 0. The number of rows of all matrices A_i in the batch.

• [in] n: rocsolver_int. n >= 0. The number of columns of all matrices A_i in the batch.

• [inout] A: array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the m-by-n matrices A_i to be factored. On exit, the factors L_i and U_i from the factorizations. The unit diagonal elements of L_i are not stored.

• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_i.

• [out] ipiv: pointer to rocsolver_int. Array on the GPU (the size depends on the value of strideP). Contains the vectors of pivot indices ipiv_i (corresponding to A_i). Dimension of ipiv_i is min(m,n).
Elements of ipiv_i are 1-based indices. For each instance A_i in the batch and for 1 <= j <= min(m,n), the row j of the matrix A_i was interchanged with row ipiv_i[j]. Matrix P_i of the factorization can be derived from ipiv_i.

- [in] strideP: rocsolver_int. Stride from the start of one vector ipiv_i to the next one ipiv_(i+1). There is no restriction for the value of strideP. Normal use case is strideP >= min(m,n).
- [out] info: pointer to rocsolver_int. Array of batch_count integers on the GPU. If info_i = 0, successful exit for factorization of A_i. If info_i = j > 0, U_i is singular. U_i(j,j) is the first zero pivot.
- [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.6.5.7.3 rocsolver_<type>getf2_strided_batched()

rocsolver_status rocsolver_zgetf2_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_double_complex *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
rocsolver_status rocsolver_cgetf2_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_float_complex *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
rocsolver_status rocsolver_dgetf2_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
rocsolver_status rocsolver_sgetf2_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)

GETF2_STRIDED_BATCHED computes the LU factorization of a batch of general m-by-n matrices using partial pivoting with row interchanges.

(This is the right-looking Level 2 BLAS version of the algorithm).

The factorization of matrix A_i in the batch has the form

\[ A_i = P_i \cdot L_i \cdot U_i \]

where P_i is a permutation matrix, L_i is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U_i is upper triangular (upper trapezoidal if m < n).

Parameters

- [in] handle: rocsolver_handle.
ReadTheDocs-Breathe Documentation, Release 1.0.0

- [in] \( m \): \texttt{rocsolver_int}. \( m \geq 0 \). The number of rows of all matrices \( A_i \) in the batch.
- [in] \( n \): \texttt{rocsolver_int}. \( n \geq 0 \). The number of columns of all matrices \( A_i \) in the batch.
- [inout] \( A \): pointer to type. Array on the GPU (the size depends on the value of \texttt{strideA}). On entry, \( n \)-by-\( n \) matrices \( A_i \) to be factored. On exit, the factors \( L_i \) and \( U_i \) from the factorization. The unit diagonal elements of \( L_i \) are not stored.
- [in] \( \texttt{lda} \): \texttt{rocsolver_int}. \( \texttt{lda} \geq m \). Specifies the leading dimension of matrices \( A_i \).
- [in] \( \texttt{strideA} \): \texttt{rocsolver_int}. Stride from the start of one matrix \( A_i \) and the next one \( A_{(i+1)} \). There is no restriction for the value of \texttt{strideA}. Normal use case is \texttt{strideA} \( \geq \texttt{lda} \times n \).
- [out] \( \texttt{ipiv} \): pointer to \texttt{rocsolver_int}. Array on the GPU (the size depends on the value of \texttt{strideP}). Contains the vectors of pivots indices \( ipiv_i \) (corresponding to \( A_i \)). Dimension of \( ipiv_i \) is \( \min(m,n) \). Elements of \( ipiv_i \) are 1-based indices. For each instance \( A_i \) in the batch and for \( 1 \leq j \leq \min(m,n) \), the row \( j \) of the matrix \( A_i \) was interchanged with row \( ipiv_i[j] \). Matrix \( P_i \) of the factorization can be derived from \( ipiv_i \).
- [in] \( \texttt{strideP} \): \texttt{rocsolver_int}. Stride from the start of one vector \( ipiv_i \) to the next one \( ipiv_{(i+1)} \). There is no restriction for the value of \texttt{strideP}. Normal use case is \texttt{strideP} \( \geq \min(m,n) \).
- [out] \( \texttt{info} \): pointer to \texttt{rocsolver_int}. Array of \( \texttt{batch_count} \) integers on the GPU. If \( \texttt{info}_i = 0 \), successful exit for factorization of \( A_i \). If \( \texttt{info}_i = j > 0 \), \( U_i \) is singular. \( U_i[j,j] \) is the first zero pivot.
- [in] \( \texttt{batch_count} \): \texttt{rocsolver_int}. \( \texttt{batch_count} \geq 0 \). Number of matrices in the batch.

2.12.6.5.7.4 \texttt{rocsolver\_<type>getrf()}

\texttt{rocsolver_status rocsolver_zgetrf}(\texttt{rocsolver\_handle handle}, \texttt{const rocsolver_int m}, \texttt{const rocsolver_int n}, \texttt{rocblas\_double\_complex *A}, \texttt{const rocsolver_int lda}, \texttt{rocsolver_int *ipiv}, \texttt{rocsolver_int *info})

\texttt{rocsolver_status rocsolver_cgetrf}(\texttt{rocsolver\_handle handle}, \texttt{const rocsolver_int m}, \texttt{const rocsolver_int n}, \texttt{rocblas\_float\_complex *A}, \texttt{const rocsolver_int lda}, \texttt{rocsolver_int *ipiv}, \texttt{rocsolver_int *info})

\texttt{rocsolver_status rocsolver_dgetrf}(\texttt{rocsolver\_handle handle}, \texttt{const rocsolver_int m}, \texttt{const rocsolver_int n}, \texttt{double *A}, \texttt{const rocsolver_int lda}, \texttt{rocsolver_int *ipiv}, \texttt{rocsolver_int *info})

\texttt{rocsolver_status rocsolver_sgetrf}(\texttt{rocsolver\_handle handle}, \texttt{const rocsolver_int m}, \texttt{const rocsolver_int n}, \texttt{float *A}, \texttt{const rocsolver_int lda}, \texttt{rocsolver_int *ipiv}, \texttt{rocsolver_int *info})

\texttt{rocsolver\_getrf} computes the \texttt{LU} factorization of a general \( m \)-by-\( n \) matrix \( A \) using partial pivoting with row interchanges.

(This is the right-looking Level 3 BLAS version of the algorithm).

The factorization has the form

\[
A = P \times L \times U
\]

where \( P \) is a permutation matrix, \( L \) is lower triangular with unit diagonal elements (lower trapezoidal if \( m > n \)), and \( U \) is upper triangular (upper trapezoidal if \( m < n \)).

**Parameters**

- [in] \( \texttt{handle} \): \texttt{rocsolver\_handle}.
- [in] \( m \): \texttt{rocsolver\_int}. \( m \geq 0 \). The number of rows of the matrix \( A \).
• [in] n: rocsolver_int. \( n \geq 0 \). The number of columns of the matrix A.

• [inout] A: pointer to type. Array on the GPU of dimension lda*\( n \). On entry, the m-by-\( n \) matrix A to be factored. On exit, the factors L and U from the factorization. The unit diagonal elements of L are not stored.

• [in] lda: rocsolver_int. \( lda \geq m \). Specifies the leading dimension of A.

• [out] ipiv: pointer to rocsolver_int. Array on the GPU of dimension min(m,\( n \)). The vector of pivot indices. Elements of ipiv are 1-based indices. For \( 1 \leq i \leq \min(m, n) \), the row \( i \) of the matrix was interchanged with row \( \text{ipiv}[i] \). Matrix P of the factorization can be derived from ipiv.

• [out] info: pointer to a rocsolver_int on the GPU. If info = 0, successful exit. If info = \( i > 0 \), U is singular. U(\( i,i \)) is the first zero pivot.

2.12.6.5.7.5 rocsolver_<type>getrf_batched()

rocsolver_status rocsolver_zgetrf_batched(
    rocsolver_handle handle,
    const rocsolver_int m,
    const rocsolver_int n, rocblas_double_complex *
    const A[],
    const rocsolver_int lda,
    rocsolver_int *
    ipiv,
    const rocsolver_int strideP,
    rocsolver_int *info,
    const rocsolver_int batch_count)

rocsolver_status rocsolver_cgetrf_batched(
    rocsolver_handle handle,
    const rocsolver_int m,
    const rocsolver_int n, rocblas_float_complex *
    const A[],
    const rocsolver_int lda,
    rocsolver_int *
    ipiv,
    const rocsolver_int strideP,
    rocsolver_int *info,
    const rocsolver_int batch_count)

rocsolver_status rocsolver_dgetrf_batched(
    rocsolver_handle handle,
    const rocsolver_int m,
    const rocsolver_int n, double *
    const A[],
    const rocsolver_int lda,
    rocsolver_int *
    ipiv,
    const rocsolver_int strideP,
    rocsolver_int *info,
    const rocsolver_int batch_count)

rocsolver_status rocsolver_sgetrf_batched(
    rocsolver_handle handle,
    const rocsolver_int m,
    const rocsolver_int n, float *
    const A[],
    const rocsolver_int lda,
    rocsolver_int *
    ipiv,
    const rocsolver_int strideP,
    rocsolver_int *info,
    const rocsolver_int batch_count)

GETRF_BATCHED computes the LU factorization of a batch of general m-by-n matrices using partial pivoting with row interchanges.

(This is the right-looking Level 3 BLAS version of the algorithm).

The factorization of matrix \( A_{i} \) in the batch has the form

\[
A_{i} = P_{i} \times L_{i} \times U_{i}
\]

where \( P_{i} \) is a permutation matrix, \( L_{i} \) is lower triangular with unit diagonal elements (lower trapezoidal if \( m > n \)), and \( U_{i} \) is upper triangular (upper trapezoidal if \( m < n \)).

Parameters

• [in] handle: rocsolver_handle.

• [in] m: rocsolver_int. \( m \geq 0 \). The number of rows of all matrices \( A_{i} \) in the batch.

• [in] n: rocsolver_int. \( n \geq 0 \). The number of columns of all matrices \( A_{i} \) in the batch.

2.12. ROCm Libraries 435
• [inout] A: array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the m-by-n matrices A_i to be factored. On exit, the factors L_i and U_i from the factorizations. The unit diagonal elements of L_i are not stored.

• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_i.

• [out] ipiv: pointer to rocsolver_int. Array on the GPU (the size depends on the value of strideP). Contains the vectors of pivot indices ipiv_i (corresponding to A_i). Dimension of ipiv_i is min(m,n). Elements of ipiv_i are 1-based indices. For each instance A_i in the batch and for 1 <= j <= min(m,n), the row j of the matrix A_i was interchanged with row ipiv_i(j). Matrix P_i of the factorization can be derived from ipiv_i.

• [in] strideP: rocsolver_int. Stride from the start of one vector ipiv_i to the next one ipiv_(i+1). There is no restriction for the value of strideP. Normal use case is strideP >= min(m,n).

• [out] info: pointer to rocsolver_int. Array of batch_count integers on the GPU. If info_i = 0, successful exit for factorization of A_i. If info_i = j > 0, U_i is singular. U_i(j,j) is the first zero pivot.

• [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

### 2.12.6.5.7.6 rocsolver_<type>getrf_strided_batched()

```c
rocsolver_status rocsolver_zgetrf_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_double_complex *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
```

```c
rocsolver_status rocsolver_cgetrf_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, rocblas_float_complex *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
```

```c
rocsolver_status rocsolver_dgetrf_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
```

```c
rocsolver_status rocsolver_sgetrf_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, rocsolver_int *ipiv, const rocsolver_int strideP, rocsolver_int *info, const rocsolver_int batch_count)
```

GETRF_STRIDED_BATCHED computes the LU factorization of a batch of general m-by-n matrices using partial pivoting with row interchanges.

(This is the right-looking Level 3 BLAS version of the algorithm).

The factorization of matrix A_i in the batch has the form
\[
A_i = P_i \cdot L_i \cdot U_i
\]

where \( P_i \) is a permutation matrix, \( L_i \) is lower triangular with unit diagonal elements (lower trapezoidal if \( m > n \)), and \( U_i \) is upper triangular (upper trapezoidal if \( m < n \)).

**Parameters**

- **[in]** handle: rocsolver_handle.
- **[in]** \( m \): rocsolver_int. \( m \geq 0 \). The number of rows of all matrices \( A_i \) in the batch.
- **[in]** \( n \): rocsolver_int. \( n \geq 0 \). The number of columns of all matrices \( A_i \) in the batch.
- **[inout]** \( A \): array on the GPU (the size depends on the value of \( \text{stride}A \)). On entry, contains the \( m \)-by-\( n \) matrices \( A_i \) to be factored. On exit, the factors \( L_i \) and \( U_i \) from the factorization. The unit diagonal elements of \( L_i \) are not stored.
- **[in]** \( \text{lda} \): rocsolver_int. \( \text{lda} \geq m \). Specifies the leading dimension of matrices \( A_i \).
- **[in]** \( \text{stride}A \): rocsolver_int. Stride from the start of one matrix \( A_i \) and the next one \( A_{(i+1)} \). There is no restriction for the value of \( \text{stride}A \). Normal use case is \( \text{stride}A \geq \text{lda} \times n \)
- **[out]** \( \text{ipiv} \): pointer to rocsolver_int. Array on the GPU (the size depends on the value of \( \text{stride}P \)). Contains the vectors of pivots indices \( \text{ipiv}_i \) (corresponding to \( A_i \)). Dimension of \( \text{ipiv}_i \) is \( \min(m,n) \). Elements of \( \text{ipiv}_i \) are 1-based indices. For each instance \( A_i \) in the batch and for \( 1 \leq j \leq \min(m,n) \), the row \( j \) of the matrix \( A_i \) was interchanged with row \( \text{ipiv}_i(j) \). Matrix \( P_i \) of the factorization can be derived from \( \text{ipiv}_i \).
- **[in]** \( \text{stride}P \): rocsolver_int. Stride from the start of one vector \( \text{ipiv}_i \) to the next one \( \text{ipiv}_{(i+1)} \). There is no restriction for the value of \( \text{stride}P \). Normal use case is \( \text{stride}P \geq \min(m,n) \).
- **[out]** info: pointer to rocsolver_int. Array of batch_count integers on the GPU. If \( \text{info}_i = 0 \), successful exit for factorization of \( A_i \). If \( \text{info}_i = j > 0 \), \( U_i \) is singular. \( U_i(j,j) \) is the first zero pivot.
- **[in]** \( \text{batch} \_\text{count} \): rocsolver_int. \( \text{batch} \_\text{count} \geq 0 \). Number of matrices in the batch.

### 2.12.6.5.7.7 rocsolver_<type>geqr2()

**rocsolver_status** rocsolver_dgeqr2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, double *ipiv)

**rocsolver_status** rocsolver_sgeqr2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, float *ipiv)

GEQR2 computes a QR factorization of a general \( m \)-by-\( n \) matrix \( A \).

(This is the unblocked version of the algorithm).

The factorization has the form

\[
A = Q \times \begin{bmatrix} R \\ 0 \end{bmatrix}
\]

where \( R \) is upper triangular (upper trapezoidal if \( m < n \)), and \( Q \) is a \( m \)-by-\( m \) orthogonal matrix represented as the product of Householder matrices

\[
Q = H(1) \times H(2) \times \ldots \times H(k), \quad \text{with} \quad k = \min(m,n)
\]

Each Householder matrix \( H(i) \), for \( i = 1,2,\ldots,k \), is given by
\[ H(i) = I - \text{ipiv}[i-1] \times v(i) \times v(i)' \]

where the first i-1 elements of the Householder vector v(i) are zero, and v(i)[i] = 1.

**Parameters**

- **[in]** \text{handle}: rocsolver\_handle.
- **[in]** \text{m}: rocsolver\_int. m >= 0. The number of rows of the matrix A.
- **[in]** \text{n}: rocsolver\_int. n >= 0. The number of columns of the matrix A.
- **[inout]** \text{A}: pointer to type. Array on the GPU of dimension lda*n. On entry, the m-by-n matrix to be factored. On exit, the elements on and above the diagonal contain the factor R; the elements below the diagonal are the m - i elements of vector v(i) for i = 1,2,...,\text{min}(m,n).
- **[in]** \text{lda}: rocsolver\_int. lda >= m. Specifies the leading dimension of A.
- **[out]** \text{ipiv}: pointer to type. Array on the GPU of dimension \text{min}(m,n). The scalar factors of the Householder matrices H(i).

**2.12.6.5.7.8 rocsolver\_<type\>_geqr2\_batched()**

\[
\text{rocsolver\_status rocsolver\_}\text{dgeqr2\_batched} (\text{rocsolver\_handle handle, const rocsolver\_int m, const rocsolver\_int n, double *const A[], const rocsolver\_int lda, double *ipiv, const rocsolver\_int strideP, const rocsolver\_int batch\_count})
\]

\[
\text{rocsolver\_status rocsolver\_sgeqr2\_batched} (\text{rocsolver\_handle handle, const rocsolver\_int m, const rocsolver\_int n, float *const A[], const rocsolver\_int lda, float *ipiv, const rocsolver\_int strideP, const rocsolver\_int batch\_count})
\]

GEQR2\_BATCHED computes the QR factorization of a batch of general m-by-n matrices. (This is the unblocked version of the algorithm).

The factorization of matrix A\_j in the batch has the form

\[
A_{-j} = Q_{-j} \times \begin{bmatrix} R_{-j} \\ 0 \end{bmatrix}
\]

where R\_j is upper triangular (upper trapezoidal if m < n), and Q\_j is a m-by-m orthogonal matrix represented as the product of Householder matrices

\[
Q_{-j} = H_{-j}(1) \times H_{-j}(2) \times \ldots \times H_{-j}(k), \text{ with } k = \text{min}(m,n)
\]

Each Householder matrices H\_j(i), for j = 1,2,\ldots,\text{batch\_count}, and i = 1,2,\ldots,k, is given by

\[
H_{-j}(i) = I - \text{ipiv}_{-j}[i-1] \times v_{-j}(i) \times v_{-j}(i)'
\]

where the first i-1 elements of Householder vector v\_j(i) are zero, and v\_j(i)[i] = 1.

**Parameters**

- **[in]** \text{handle}: rocsolver\_handle.
- **[in]** \text{m}: rocsolver\_int. m >= 0. The number of rows of all the matrices A\_j in the batch.
- **[in]** \text{n}: rocsolver\_int. n >= 0. The number of columns of all the matrices A\_j in the batch.
• **[inout] A**: Array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the m-by-n matrices A_j to be factored. On exit, the elements on and above the diagonal contain the factor R_j. The elements below the diagonal are the m - i elements of vector v_j(i) for i=1,2,...,min(m,n).

• **[in] lda**: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_j.

• **[out] ipiv**: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors ipiv_j of scalar factors of the Householder matrices H_j(i).

• **[in] strideP**: rocsolver_int. Stride from the start of one vector ipiv_j to the next one ipiv_(j+1). There is no restriction for the value of strideP. Normal use is strideP >= min(m,n).

• **[in] batch_count**: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.6.5.7.9 rocsolver_<type>geqr2_strided_batched()

rocsolver_status rocsolver_dgeqr2_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, double *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

rocsolver_status rocsolver_sgeqr2_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, float *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

GEQR2_STRIDED_BATCHED computes the QR factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix A_j in the batch has the form

\[ A_j = Q_j \times [ R_j ] \]
\[ \hspace{1cm} [ 0 ] \]

where R_j is upper triangular (upper trapezoidal if m < n), and Q_j is a m-by-m orthogonal matrix represented as the product of Householder matrices

\[ Q_j = H_j(1) \times H_j(2) \times \ldots \times H_j(k), \text{ with } k = \min(m,n) \]

Each Householder matrices H_j(i), for j = 1,2,...,batch_count, and i = 1,2,...,k, is given by

\[ H_j(i) = I - ipiv_j[i-1] \times v_j(i) \times v_j(i)' \]

where the first i-1 elements of Householder vector v_j(i) are zero, and v_j(i)[i] = 1.

**Parameters**

• **[in] handle**: rocsolver_handle.

• **[in] m**: rocsolver_int. m >= 0. The number of rows of all the matrices A_j in the batch.

• **[in] n**: rocsolver_int. n >= 0. The number of columns of all the matrices A_j in the batch.
• [inout] A: pointer to type. Array on the GPU (the size depends on the value of strideA). On entry, the m-by-n matrices A_j to be factored. On exit, the elements on and above the diagonal contain the factor R_j. The elements below the diagonal are the m - i elements of vector v_j(i) for i = 1,2,...,min(m,n).

• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_j.

• [in] strideA: rocsolver_int. Stride from the start of one matrix A_j and the next one A_(j+1). There is no restriction for the value of strideA. Normal use case is strideA >= lda*n.

• [out] ipiv: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors ipiv_j of scalar factors of the Householder matrices H_j(i).

• [in] strideP: rocsolver_int. Stride from the start of one vector ipiv_j to the next one ipiv_(j+1). There is no restriction for the value of strideP. Normal use is strideP >= min(m,n).

• [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.6.5.7.10 rocsolver_<type>geqrf()

rocsolver_status rocsolver_dgeqrf(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, double *ipiv)

rocsolver_status rocsolver_sgeqrf(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, float *ipiv)

GEQRF computes a QR factorization of a general m-by-n matrix A.

(This is the blocked version of the algorithm).

The factorization has the form

\[
A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}
\]

where R is upper triangular (upper trapezoidal if m < n), and Q is a m-by-m orthogonal matrix represented as the product of Householder matrices

\[
Q = H(1) \times H(2) \times \ldots \times H(k), \text{ with } k = \min(m,n)
\]

Each Householder matrix H(i), for i = 1,2,...,k, is given by

\[
H(i) = I - \text{ipiv[i-1]} \times v(i) \times v(i)'
\]

where the first i-1 elements of the Householder vector v(i) are zero, and v(i)[i] = 1.

Parameters

• [in] handle: rocsolver_handle.

• [in] m: rocsolver_int. m >= 0. The number of rows of the matrix A.

• [in] n: rocsolver_int. n >= 0. The number of columns of the matrix A.

• [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the m-by-n matrix to be factored. On exit, the elements on and above the diagonal contain the factor R; the elements below the diagonal are the m - i elements of vector v(i) for i = 1,2,...,min(m,n).

• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.

• [out] ipiv: pointer to type. Array on the GPU of dimension min(m,n). The scalar factors of the Householder matrices H(i).
GEQRF_BATCHED computes the QR factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix $A_j$ in the batch has the form

$$A_j = Q_j \times \begin{bmatrix} R_j \\ 0 \end{bmatrix}$$

where $R_j$ is upper triangular (upper trapezoidal if $m < n$), and $Q_j$ is a $m$-by-$m$ orthogonal matrix represented as the product of Householder matrices

$$Q_j = H_j(1) \times H_j(2) \times \ldots \times H_j(k), \text{ with } k = \min(m, n)$$

Each Householder matrices $H_j(i)$, for $j = 1,2,\ldots,\text{batch\_count}$, and $i = 1,2,\ldots,k$, is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] \times v_j(i) \times v_j(i)^t$$

where the first $i-1$ elements of vector Householder vector $v_j(i)$ are zero, and $v_j(i)[i] = 1$.

**Parameters**

- **[in]** handle: rocsolver_handle.
- **[in]** m: rocsolver_int. $m \geq 0$. The number of rows of all the matrices $A_j$ in the batch.
- **[in]** n: rocsolver_int. $n \geq 0$. The number of columns of all the matrices $A_j$ in the batch.
- **[inout]** A: Array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the $m$-by-$n$ matrices $A_j$ to be factored. On exit, the elements on and above the diagonal contain the factor $R_j$. The elements below the diagonal are the $m - i$ elements of vector $v_j(i)$ for $i=1,2,\ldots,\min(m,n)$.
- **[in]** lda: rocsolver_int. lda $\geq m$. Specifies the leading dimension of matrices $A_j$.
- **[out]** ipiv: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors ipiv_j of scalar factors of the Householder matrices $H_j(i)$.
- **[in]** strideP: rocsolver_int. Stride from the start of one vector ipiv_j to the next one ipiv_j(j+1). There is no restriction for the value of strideP. Normal use is strideP $\geq \min(m,n)$.
- **[in]** batch_count: rocsolver_int. batch_count $\geq 0$. Number of matrices in the batch.
2.12.6.5.7.12 rocsolver_<type>geqrf_strided_batched()

rocsolver_status rocsolver_dgeqrf_strided_batched(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, double *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

rocsolver_status rocsolver_sgeqrf_strided_batched(rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, float *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

GEQRF_STRIDED_BATCHED computes the QR factorization of a batch of general m-by-n matrices.  
(This is the blocked version of the algorithm).

The factorization of matrix A_j in the batch has the form

$$A_j = Q_j \times \begin{bmatrix} R_j \\ 0 \end{bmatrix}$$

where R_j is upper triangular (upper trapezoidal if m < n), and Q_j is a m-by-m orthogonal matrix represented as the product of Householder matrices

$$Q_j = H_j(1) \times H_j(2) \times \ldots \times H_j(k), \text{ with } k = \min(m,n)$$

Each Householder matrices H_j(i), for j = 1,2,…,batch_count, and i = 1,2,…,k, is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] \times v_j(i) \times v_j(i)'$$

where the first i-1 elements of vector Householder vector v_j(i) are zero, and v_j(i)[i] = 1.

Parameters

- [in] handle: rocsolver_handle.
- [in] m: rocsolver_int. m >= 0. The number of rows of all the matrices A_j in the batch.
- [in] n: rocsolver_int. n >= 0. The number of columns of all the matrices A_j in the batch.
- [inout] A: pointer to type. Array on the GPU (the size depends on the value of strideA). On entry, the m-by-n matrices A_j to be factored. On exit, the elements on and above the diagonal contain the factor R_j. The elements below the diagonal are the m - i elements of vector v_j(i) for i = 1,2,…,\min(m,n).
- [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_j.
- [in] strideA: rocsolver_int. Stride from the start of one matrix A_j and the next one A_(j+1). There is no restriction for the value of strideA. Normal use case is strideA >= lda*n.
- [out] ipiv: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors ipiv_j of scalar factors of the Householder matrices H_j(i).
- [in] strideP: rocsolver_int. Stride from the start of one vector ipiv_j to the next one ipiv_(j+1). There is no restriction for the value of strideP. Normal use is strideP >= \min(m,n).
- [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.
2.12.6.5.7.13 rocsolver_<type>gelq2()

rocsolver_status rocsolver_dgelq2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, double *ipiv)

rocsolver_status rocsolver_sgelq2 (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, float *ipiv)

GELQ2 computes a LQ factorization of a general m-by-n matrix A.

(This is the unblocked version of the algorithm).

The factorization has the form

\[ A = [ L \ 0 ] \times Q \]

where L is lower triangular (lower trapezoidal if m > n), and Q is a n-by-n orthogonal matrix represented as the product of Householder matrices

\[ Q = H(k) \times H(k-1) \times \ldots \times H(1), \text{ with } k = \min(m,n) \]

Each Householder matrix \( H(i) \), for \( i = 1,2,\ldots,k \), is given by

\[ H(i) = I - ipiv[i-1] \times v(i)' \times v(i) \]

where the first \( i-1 \) elements of the Householder vector \( v(i) \) are zero, and \( v(i)[i] = 1 \).

Parameters

- [in] handle: rocsolver_handle.
- [in] m: rocsolver_int. m >= 0. The number of rows of the matrix A.
- [in] n: rocsolver_int. n >= 0. The number of columns of the matrix A.
- [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the m-by-n matrix to be factored. On exit, the elements on and below the diagonal contain the factor L; the elements above the diagonal are the n - i elements of vector v(i) for i = 1,2,\ldots,min(m,n).
- [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
- [out] ipiv: pointer to type. Array on the GPU of dimension min(m,n). The scalar factors of the Householder matrices H(i).

2.12.6.5.7.14 rocsolver_<type>gelq2_batched()

rocsolver_status rocsolver_dgelq2_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A[], const rocsolver_int lda, double *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

rocsolver_status rocsolver_sgelq2_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A[], const rocsolver_int lda, float *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

GELQ2_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix \( A_j \) in the batch has the form
where \( L_j \) is lower triangular (lower trapezoidal if \( m > n \)), and \( Q_j \) is a \( n \)-by-\( n \) orthogonal matrix represented as the product of Householder matrices

\[
Q_j = H_j(k) \times H_j(k-1) \times \ldots \times H_j(1), \text{ with } k = \min(m,n)
\]

Each Householder matrices \( H_j(i) \), for \( j = 1,2,\ldots,\text{batch\_count} \), and \( i = 1,2,\ldots,k \), is given by

\[
H_j(i) = I - \text{ipiv}_j[i-1] \times v_j(i)' \times v_j(i)
\]

where the first \( i-1 \) elements of Householder vector \( v_j(i) \) are zero, and \( v_j(i)[i] = 1 \).

**Parameters**

- **[in]** handle: rocsolver_handle.
- **[in]** \( m \): rocsolver_int. \( m \geq 0 \). The number of rows of all the matrices \( A_j \) in the batch.
- **[in]** \( n \): rocsolver_int. \( n \geq 0 \). The number of columns of all the matrices \( A_j \) in the batch.
- **[inout]** \( A \): Array of pointers to type. Each pointer points to an array on the GPU of dimension \( \text{lda} \times n \). On entry, the \( m \)-by-\( n \) matrices \( A_j \) to be factored. On exit, the elements on and below the diagonal contain the factor \( L_j \). The elements above the diagonal are the \( n - i \) elements of vector \( v_j(i) \) for \( i = 1,2,\ldots,\min(m,n) \).
- **[in]** \( \text{lda} \): rocsolver_int. \( \text{lda} \geq m \). Specifies the leading dimension of matrices \( A_j \).
- **[out]** ipiv: pointer to type. Array on the GPU (the size depends on the value of \( \text{strideP} \)). Contains the vectors \( \text{ipiv}_j \) of scalar factors of the Householder matrices \( H_j(i) \).
- **[in]** \( \text{strideP} \): rocsolver_int. Stride from the start of one vector \( \text{ipiv}_j \) to the next one \( \text{ipiv}_j(j+1) \). There is no restriction for the value of \( \text{strideP} \). Normal use is \( \text{strideP} \geq \min(m,n) \).
- **[in]** \( \text{batch\_count} \): rocsolver_int. \( \text{batch\_count} \geq 0 \). Number of matrices in the batch.

### 2.12.6.5.7.15 rocsholver_<type>_gelq2_strided_batched()

**rocsholver_status** rocsholver_dgelq2_strided_batched

\( \text{rocsholver\_status rocsholver\_dgelq2\_strided\_batched(rocsholver\_handle \ handle, const \ rocsholver\_int \ m, const \ rocsholver\_int \ n, double \ *A, const \ rocsholver\_int \ lda, const \ rocsholver\_int \ strideA, double \ *\text{ipiv}, const \ rocsholver\_int \ strideP, const \ rocsholver\_int \ \text{batch\_count})} \)

**rocsholver_status** rocsholver_sgelq2_strided_batched

\( \text{rocsholver\_status rocsholver\_sgelq2\_strided\_batched(rocsholver\_handle \ handle, const \ rocsholver\_int \ m, const \ rocsholver\_int \ n, float \ *A, const \ rocsholver\_int \ lda, const \ rocsholver\_int \ strideA, float \ *\text{ipiv}, const \ rocsholver\_int \ strideP, const \ rocsholver\_int \ \text{batch\_count})} \)

GELQ2\_STRIDED\_BATCHED computes the LQ factorization of a batch of general \( m \)-by-\( n \) matrices.

(This is the unblocked version of the algorithm).

The factorization of matrix \( A_j \) in the batch has the form

\[
A_j = [ L_j \ 0 ] \times Q_j
\]
where $L_j$ is lower triangular (lower trapezoidal if $m > n$), and $Q_j$ is a $n$-by-$n$ orthogonal matrix represented as the product of Householder matrices

$$Q_j = H_j(k) \ast H_j(k-1) \ast \ldots \ast H_j(1), \text{ with } k = \min(m, n)$$

Each Householder matrices $H_j(i)$, for $j = 1,2,\ldots,\text{batch\_count}$, and $i = 1,2,\ldots,k$, is given by

$$H_j(i) = I - \text{ipiv}_j[i-1] \ast v_j(i)' \ast v_j(i)$$

where the first $i-1$ elements of vector Householder vector $v_j(i)$ are zero, and $v_j(i)[i] = 1$.

Parameters

- [in] handle: rocsolver\_handle.
- [in] m: rocsolver\_int. $m > 0$. The number of rows of all the matrices $A_j$ in the batch.
- [in] n: rocsolver\_int. $n > 0$. The number of columns of all the matrices $A_j$ in the batch.
- [inout] $A$: pointer to type. Array on the GPU (the size depends on the value of strideA). On entry, the $m$-by-$n$ matrices $A_j$ to be factored. On exit, the elements on and below the diagonal contain the factor $L_j$. The elements above the diagonal are the $n - i$ elements of vector $v_j(i)$ for $i = 1,2,\ldots,\min(m,n)$.
- [in] lda: rocsolver\_int. $lda \geq m$. Specifies the leading dimension of matrices $A_j$.
- [in] strideA: rocsolver\_int. Stride from the start of one matrix $A_j$ and the next one $A_{j+1}$. There is no restriction for the value of strideA. Normal use case is $strideA \geq lda \ast n$.
- [out] ipiv: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors $\text{ipiv}_j$ of scalar factors of the Householder matrices $H_j(i)$.
- [in] strideP: rocsolver\_int. Stride from the start of one vector $\text{ipiv}_j$ to the next one $\text{ipiv}_{j+1}$. There is no restriction for the value of strideP. Normal use is $strideP \geq \min(m,n)$.
- [in] batch\_count: rocsolver\_int. batch\_count $\geq 0$. Number of matrices in the batch.

2.12.5.7.16 rocsolver\_<type>gelqf()
where the first i-1 elements of the Householder vector v(i) are zero, and v(i)[i] = 1.

Parameters

- [in] handle: rocsolver_handle.
- [in] m: rocsolver_int. m >= 0. The number of rows of the matrix A.
- [in] n: rocsolver_int. n >= 0. The number of columns of the matrix A.
- [inout] A: pointer to type. Array on the GPU of dimension lda*n. On entry, the m-by-n matrix to be factored. On exit, the elements on and below the diagonal contain the factor L; the elements above the diagonal are the n - i elements of vector v(i) for i = 1,2,…,min(m,n).
- [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of A.
- [out] ipiv: pointer to type. Array on the GPU of dimension min(m,n). The scalar factors of the Householder matrices H(i).

2.12.6.5.7.17 rocsolver_<type>gelqf_batched()

rocsolver_status rocsolver_dgelqf_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *const A[], const rocsolver_int lda, double *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

GELQF_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix A_j in the batch has the form

\[ A_j = [ L_j \ 0 ] \times Q_j \]

where L_j is lower triangular (lower trapezoidal if m > n), and Q_j is a n-by-n orthogonal matrix represented as the product of Householder matrices

\[ Q_j = H_j(k) \times H_j(k-1) \times \ldots \times H_j(1), \text{ with } k = \min(m,n) \]

Each Householder matrices H_j(i), for j = 1,2,…,batch_count, and i = 1,2,…,k, is given by

\[ H_j(i) = I - \text{ipiv}_j[i-1] \times v_j(i)' \times v_j(i) \]

where the first i-1 elements of Householder vector v_j(i) are zero, and v_j(i)[i] = 1.

Parameters

- [in] handle: rocsolver_handle.
- [in] m: rocsolver_int. m >= 0. The number of rows of all the matrices A_j in the batch.
- [in] n: rocsolver_int. n >= 0. The number of columns of all the matrices A_j in the batch.
- [inout] A: Array of pointers to type. Each pointer points to an array on the GPU of dimension lda*n. On entry, the m-by-n matrices A_j to be factored. On exit, the elements on and below the diagonal contain the factor L_j. The elements above the diagonal are the n - i elements of vector v_j(i) for i=1,2,…,min(m,n).
• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_j.
• [out] ipiv: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors ipiv_j of scalar factors of the Householder matrices H_j(i).
• [in] strideP: rocsolver_int. Stride from the start of one vector ipiv_j to the next one ipiv_(j+1). There is no restriction for the value of strideP. Normal use is strideP >= min(m,n).
• [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

2.12.5.7.18 rocsolver_<type>gelqf_strided_batched()

rocsolver_status rocsolver_dgelqf_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, double *A, const rocsolver_int lda, const rocsolver_int strideA, double *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

rocsolver_status rocsolver_sgelqf_strided_batched (rocsolver_handle handle, const rocsolver_int m, const rocsolver_int n, float *A, const rocsolver_int lda, const rocsolver_int strideA, float *ipiv, const rocsolver_int strideP, const rocsolver_int batch_count)

GELQF_STRIDED_BATCHED computes the LQ factorization of a batch of general m-by-n matrices.

(This is the blocked version of the algorithm).

The factorization of matrix A_j in the batch has the form

\[
A_j = [ L_j \ 0 ] \times Q_j
\]

where L_j is lower triangular (lower trapezoidal if m > n), and Q_j is a n-by-n orthogonal matrix represented as the product of Householder matrices

\[
Q_j = H_j(k) \times H_j(k-1) \times \ldots \times H_j(1), \text{ with } k = \min(m, n)
\]

Each Householder matrices H_j(i), for j = 1,2,..,batch_count, and i = 1,2,..,k, is given by

\[
H_j(i) = I - ipiv_j[i-1] \times v_j(i)' \times v_j(i)
\]

where the first i-1 elements of vector Householder vector v_j(i) are zero, and v_j(i)[i] = 1.

Parameters
• [in] handle: rocsolver_handle.
• [in] m: rocsolver_int. m >= 0. The number of rows of all the matrices A_j in the batch.
• [in] n: rocsolver_int. n >= 0. The number of columns of all the matrices A_j in the batch.
• [inout] A: pointer to type. Array on the GPU (the size depends on the value of strideA). On entry, the m-by-n matrices A_j to be factored. On exit, the elements on and below the diagonal contain the factor L_j. The elements above the diagonal are the n - i elements of vector v_j(i) for i = 1,2,..,min(m,n).
• [in] lda: rocsolver_int. lda >= m. Specifies the leading dimension of matrices A_j.
• [in] strideA: rocsolver_int. Stride from the start of one matrix A_j and the next one A_(j+1). There is no restriction for the value of strideA. Normal use case is strideA >= lda*n.

• [out] ipiv: pointer to type. Array on the GPU (the size depends on the value of strideP). Contains the vectors ipiv_j of scalar factors of the Householder matrices H_j(i).

• [in] strideP: rocsolver_int. Stride from the start of one vector ipiv_j to the next one ipiv_(j+1). There is no restriction for the value of strideP. Normal use is strideP >= min(m,n).

• [in] batch_count: rocsolver_int. batch_count >= 0. Number of matrices in the batch.

### 2.12.6.5.8 General systems solvers

#### 2.12.6.5.8.1 rocsolver_<type>getrs()

```c
rocsolver_status rocsolver_zgetrs (rocsolver_handle handle, const rocsolver_operation trans, const rocsolver_int n, const rocsolver_int nrhs, rocblas_double_complex *A, const rocsolver_int lda, const rocsolver_int *ipiv, rocblas_double_complex *B, const rocsolver_int ldb)
```

```c
rocsolver_status rocsolver_cgetrs (rocsolver_handle handle, const rocsolver_operation trans, const rocsolver_int n, const rocsolver_int nrhs, rocblas_float_complex *A, const rocsolver_int lda, const rocsolver_int *ipiv, rocblas_float_complex *B, const rocsolver_int ldb)
```

```c
rocsolver_status rocsolver_dgetrs (rocsolver_handle handle, const rocsolver_operation trans, const rocsolver_int n, const rocsolver_int nrhs, double *A, const rocsolver_int lda, const rocsolver_int *ipiv, double *B, const rocsolver_int ldb)
```

```c
rocsolver_status rocsolver_sgetrs (rocsolver_handle handle, const rocsolver_operation trans, const rocsolver_int n, const rocsolver_int nrhs, float *A, const rocsolver_int lda, const rocsolver_int *ipiv, float *B, const rocsolver_int ldb)
```

GETRS solves a system of n linear equations on n variables using the LU factorization computed by GETRF.

It solves one of the following systems:

\[
\begin{align*}
A \times X &= B \text{ (no transpose), } \\
A^T \times X &= B \text{ (transpose), or } \\
A^{\ast} \times X &= B \text{ (conjugate transpose)}
\end{align*}
\]

depending on the value of trans.

**Parameters**

• [in] handle: rocsolver_handle.

• [in] trans: rocsolver_operation. Specifies the form of the system of equations.

• [in] n: rocsolver_int. n >= 0. The order of the system, i.e. the number of columns and rows of A.

• [in] nrhs: rocsolver_int. nrhs >= 0. The number of right hand sides, i.e., the number of columns of the matrix B.

• [in] A: pointer to type. Array on the GPU of dimension lda*n. The factors L and U of the factorization \( A = P^L \times U \) returned by GETRF.

• [in] lda: rocsolver_int. lda >= n. The leading dimension of A.
GETRF solves a batch of systems of \( n \) linear equations on \( n \) variables using the LU factorization computed by GETRF_BATCHED.

For each instance \( j \) in the batch, it solves one of the following systems:

\[
A_{\cdot j} \cdot X_{\cdot j} = B_{\cdot j} \quad \text{(no transpose)}, \\
A_{\cdot j}' \cdot X_{\cdot j} = B_{\cdot j} \quad \text{(transpose), or} \\
A_{\cdot j}^* \cdot X_{\cdot j} = B_{\cdot j} \quad \text{(conjugate transpose)}
\]

depending on the value of \( \text{trans} \).

**Parameters**

- [in] \( \text{handle} \): rocsolver_handle.
- [in] \( \text{trans} \): rocsolver_operation. Specifies the form of the system of equations of each instance in the batch.
- [in] \( n \): rocsolver_int. \( n \geq 0 \). The order of the system, i.e. the number of columns and rows of all \( A_{\cdot j} \) matrices.

---

2.12.6.5.8.2 \textit{rocsolver_<type>getrs\_batched()}

\textit{rocsolver\_status rocsolver\_zgetrs\_batched} (rocblas\_handle handle, const rocblas\_operation \( \text{trans} \), const rocblas\_int \( n \), const rocblas\_int \( \text{nrhs} \), rocblas\_double\_complex *\( \text{const A}[] \), const rocblas\_int \( \text{lda} \), const rocblas\_int *\( \text{ipiv} \), const rocblas\_int \( \text{strideP} \), rocblas\_double\_complex *\( \text{const B}[] \), const rocblas\_int \( \text{ldb} \), const rocblas\_int \( \text{batch\_count} \))

\textit{rocsolver\_status rocsolver\_cgetrs\_batched} (rocblas\_handle handle, const rocblas\_operation \( \text{trans} \), const rocblas\_int \( n \), const rocblas\_int \( \text{nrhs} \), rocblas\_float\_complex *\( \text{const A}[] \), const rocblas\_int \( \text{lda} \), const rocblas\_int *\( \text{ipiv} \), const rocblas\_int \( \text{strideP} \), rocblas\_float\_complex *\( \text{const B}[] \), const rocblas\_int \( \text{ldb} \), const rocblas\_int \( \text{batch\_count} \))

\textit{rocsolver\_status rocsolver\_dgetrs\_batched} (rocblas\_handle handle, const rocblas\_operation \( \text{trans} \), const rocblas\_int \( n \), const rocblas\_int \( \text{nrhs} \), double *\( \text{const A}[] \), const rocblas\_int \( \text{lda} \), const rocblas\_int *\( \text{ipiv} \), const rocblas\_int \( \text{strideP} \), double *\( \text{const B}[] \), const rocblas\_int \( \text{ldb} \), const rocblas\_int \( \text{batch\_count} \))

\textit{rocsolver\_status rocsolver\_sgetrs\_batched} (rocblas\_handle handle, const rocblas\_operation \( \text{trans} \), const rocblas\_int \( n \), const rocblas\_int \( \text{nrhs} \), float *\( \text{const A}[] \), const rocblas\_int \( \text{lda} \), const rocblas\_int *\( \text{ipiv} \), const rocblas\_int \( \text{strideP} \), float *\( \text{const B}[] \), const rocblas\_int \( \text{ldb} \), const rocblas\_int \( \text{batch\_count} \))
• [in] \( \text{nrhs} \): \text{roc solver int}. \( \text{nrhs} \geq 0 \). The number of right hand sides, i.e., the number of columns of all the matrices \( B_j \).
• [in] \( A \): Array of pointers to type. Each pointer points to an array on the GPU of dimension \( \text{lda} \times n \). The factors \( L_j \) and \( U_j \) of the factorization \( A_j = P_j \text{L}_j \text{U}_j \) returned by \text{GETRF\_BATCHED}.
• [in] \( \text{lda} \): \text{roc solver int}. \( \text{lda} \geq n \). The leading dimension of matrices \( A_j \).
• [in] \( \text{ipiv} \): pointer to \text{roc solver int}. Array on the GPU (the size depends on the value of \text{strideP}). Contains the vectors \( \text{ipiv}_j \) of pivot indices returned by \text{GETRF\_BATCHED}.
• [inout] \( B \): Array of pointers to type. Each pointer points to an array on the GPU of dimension \( \text{ldb} \times \text{nrhs} \). On entry, the right hand side matrices \( B_j \). On exit, the solution matrix \( X_j \) of each system in the batch.
• [in] \( \text{ldb} \): \text{roc solver int}. \( \text{ldb} \geq n \). The leading dimension of matrices \( B_j \).
• [in] \( \text{batch\_count} \): \text{roc solver int}. \( \text{batch\_count} \geq 0 \). Number of instances (systems) in the batch.

2.12.6.5.8.3 \text{roc solver\_<type>\text{getrs\_strided\_batched}(\text{\textbackslash})}

\text{roc solver status \text{roc solver\_zgetrs\_strided\_batched}(rocblas\_handle \ handle, \ const rocblas\_operation \ trans, \ const rocblas\_int \ n, \ const rocblas\_int \ nrhs, \ rocblas\_double\_complex \ *A, \ const rocblas\_int \ lda, \ const rocblas\_int \ strideA, \ const rocblas\_int \ *ipiv, \ const rocblas\_int \ strideP, \ rocblas\_double\_complex \ *B, \ const rocblas\_int \ ldb, \ const rocblas\_int \ strideB, \ const rocblas\_int \ batch\_count)}

\text{roc solver status \text{roc solver\_cgetrs\_strided\_batched}(rocblas\_handle \ handle, \ const rocblas\_operation \ trans, \ const rocblas\_int \ n, \ const rocblas\_int \ nrhs, \ rocblas\_float\_complex \ *A, \ const rocblas\_int \ lda, \ const rocblas\_int \ strideA, \ const rocblas\_int \ *ipiv, \ const rocblas\_int \ strideP, \ rocblas\_float\_complex \ *B, \ const rocblas\_int \ ldb, \ const rocblas\_int \ strideB, \ const rocblas\_int \ batch\_count)}

\text{roc solver status \text{roc solver\_dgetrs\_strided\_batched}(rocblas\_handle \ handle, \ const rocblas\_operation \ trans, \ const rocblas\_int \ n, \ const rocblas\_int \ nrhs, \ double \ *A, \ const rocblas\_int \ lda, \ const rocblas\_int \ strideA, \ const rocblas\_int \ *ipiv, \ const rocblas\_int \ strideP, \ double \ *B, \ const rocblas\_int \ ldb, \ const rocblas\_int \ strideB, \ const rocblas\_int \ batch\_count)}
rocSOLVER auxiliary functions are aliases of rocBLAS auxiliary functions. See rocBLAS auxiliary functions here.
2.12.6.5.9 rocSOLVER handle auxiliaries

2.12.6.5.9.1 rocsolver_create_handle()

rocsolver_status rocsolver_create_handle(rocsolver_handle *handle)
Create rocSOLVER handle.

2.12.6.5.9.2 rocsolver_destroy_handle()

rocsolver_status rocsolver_destroy_handle(rocsolver_handle handle)
Destroy rocSOLVER handle.

2.12.6.5.9.3 rocsolver_add_stream()

rocsolver_status rocsolver_add_stream(rocsolver_handle handle, hipStream_t stream)
Add stream to handle.

2.12.6.5.9.4 rocsolver_set_stream()

rocsolver_status rocsolver_set_stream(rocsolver_handle handle, hipStream_t stream)
Remove any streams from handle, and add one.

2.12.6.5.9.5 rocsolver_get_stream()

rocsolver_status rocsolver_get_stream(rocsolver_handle handle, hipStream_t *stream)
Get stream [0] from handle.

2.12.6.5.10 Other auxiliaries

2.12.6.5.10.1 rocsolver_set_vector()

rocsolver_status rocsolver_set_vector(rocsolver_int n, rocsolver_int elem_size, const void *x, rocsolver_int incx, void *y, rocsolver_int incy)
Copy vector from host to device.

2.12.6.5.10.2 rocsolver_get_vector()

rocsolver_status rocsolver_get_vector(rocsolver_int n, rocsolver_int elem_size, const void *x, rocsolver_int incx, void *y, rocsolver_int incy)
Copy vector from device to host.
2.12.6.5.10.3 rocsolver_set_matrix()

rocsolver_status rocsolver_set_matrix(rocsolver_int rows, rocsolver_int cols, rocsolver_int elem_size, const void *a, rocsolver_int lda, void *b, rocsolver_int ldb)

Copy matrix from host to device.

2.12.6.5.10.4 rocsolver_get_matrix()

rocsolver_status rocsolver_get_matrix(rocsolver_int rows, rocsolver_int cols, rocsolver_int elem_size, const void *a, rocsolver_int lda, void *b, rocsolver_int ldb)

Copy matrix from device to host.

2.12.7 hipSPARSE

hipSPARSE is a SPARSE marshalling library, with multiple supported backends. It sits between the application and a ‘worker’ SPARSE library, marshalling inputs into the backend library and marshalling results back to the application.

hipSPARSE exports an interface that does not require the client to change, regardless of the chosen backend. Currently, hipSPARSE supports rocSPARSE and cuSPARSE as backends. Refer the hipSparse wiki page hipsparsewiki

2.12.7.1 Installing pre-built packages

Download pre-built packages either from ROCm’s package servers or by clicking the github releases tab and manually downloading, which could be newer. Release notes are available for each release on the releases tab.

• sudo apt update && sudo apt install hipsparse

2.12.7.2 Quickstart build

Bash helper build script (Ubuntu only)

The root of this repository has a helper bash script install.sh to build and install hipSPARSE on Ubuntu with a single command. It does not take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it’s a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access, so it may prompt you for a password.

```
./install -h -- shows help
./install -id -- build library, build dependencies and install (-d flag only needs to be passed once on a system)
```

2.12.7.3 Manual build (all supported platforms)

If you use a distro other than Ubuntu, or would like more control over the build process, the hipsbuild build wiki has helpful information on how to configure cmake and manually build.
2.12.7.4 Functions

A list of export from hipSPARSE can be found on the wiki.

2.12.7.5 hipSPARSE interface examples

The hipSPARSE interface is compatible with rocSPARSE and cuSPARSE-v2 APIs. Porting a CUDA application which originally calls the cuSPARSE API to an application calling hipSPARSE API should be relatively straightforward. For example, the hipSPARSE SCSRMV interface is

```c
hipsparseStatus_t
hipsparseScsrmv(hipsparseHandle_t handle,
               hipsparseOperation_t transA,
               int m, int n, int nnz, const float *alpha,
               const hipsparseMatDescr_t descrA,
               const float *csrValA,
               const int *csrRowPtrA, const int *csrColIndA,
               const float *x, const float *beta,
               float *y);
```

hipSPARSE assumes matrix A and vectors x, y are allocated in GPU memory space filled with data. Users are responsible for copying data from/to the host and device memory.

2.12.8 rocALUTION

2.12.8.1 Introduction

2.12.8.2 Overview

rocALUTION is a sparse linear algebra library with focus on exploring fine-grained parallelism, targeting modern processors and accelerators including multi/many-core CPU and GPU platforms. The main goal of this package is to provide a portable library for iterative sparse methods on state of the art hardware. rocALUTION can be seen as middle-ware between different parallel backends and application specific packages.

The major features and characteristics of the library are

- **Various backends**
  - Host - fallback backend, designed for CPUs
  - GPU/HIP - accelerator backend, designed for HIP capable AMD GPUs
  - OpenMP - designed for multi-core CPUs
  - MPI - designed for multi-node and multi-GPU configurations

- **Easy to use** The syntax and structure of the library provide easy learning curves. With the help of the examples, anyone can try out the library - no knowledge in HIP, OpenMP or MPI programming required.

- **No special hardware requirements** There are no hardware requirements to install and run rocALUTION. If a GPU device and HIP is available, the library will use them.

- **Variety of iterative solvers**
  - Fixed-Point iteration - Jacobi, Gauss-Seidel, Symmetric-Gauss Seidel, SOR and SSOR
Krylov subspace methods - CR, CG, BiCGStab, BiCGStab(l), GMRES, IDR, QMRCGSTAB, Flexible CG/GMRES

- Mixed-precision defect-correction scheme
- Chebyshev iteration
- Multiple MultiGrid schemes, geometric and algebraic

- **Various preconditioners**
  - Matrix splitting - Jacobi, (Multi-colored) Gauss-Seidel, Symmetric Gauss-Seidel, SOR, SSOR
  - Factorization - ILU(0), ILU(p) (based on levels), ILU(p,q) (power(q)-pattern method), Multi-Elimination ILU (nested/recusive), ILUT (based on threshold) and IC(0)
  - Approximate Inverse - Chebyshev matrix-valued polynomial, SPAI, FSAI and TNS
  - Diagonal-based preconditioner for Saddle-point problems
  - Block-type of sub-preconditioners/solvers
  - Additive Schwarz and Restricted Additive Schwarz
  - Variable type preconditioners

- **Generic and robust design** rocALUTION is based on a generic and robust design allowing expansion in the direction of new solvers and preconditioners and support for various hardware types. Furthermore, the design of the library allows the use of all solvers as preconditioners in other solvers. For example you can easily define a CG solver with a Multi-Elimination preconditioner, where the last-block is preconditioned with another Chebyshev iteration method which is preconditioned with a multi-colored Symmetric Gauss-Seidel scheme.

- **Portable code and results** All code based on rocALUTION is portable and independent of HIP or OpenMP. The code will compile and run everywhere. All solvers and preconditioners are based on a single source code, which delivers portable results across all supported backends (variations are possible due to different rounding modes on the hardware). The only difference which you can see for a hardware change is the performance variation.

- **Support for several sparse matrix formats** Compressed Sparse Row (CSR), Modified Compressed Sparse Row (MCSR), Dense (DENSE), Coordinate (COO), ELL, Diagonal (DIA), Hybrid format of ELL and COO (HYB).

The code is open-source under MIT license and hosted on here: https://github.com/ROCmSoftwarePlatform/rocALUTION

2.12.8.3 Building and Installing

2.12.8.4 Installing from AMD ROCm repositories

rocALUTION can be installed from AMD ROCm repository. The repository hosts the single-node, accelerator enabled version of the library. If a different setup is required, e.g. multi-node support, rocALUTION need to be built from source, see Building from GitHub repository.

For detailed instructions on how to set up ROCm on different platforms, see the AMD ROCm Platform Installation Guide for Linux.

rocALUTION has the following run-time dependencies

- **AMD ROCm** 3.0 or later (optional, for HIP support)
- **rocSPARSE** (optional, for HIP support)
• rocBLAS (optional, for HIP support)
• OpenMP (optional, for OpenMP support)
• MPI (optional, for multi-node / multi-GPU support)

2.12.8.5 Building rocALUTION from Github repository

To build rocALUTION from source, the following compile-time and run-time dependencies must be met:

• git
• `CMake <https://cmake.org/>`_ 3.5 or later
• AMD ROCm 3.0 or later (optional, for HIP support)
• rocSPARSE (optional, for HIP support)
• rocBLAS (optional, for HIP support)
• rocPRIM (optional, for HIP support)
• OpenMP (optional, for OpenMP support)
• MPI (optional, for multi-node / multi-GPU support)
• googletest (optional, for clients)

2.12.8.6 Download rocALUTION

The rocALUTION source code is available at the rocALUTION github page. Download the master branch using:

```bash
git clone -b master https://github.com/ROCmSoftwarePlatform/rocALUTION.git
cd rocALUTION
```

Below are steps to build different packages of the library, including dependencies and clients. It is recommended to install rocALUTION using the `install.sh` script.

2.12.8.7 Using `install.sh` to build dependencies + library

The following table lists common uses of `install.sh` to build dependencies + library. Accelerator support via HIP and OpenMP will be enabled by default, whereas MPI is disabled.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./install.sh -h</td>
<td>Print help information.</td>
</tr>
<tr>
<td>./install.sh -d</td>
<td>Build dependencies and library in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.</td>
</tr>
<tr>
<td>./install.sh</td>
<td>Build library in your local directory. It is assumed dependencies are available.</td>
</tr>
<tr>
<td>./install.sh -i</td>
<td>Build library, then build and install rocALUTION package in /opt/rocm/rocalution. You will be prompted for sudo access. This will install for all users.</td>
</tr>
<tr>
<td>./install.sh --host</td>
<td>Build library in your local directory without HIP support. It is assumed dependencies are available.</td>
</tr>
<tr>
<td>./install.sh --mpi</td>
<td>Build library in your local directory with HIP and MPI support. It is assumed dependencies are available.</td>
</tr>
</tbody>
</table>

2.12.8.8 Using `install.sh` to build dependencies + library + client

The client contains example code, unit tests and benchmarks. Common uses of `install.sh` to build them are listed in the table below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./install.sh -h</td>
<td>Print help information.</td>
</tr>
<tr>
<td>./install.sh -dc</td>
<td>Build dependencies, library and client in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.</td>
</tr>
<tr>
<td>./install.sh -c</td>
<td>Build library and client in your local directory. It is assumed dependencies are available.</td>
</tr>
<tr>
<td>./install.sh -idc</td>
<td>Build library, dependencies and client, then build and install rocALUTION package in /opt/rocm/rocalution. You will be prompted for sudo access. This will install for all users.</td>
</tr>
<tr>
<td>./install.sh -ic</td>
<td>Build library and client, then build and install rocALUTION package in /opt/rocm/rocalution. You will be prompted for sudo access. This will install for all users.</td>
</tr>
</tbody>
</table>
2.12.8.9 Using individual commands to build rocALUTION

CMake 3.5 or later is required in order to build rocALUTION without the use of install.sh.

rocALUTION can be built with cmake using the following commands:

```bash
# Create and change to build directory
mkdir -p build/release ; cd build/release

# Default install path is /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to adjust it
# Configure rocALUTION
# Build options:
# SUPPORT_HIP - build rocALUTION with HIP support (ON)
# SUPPORT_OMP - build rocALUTION with OpenMP support (ON)
# SUPPORT_MPI - build rocALUTION with MPI (multi-node) support (OFF)
# BUILD_SHARED - build rocALUTION as shared library (ON, recommended)
# BUILD_EXAMPLES - build rocALUTION examples (ON)

cmake ../.. -DSUPPORT_HIP=ON \
  -DSUPPORT_MPI=OFF \
  -DSUPPORT_OMP=ON

# Compile rocALUTION library
make -j$(nproc)

# Install rocALUTION to /opt/rocm
sudo make install
```

GoogleTest is required in order to build rocALUTION client.

rocALUTION with dependencies and client can be built using the following commands:

```bash
# Install googletest
mkdir -p build/release/deps ; cd build/release/deps
cmake ../../../deps
sudo make -j$(nproc) install

# Change to build directory
cd..

# Default install path is /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to adjust it

cmake ../.. -DBUILD_CLIENTS_TESTS=ON \
  -DBUILD_CLIENTS_SAMPLES=ON

# Compile rocALUTION library
make -j$(nproc)

# Install rocALUTION to /opt/rocm
sudo make install
```

The compilation process produces a shared library file `librocalution.so` and `librocalution_hip.so` if HIP support is enabled. Ensure that the library objects can be found in your library path. If you do not copy the library to a specific location you can add the path under Linux in the `LD_LIBRARY_PATH` variable.

```bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path_to_rocalution
```
2.12.8.10 Common build problems

1. **Issue:** HIP (/opt/rocm/hip) was built using hcc 1.0.xxx-xxx-xxx-xxx, but you are using /opt/rocm/bin/hcc with version 1.0.yyy-yyyy-yyyy-yyyy from hipcc (version mismatch). Please rebuild HIP including cmake or update HCC_HOME variable.

   **Solution:** Download HIP from github and use hcc to build from source and then use the built HIP instead of /opt/rocm/hip.

2. **Issue:** HCC RUNTIME ERROR: Failed to find compatible kernel

   **Solution:** export HCC_AMDGPU_TARGET=gfx803,gfx900,gfx906,gfx908

3. **Issue:** Could not find a package configuration file provided by “ROCM” with any of the following names:
   ROCMConfig.cmake |br| rocm-config.cmake

   **Solution:** Install ROCm cmake modules either from source or from AMD ROCm repository

4. **Issue:** Could not find a package configuration file provided by “ROCSPARSE” with any of the following names:
   ROCSPARSE.cmake |br| rocsparse-config.cmake

   **Solution:** Install rocSPARSE either from source or from ‘AMD ROCm repository’<https://rocm.github.io/ROCmInstall.html#installing-from-amd-rocm-repositories>

5. **Issue:** Could not find a package configuration file provided by “ROCBLAS” with any of the following names:
   ROCBLAS.cmake |br| rocblas-config.cmake

   **Solution:** Install rocBLAS either from source or from ‘AMD ROCm repository’<https://rocm.github.io/ROCmInstall.html#installing-from-amd-rocm-repositories>

2.12.8.11 Simple Test

You can test the installation by running a CG solver on a Laplace matrix. After compiling the library you can perform the CG solver test by executing

```
cd rocALUTION/build/release/examples
gzip -d gr_30_30.mtx.gz
./clients/staging/cg gr_30_30.mtx
```

For more information regarding rocALUTION library and corresponding API documentation, refer rocALUTION

2.12.8.12 API

This section provides details of the library API
template<typename DataType>
void rocalution::allocate_host (int size, DataType **ptr)
Allocate buffer on the host.
allocate_host allocates a buffer on the host.

Parameters

• [in] size: number of elements the buffer need to be allocated for
• [out] ptr: pointer to the position in memory where the buffer should be allocated, it is expected that *ptr == NULL

Template Parameters

• DataType: can be char, int, unsigned int, float, double, std::complex<float> or std::complex<double>.

template<typename DataType>
void rocalution::free_host (DataType **ptr)
Free buffer on the host.
free_host deallocates a buffer on the host. *ptr will be set to NULL after successful deallocation.

Parameters

• [inout] ptr: pointer to the position in memory where the buffer should be deallocated, it is expected that *ptr != NULL

Template Parameters

• DataType: can be char, int, unsigned int, float, double, std::complex<float> or std::complex<double>.

template<typename DataType>
void rocalution::set_to_zero_host (int size, DataType *ptr)
Set a host buffer to zero.
set_to_zero_host sets a host buffer to zero.

Parameters

• [in] size: number of elements
• [inout] ptr: pointer to the host buffer

Template Parameters

• DataType: can be char, int, unsigned int, float, double, std::complex<float> or std::complex<double>.

double rocalution::rocalution_time (void)
Return current time in microseconds.
2.12.8.12.2 Backend Manager

int rocalution::init_rocalution (int rank = -1, int dev_per_node = 1)
Initialize rocALUTION platform.

init_rocalution defines a backend descriptor with information about the hardware and its specifications. All objects created after that contain a copy of this descriptor. If the specifications of the global descriptor are changed (e.g. set different number of threads) and new objects are created, only the new objects will use the new configurations.

For control, the library provides the following functions

• set_device_rocalution() is a unified function to select a specific device. If you have compiled the library with a backend and for this backend there are several available devices, you can use this function to select a particular one. This function has to be called before init_rocalution().

• set_omp_threads_rocalution() sets the number of OpenMP threads. This function has to be called after init_rocalution().

Example

```
#include <rocalution.hpp>
using namespace rocalution;

int main(int argc, char* argv[])
{
  init_rocalution();

  // ...

  stop_rocalution();

  return 0;
}
```

Parameters

• [in] rank: specifies MPI rank when multi-node environment
• [in] dev_per_node: number of accelerator devices per node, when in multi-GPU environment

int rocalution::stop_rocalution (void)
Shutdown rocALUTION platform.

stop_rocalution shuts down the rocALUTION platform.

void rocalution::set_device_rocalution (int dev)
Set the accelerator device.

set_device_rocalution lets the user select the accelerator device that is supposed to be used for the computation.

Parameters

• [in] dev: accelerator device ID for computation

void rocalution::set_omp_threads_rocalution (int nthreads)
Set number of OpenMP threads.
The number of threads which rocALUTION will use can be set with `set_omp_threads_rocalution` or by the global OpenMP environment variable (for Unix-like OS this is `OMP_NUM_THREADS`). During the initialization phase, the library provides affinity thread-core mapping:

- If the number of cores (including SMT cores) is greater or equal than two times the number of threads, then all the threads can occupy every second core ID (e.g. 0, 2, 4, ...). This is to avoid having two threads working on the same physical core, when SMT is enabled.
- If the number of threads is less or equal to the number of cores (including SMT), and the previous clause is false, then the threads can occupy every core ID (e.g. 0, 1, 2, 3, ...).
- If non of the above criteria is matched, then the default thread-core mapping is used (typically set by the OS).

**Note** The thread-core mapping is available only for Unix-like OS.

**Note** The user can disable the thread affinity by calling `set_omp_affinity_rocalution()`, before initializing the library (i.e. before `init_rocalution()`).

**Parameters**

- `[in] nthreads`: number of OpenMP threads

```c
void rocalution::set_omp_affinity_rocalution (bool affinity)
Enable/disable OpenMP host affinity.
set_omp_affinity_rocalution enables / disables OpenMP host affinity.
```

**Parameters**

- `[in] affinity`: boolean to turn on/off OpenMP host affinity

```c
void rocalution::set_omp_threshold_rocalution (int threshold)
Set OpenMP threshold size.
Whenever you want to work on a small problem, you might observe that the OpenMP host backend is (slightly) slower than using no OpenMP. This is mainly attributed to the small amount of work, which every thread should perform and the large overhead of forking/joining threads. This can be avoid by the OpenMP threshold size parameter in rocALUTION. The default threshold is set to 10000, which means that all matrices under (and equal) this size will use only one thread (disregarding the number of OpenMP threads set in the system). The threshold can be modified with `set_omp_threshold_rocalution`.
```

**Parameters**

- `[in] threshold`: OpenMP threshold size

```c
void rocalution::info_rocalution (void)
Print info about rocALUTION.
info_rocalution prints information about the rocALUTION platform
```

**Warning**: doxygenfunction: Unable to resolve multiple matches for function “rocalution::info_rocalution” with arguments (const struct Rocalution_Backend_Descriptor) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- `void info_rocalution(const struct Rocalution_Backend_Descriptor backend_descriptor)`
- `void info_rocalution(void)`
void rocalution::disable_accelerator_rocalution (bool onoff = true)
        Disable/Enable the accelerator.
        If you want to disable the accelerator (without re-compiling the code), you need to call disable_accelerator_rocalution before init_rocalution().

        Parameters
        • [in] onoff: boolean to turn on/off the accelerator

void rocalution::_rocalution_sync (void)
        Sync rocALUTION.
        _rocalution_sync blocks the host until all active asynchronous transfers are completed.

2.12.8.12.3 Base Rocalution

template<typename ValueType>
class BaseRocalution : public rocalution::RocalutionObj
        Base class for all operators and vectors.

        Template Parameters
        • ValueType: - can be int, float, double, std::complex<float> and std::complex<double>

        Subclassed by rocalution::Operator<ValueType>, rocalution::Vector<ValueType>

void rocalution::BaseRocalution::MoveToAccelerator (void) = 0
        Move the object to the accelerator backend.

void rocalution::BaseRocalution::MoveToHost (void) = 0
        Move the object to the host backend.

void rocalution::BaseRocalution::MoveToAcceleratorAsync (void)
        Move the object to the accelerator backend with async move.

void rocalution::BaseRocalution::MoveToHostAsync (void)
        Move the object to the host backend with async move.

void rocalution::BaseRocalution::Sync (void)
        Sync (the async move)

        Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::BaseRocalution::CloneBackend” with arguments (const BaseRocalution<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:
        - template<typename ValueType2> void CloneBackend(const BaseRocalution<ValueType2> & src)
        - void CloneBackend(const BaseRocalution<ValueType> & src)

void rocalution::BaseRocalution::Info (void) const = 0
        Print object information.
        Info can print object information about any rocALUTION object. This information consists of object properties and backend data.

        Example
void rocalution::BaseRocalution::Clear (void) = 0
  Clear (free all data) the object.

2.12.8.12.4 Operator

template<type_name ValueType>
class Operator : public rocalution::BaseRocalution<ValueType>
  Operator class.

  The Operator class defines the generic interface for applying an operator (e.g. matrix or stencil) from/to global and local vectors.

  Template Parameters

    • ValueType: - can be int, float, double, std::complex<float> and std::complex<double>

  Subclassed by rocalution::GlobalMatrix< ValueType >, rocalution::LocalMatrix< ValueType >, rocalution::LocalStencil< ValueType >

IndexType2 rocalution::Operator::GetM (void) const = 0
  Return the number of rows in the matrix/stencil.

IndexType2 rocalution::Operator::GetN (void) const = 0
  Return the number of columns in the matrix/stencil.

IndexType2 rocalution::Operator::GetNnz (void) const = 0
  Return the number of non-zeros in the matrix/stencil.

int rocalution::Operator::GetLocalM (void) const
  Return the number of rows in the local matrix/stencil.

int rocalution::Operator::GetLocalN (void) const
  Return the number of columns in the local matrix/stencil.

int rocalution::Operator::GetLocalNnz (void) const
  Return the number of non-zeros in the local matrix/stencil.

int rocalution::Operator::GetGhostM (void) const
  Return the number of rows in the ghost matrix/stencil.

int rocalution::Operator::GetGhostN (void) const
  Return the number of columns in the ghost matrix/stencil.

int rocalution::Operator::GetGhostNnz (void) const
  Return the number of non-zeros in the ghost matrix/stencil.

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Operator::Apply” with arguments (const LocalVector<ValueType>&, LocalVector<ValueType> *) const in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void Apply(const GlobalVector<ValueType>& in, GlobalVector<ValueType> *out) const
- void Apply(const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
2.12.8.12.5 Vector

template<typename ValueType>
class Vector : public rocalution::BaseRocalution<ValueType>
{
    Vector class.
    
    The Vector class defines the generic interface for local and global vectors.
    
    **Template Parameters**
    
    - ValueType: - can be int, float, double, std::complex<float> and std::complex<double>

    Subclassed by rocalution::LocalVector< int >, rocalution::GlobalVector< ValueType >, rocalution::LocalVector< ValueType >

    IndexType2 rocalution::Vector::GetSize (void) const = 0
    
    Return the size of the vector.

    int rocalution::Vector::GetLocalSize (void) const
    
    Return the size of the local vector.

    int rocalution::Vector::GetGhostSize (void) const
    
    Return the size of the ghost vector.

    bool rocalution::Vector::Check (void) const = 0
    
    Perform a sanity check of the vector.
Checks, if the vector contains valid data, i.e. if the values are not infinity and not NaN (not a number).

Return Value

- true: if the vector is ok (empty vector is also ok).
- false: if there is something wrong with the values.

```cpp
void rocalution::Vector::Zeros (void) = 0
    Set all values of the vector to 0.

void rocalution::Vector::Ones (void) = 0
    Set all values of the vector to 1.

void rocalution::Vector::SetValues (ValueType val) = 0
    Set all values of the vector to given argument.

void rocalution::Vector::SetRandomUniform (unsigned long long seed, ValueType a = static_cast<ValueType>(-1), ValueType b = static_cast<ValueType>(1)) = 0
    Fill the vector with random values from interval [a,b].

void rocalution::Vector::SetRandomNormal (unsigned long long seed, ValueType mean = static_cast<ValueType>(0), ValueType var = static_cast<ValueType>(1)) = 0
    Fill the vector with random values from normal distribution.

void rocalution::Vector::ReadFileASCII (const std::string filename) = 0
    Read vector from ASCII file.

Example

```cpp
LocalVector<ValueType> vec;
vec.ReadFileASCII("my_vector.dat");
```

Parameters

- [in] filename: name of the file containing the ASCII data.

void rocalution::Vector::WriteFileASCII (const std::string filename) const = 0
    Write vector to ASCII file.

Example

```cpp
LocalVector<ValueType> vec;
// Allocate and fill vec
// ...
vec.WriteFileASCII("my_vector.dat");
```

Parameters

- [in] filename: name of the file to write the ASCII data to.
void rocalution::Vector::ReadFileBinary(const std::string filename) = 0

Read vector from binary file.

Read a vector from binary file. For details on the format, see WriteFileBinary().

Example

LocalVector<ValueType> vec;
vec.ReadFileBinary("my_vector.bin");

Parameters

• [in] filename: name of the file containing the data.

void rocalution::Vector::WriteFileBinary(const std::string filename) const = 0

Write vector to binary file.

Write a vector to binary file.

The binary format contains a header, the rocALUTION version and the vector data as follows

```cpp
// Header
out << "#rocALUTION binary vector file" << std::endl;

// rocALUTION version
out.write((char*)&version, sizeof(int));

// Vector data
out.write((char*)&size, sizeof(int));
out.write((char*)vec_val, size * sizeof(double));
```

Note  Vector values array is always stored in double precision (e.g. double or std::complex<double>).

Example

```cpp
LocalVector<ValueType> vec;
// Allocate and fill vec
// ...
vec.WriteFileBinary("my_vector.bin");
```

Parameters

• [in] filename: name of the file to write the data to.

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::CopyFrom” with arguments (const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void CopyFrom(const GlobalVector<ValueType> &src)
- void CopyFrom(const LocalVector<ValueType> &src)
- void CopyFrom(const LocalVector<ValueType> &src, int src_offset, int dst_offset,
  → int size)
Warning:  doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::CopyFrom” with arguments (const GlobalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void CopyFrom(const GlobalVector<ValueType> &src)
- void CopyFrom(const LocalVector<ValueType> &src)
- void CopyFrom(const LocalVector<ValueType> &src, int src_offset, int dst_offset, ...
  → int size)

void rocalution::Vector::CopyFromAsync (const LocalVector<ValueType> &src)
Async copy from another local vector.

void rocalution::Vector::CopyFromFloat (const LocalVector<float> &src)
Copy values from another local float vector.

void rocalution::Vector::CopyFromDouble (const LocalVector<double> &src)
Copy values from another local double vector.

Warning:  doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::CopyFrom” with arguments (const LocalVector<ValueType>&, int, int, int) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void CopyFrom(const GlobalVector<ValueType> &src)
- void CopyFrom(const LocalVector<ValueType> &src)
- void CopyFrom(const LocalVector<ValueType> &src, int src_offset, int dst_offset, ...
  → int size)

Warning:  doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::CloneFrom” with arguments (const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void CloneFrom(const GlobalVector<ValueType> &src)
- void CloneFrom(const LocalVector<ValueType> &src)

Warning:  doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::CloneFrom” with arguments (const GlobalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void CloneFrom(const GlobalVector<ValueType> &src)
- void CloneFrom(const LocalVector<ValueType> &src)

Warning:  doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::AddScale” with arguments (const LocalVector<ValueType>&, ValueType) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void AddScale(const GlobalVector<ValueType> &x, ValueType alpha)
- void AddScale(const LocalVector<ValueType> &x, ValueType alpha)
Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::AddScale” with arguments (const GlobalVector<ValueType>&, ValueType) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:
  - void AddScale(const GlobalVector<ValueType>& x, ValueType alpha)
  - void AddScale(const LocalVector<ValueType>& x, ValueType alpha)

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAdd” with arguments (ValueType, const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:
  - void ScaleAdd(ValueType alpha, const GlobalVector<ValueType>& x)
  - void ScaleAdd(ValueType alpha, const LocalVector<ValueType>& x)

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAdd” with arguments (ValueType, const GlobalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:
  - void ScaleAdd(ValueType alpha, const GlobalVector<ValueType>& x)
  - void ScaleAdd(ValueType alpha, const LocalVector<ValueType>& x)

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAddScale” with arguments (ValueType, const GlobalVector<ValueType>&, ValueType) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:
  - void ScaleAddScale(ValueType alpha, const GlobalVector<ValueType>& x, ValueType beta)
  - void ScaleAddScale(ValueType alpha, const GlobalVector<ValueType>& x, ValueType beta, int src_offset, int dst_offset, int size)
  - void ScaleAddScale(ValueType alpha, const LocalVector<ValueType>& x, ValueType beta)
  - void ScaleAddScale(ValueType alpha, const LocalVector<ValueType>& x, ValueType beta, int src_offset, int dst_offset, int size)

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAddScale” with arguments (ValueType, const GlobalVector<ValueType>&, ValueType) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:
  - void ScaleAddScale(ValueType alpha, const GlobalVector<ValueType>& x, ValueType beta)
  - void ScaleAddScale(ValueType alpha, const GlobalVector<ValueType>& x, ValueType beta, int src_offset, int dst_offset, int size)
  - void ScaleAddScale(ValueType alpha, const LocalVector<ValueType>& x, ValueType beta)
  - void ScaleAddScale(ValueType alpha, const LocalVector<ValueType>& x, ValueType beta, int src_offset, int dst_offset, int size)
Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAddScale” with arguments (ValueType, const LocalVector<ValueType>&, ValueType, int, int, int) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void ScaleAddScale(ValueType alpha, const GlobalVector<ValueType> &x, ValueType beta)
- void ScaleAddScale(ValueType alpha, const GlobalVector<ValueType> &x, ValueType beta, int src_offset, int dst_offset, int size)
- void ScaleAddScale(ValueType alpha, const LocalVector<ValueType> &x, ValueType beta)
- void ScaleAddScale(ValueType alpha, const LocalVector<ValueType> &x, ValueType beta, int src_offset, int dst_offset, int size)

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAdd2” with arguments (ValueType, const LocalVector<ValueType>&, ValueType, const LocalVector<ValueType>&, ValueType) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void ScaleAdd2(ValueType alpha, const GlobalVector<ValueType> &x, ValueType beta, const GlobalVector<ValueType> &y, ValueType gamma)
- void ScaleAdd2(ValueType alpha, const LocalVector<ValueType> &x, ValueType beta, const LocalVector<ValueType> &y, ValueType gamma)

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::ScaleAdd2” with arguments (ValueType, const GlobalVector<ValueType>&, ValueType, const GlobalVector<ValueType>&, ValueType) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void ScaleAdd2(ValueType alpha, const GlobalVector<ValueType> &x, ValueType beta, const GlobalVector<ValueType> &y, ValueType gamma)
- void ScaleAdd2(ValueType alpha, const LocalVector<ValueType> &x, ValueType beta, const LocalVector<ValueType> &y, ValueType gamma)

void rocalution::Vector::Scale (ValueType alpha) = 0
Perform vector scaling this = alpha * this.
**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::Dot” with arguments (const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- ValueType Dot(const GlobalVector<ValueType> &x) const
- ValueType Dot(const LocalVector<ValueType> &x) const

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::Dot” with arguments (const GlobalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- ValueType Dot(const GlobalVector<ValueType> &x) const
- ValueType Dot(const LocalVector<ValueType> &x) const

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::DotNonConj” with arguments (const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- ValueType DotNonConj(const GlobalVector<ValueType> &x) const
- ValueType DotNonConj(const LocalVector<ValueType> &x) const

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::DotNonConj” with arguments (const GlobalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- ValueType DotNonConj(const GlobalVector<ValueType> &x) const
- ValueType DotNonConj(const LocalVector<ValueType> &x) const

```cpp
ValueType rocalution::Vector::Norm(void) const = 0
Compute L_2 norm of the vector, return = sqrt(this^T this)
```

```cpp
ValueType rocalution::Vector::Reduce(void) const = 0
Reduce the vector.
```

```cpp
ValueType rocalution::Vector::Asum(void) const = 0
Compute the sum of absolute values of the vector, return = sum(|this|)
```

```cpp
int rocalution::Vector::Amax (ValueType &value) const = 0
Compute the absolute max of the vector, return = index(max(|this|))
```

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::Vector::PointWiseMult” with arguments (const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void PointWiseMult(const GlobalVector<ValueType> &x)
- void PointWiseMult(const GlobalVector<ValueType> &x, const GlobalVector<ValueType> &y)
- void PointWiseMult(const LocalVector<ValueType> &x)
- void PointWiseMult(const LocalVector<ValueType> &x, const LocalVector<ValueType> &y)
Warning: doxygenfunction: Unable to resolve multiple matches for function
“roculation::Vector::PointWiseMult” with arguments (const GlobalVector<ValueType>&) in doxygen xml output
for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void PointWiseMult(const GlobalVector<ValueType> &x)
- void PointWiseMult(const GlobalVector<ValueType> &x, const GlobalVector<ValueType> &y)
- void PointWiseMult(const LocalVector<ValueType> &x)
- void PointWiseMult(const LocalVector<ValueType> &x, const LocalVector<ValueType> &y)

Warning: doxygenfunction: Unable to resolve multiple matches for function
“roculation::Vector::PointWiseMult” with arguments (const LocalVector<ValueType>&, const LocalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void PointWiseMult(const GlobalVector<ValueType> &x)
- void PointWiseMult(const GlobalVector<ValueType> &x, const GlobalVector<ValueType> &y)
- void PointWiseMult(const LocalVector<ValueType> &x)
- void PointWiseMult(const LocalVector<ValueType> &x, const LocalVector<ValueType> &y)

Warning: doxygenfunction: Unable to resolve multiple matches for function
“roculation::Vector::PointWiseMult” with arguments (const GlobalVector<ValueType>&, const GlobalVector<ValueType>&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void PointWiseMult(const GlobalVector<ValueType> &x)
- void PointWiseMult(const GlobalVector<ValueType> &x, const GlobalVector<ValueType> &y)
- void PointWiseMult(const LocalVector<ValueType> &x)
- void PointWiseMult(const LocalVector<ValueType> &x, const LocalVector<ValueType> &y)

void rocalution::Vector::Power (double power) = 0
Perform power operation to a vector.
2.12.8.12.6 Local Matrix

```cpp
template<
    typename ValueType
>
class LocalMatrix  :
    public rocalution::Operator<ValueType> 
{
    
    A `LocalMatrix` is called local, because it will always stay on a single system. The system can contain several CPUs via UMA or NUMA memory system or it can contain an accelerator.

    **Template Parameters**

    - `ValueType`: can be int, float, double, std::complex<float> and std::complex<double

    unsigned int rocalution::LocalMatrix::GetFormat () const
    
    Return the matrix format id (see matrix_formats.hpp)

    bool rocalution::LocalMatrix::Check () const
    
    Perform a sanity check of the matrix.

    Checks, if the matrix contains valid data, i.e. if the values are not infinity and not NaN (not a number) and if the structure of the matrix is correct (e.g. indices cannot be negative, CSR and COO matrices have to be sorted, etc.).

    **Return Value**

    - `true`: if the matrix is ok (empty matrix is also ok).
    - `false`: if there is something wrong with the structure or values.

    void rocalution::LocalMatrix::AllocateCSR (const std::string name, int nnz, int nrow, int ncol)
    
    Allocate a local matrix with name and sizes.

    The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

    **Example**

    ```cpp
    LocalMatrix<ValueType> mat;

    mat.AllocateCSR("my CSR matrix", 456, 100, 100);
    mat.Clear();

    mat.AllocateCOO("my COO matrix", 200, 100, 100);
    mat.Clear();
    ```

    void rocalution::LocalMatrix::AllocateBCSR (void)
    
    Allocate a local matrix with name and sizes.

    The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

    **Example**

    ```cpp
    ```
```
void rocalution::LocalMatrix::AllocateCSR(const std::string name, int nnz, int nrow, int ncol)
Allocate a local matrix with name and sizes.
The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

Example

LocalMatrix<ValueType> mat;
mat.AllocateCSR("my CSR matrix", 456, 100, 100);
mat.Clear();
mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();

void rocalution::LocalMatrix::AllocateCOO(const std::string name, int nnz, int nrow, int ncol)
Allocate a local matrix with name and sizes.
The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

Example

LocalMatrix<ValueType> mat;
mat.AllocateCSR("my CSR matrix", 456, 100, 100);
mat.Clear();
mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();

void rocalution::LocalMatrix::AllocateDIA(const std::string name, int nnz, int nrow, int ncol, int ndiag)
Allocate a local matrix with name and sizes.
The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

Example
LocalMatrix<ValueType> mat;
mat.AllocateCSR("my CSR matrix", 456, 100, 100);
mat.Clear();
mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();

void rocalution::LocalMatrix::AllocateELL(const std::string name, int nnz, int nrow, int ncol, int max_row)

Allocate a local matrix with name and sizes.

The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

Example

LocalMatrix<ValueType> mat;
mat.AllocateCSR("my CSR matrix", 456, 100, 100);
mat.Clear();
mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();

void rocalution::LocalMatrix::AllocateHYB(const std::string name, int ell_nnz, int coo_nnz, int ell_max_row, int nrow, int ncol)

Allocate a local matrix with name and sizes.

The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

Example

LocalMatrix<ValueType> mat;
mat.AllocateCSR("my CSR matrix", 456, 100, 100);
mat.Clear();
mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();

void rocalution::LocalMatrix::AllocateDENSE(const std::string name, int nrow, int ncol)

Allocate a local matrix with name and sizes.

The local matrix allocation functions require a name of the object (this is only for information purposes) and corresponding number of non-zero elements, number of rows and number of columns. Furthermore, depending on the matrix format, additional parameters are required.

Example

LocalMatrix<ValueType> mat;
mat.AllocateCSR("my CSR matrix", 456, 100, 100);

(continues on next page)
mat.Clear();
mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();

```cpp
void rocalution::LocalMatrix::SetDataPtrCOO(int **row, int **col, ValueType **val, std::string name, int nnz, int nrow, int ncol)
```

Initialize a `LocalMatrix` on the host with externally allocated data.

`SetDataPtr` functions have direct access to the raw data via pointers. Already allocated data can be set by passing their pointers.

**Note** Setting data pointers will leave the original pointers empty (set to NULL).

**Example**

```cpp
// Allocate a CSR matrix
int* csr_row_ptr = new int[100 + 1];
int* csr_col_ind = new int[345];
ValueType* csr_val = new ValueType[345];

// Fill the CSR matrix
// ...

// rocALUTION local matrix object
LocalMatrix<ValueType> mat;

// Set the CSR matrix data, csr_row_ptr, csr_col_ind and csr_val pointers become invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100, 100);
```

```cpp
void rocalution::LocalMatrix::SetDataPtrCSR(int **row_offset, int **col, ValueType **val, std::string name, int nnz, int nrow, int ncol)
```

Initialize a `LocalMatrix` on the host with externally allocated data.

`SetDataPtr` functions have direct access to the raw data via pointers. Already allocated data can be set by passing their pointers.

**Note** Setting data pointers will leave the original pointers empty (set to NULL).

**Example**

```cpp
// Allocate a CSR matrix
int* csr_row_ptr = new int[100 + 1];
int* csr_col_ind = new int[345];
ValueType* csr_val = new ValueType[345];

// Fill the CSR matrix
// ...

// rocALUTION local matrix object
LocalMatrix<ValueType> mat;

// Set the CSR matrix data, csr_row_ptr, csr_col_ind and csr_val pointers become invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100, 100);
```
void rocALUTION::LocalMatrix::SetDataPtrCSR(int **row_offset, int **col, ValueType **val, std::string name, int nnz, int nrow, int ncol)

Initialize a LocalMatrix on the host with externally allocated data.

SetDataPtr functions have direct access to the raw data via pointers. Already allocated data can be set by passing their pointers.

Note Setting data pointers will leave the original pointers empty (set to NULL).

Example

```cpp
// Allocate a CSR matrix
int* csr_row_ptr = new int[100 + 1];
int* csr_col_ind = new int[345];
ValueType* csr_val = new ValueType[345];

// Fill the CSR matrix
// ...

// rocALUTION local matrix object
LocalMatrix<ValueType> mat;

// Set the CSR matrix data, csr_row_ptr, csr_col and csr_val pointers become invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100,
                   -100);
```

void rocALUTION::LocalMatrix::SetDataPtrELL(int **col, ValueType **val, std::string name, int nnz, int nrow, int ncol, int max_row)

Initialize a LocalMatrix on the host with externally allocated data.

SetDataPtr functions have direct access to the raw data via pointers. Already allocated data can be set by passing their pointers.

Note Setting data pointers will leave the original pointers empty (set to NULL).

Example

```cpp
// Allocate a CSR matrix
int* csr_row_ptr = new int[100 + 1];
int* csr_col_ind = new int[345];
ValueType* csr_val = new ValueType[345];

// Fill the CSR matrix
// ...

// rocALUTION local matrix object
LocalMatrix<ValueType> mat;

// Set the CSR matrix data, csr_row_ptr, csr_col and csr_val pointers become invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100,
                   -100);
```
void rocalution::LocalMatrix::SetDataPtrDIA(int **offset, ValueType **val, std::string name, int nnz, int nrow, int ncol, int num_diag)

Initialize a LocalMatrix on the host with externally allocated data.

SetDataPtr functions have direct access to the raw data via pointers. Already allocated data can be set by passing their pointers.

Note Setting data pointers will leave the original pointers empty (set to NULL).

Example

```cpp
// Allocate a CSR matrix
int* csr_row_ptr = new int[100 + 1];
int* csr_col_ind = new int[345];
ValueType* csr_val = new ValueType[345];

// Fill the CSR matrix
...

// rocALUTION local matrix object
LocalMatrix<ValueType> mat;

// Set the CSR matrix data, csr_row_ptr, csr_col and csr_val pointers become invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100, ...
˓→100);
```

void rocalution::LocalMatrix::SetDataPtrDENSE(ValueType **val, std::string name, int nrow, int ncol)

Initialize a LocalMatrix on the host with externally allocated data.

SetDataPtr functions have direct access to the raw data via pointers. Already allocated data can be set by passing their pointers.

Note Setting data pointers will leave the original pointers empty (set to NULL).

Example

```cpp
// Allocate a CSR matrix
int* csr_row_ptr = new int[100 + 1];
int* csr_col_ind = new int[345];
ValueType* csr_val = new ValueType[345];

// Fill the CSR matrix
...

// rocALUTION local matrix object
LocalMatrix<ValueType> mat;

// Set the CSR matrix data, csr_row_ptr, csr_col and csr_val pointers become invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100, ...
˓→100);
```

void rocalution::LocalMatrix::LeaveDataPtrCOO(int **row, int **col, ValueType **val)

Leave a LocalMatrix to host pointers.

LeaveDataPtr functions have direct access to the raw data via pointers. A LocalMatrix object can leave its raw data to host pointers. This will leave the LocalMatrix empty.
Example

```cpp
// rocALUTION CSR matrix object
LocalMatrix<ValueType> mat;

// Allocate the CSR matrix
mat.AllocateCSR("my_matrix", 345, 100, 100);

// Fill CSR matrix
// ...

int* csr_row_ptr = NULL;
int* csr_col_ind = NULL;
ValueType* csr_val = NULL;

// Get (steal) the data from the matrix, this will leave the local matrix empty
mat.LeaveDataPtrCSR(&csr_row_ptr, &csr_col_ind, &csr_val);
```

void rocalution::LocalMatrix::LeaveDataPtrCSR(int **row_offset, int **col, ValueType **val)

Leave a `LocalMatrix` to host pointers.

`LeaveDataPtr` functions have direct access to the raw data via pointers. A `LocalMatrix` object can leave its raw data to host pointers. This will leave the `LocalMatrix` empty.

Example

```cpp
// rocALUTION CSR matrix object
LocalMatrix<ValueType> mat;

// Allocate the CSR matrix
mat.AllocateCSR("my_matrix", 345, 100, 100);

// Fill CSR matrix
// ...

int* csr_row_ptr = NULL;
int* csr_col_ind = NULL;
ValueType* csr_val = NULL;

// Get (steal) the data from the matrix, this will leave the local matrix empty
mat.LeaveDataPtrMCSR(&csr_row_ptr, &csr_col_ind, &csr_val);
```

void rocalution::LocalMatrix::LeaveDataPtrMCSR(int **row_offset, int **col, ValueType **val)

Leave a `LocalMatrix` to host pointers.

`LeaveDataPtr` functions have direct access to the raw data via pointers. A `LocalMatrix` object can leave its raw data to host pointers. This will leave the `LocalMatrix` empty.

Example
// Allocate the CSR matrix
mat.AllocateCSR("my_matrix", 345, 100, 100);

// Fill CSR matrix
// ...

int* csr_row_ptr = NULL;
int* csr_col_ind = NULL;
ValueType* csr_val = NULL;

// Get (steal) the data from the matrix, this will leave the local matrix object empty
mat.LeaveDataPtrCSR(&csr_row_ptr, &csr_col_ind, &csr_val);

void rocalution::LocalMatrix::LeaveDataPtrELL(int **col, ValueType **val, int &max_row)
Leave a LocalMatrix to host pointers.

LeaveDataPtr functions have direct access to the raw data via pointers. A LocalMatrix object can leave its raw data to host pointers. This will leave the LocalMatrix empty.

Example

// rocALUTION CSR matrix object
LocalMatrix<ValueType> mat;

// Allocate the CSR matrix
mat.AllocateCSR("my_matrix", 345, 100, 100);

// Fill CSR matrix
// ...

int* csr_row_ptr = NULL;
int* csr_col_ind = NULL;
ValueType* csr_val = NULL;

// Get (steal) the data from the matrix, this will leave the local matrix object empty
mat.LeaveDataPtrCSR(&csr_row_ptr, &csr_col_ind, &csr_val);

void rocalution::LocalMatrix::LeaveDataPtrDIA(int **offset, ValueType **val, int &num_diag)
Leave a LocalMatrix to host pointers.

LeaveDataPtr functions have direct access to the raw data via pointers. A LocalMatrix object can leave its raw data to host pointers. This will leave the LocalMatrix empty.

Example

// rocALUTION CSR matrix object
LocalMatrix<ValueType> mat;

// Allocate the CSR matrix
mat.AllocateCSR("my_matrix", 345, 100, 100);

// Fill CSR matrix
void rocalution::LocalMatrix::LeaveDataPtrDENSE (ValueType **val)
Leave a LocalMatrix to host pointers.
LeaveDataPtr functions have direct access to the raw data via pointers. A LocalMatrix object can leave its raw data to host pointers. This will leave the LocalMatrix empty.

Example

// rocALUTION CSR matrix object
LocalMatrix<ValueType> mat;

// Allocate the CSR matrix
mat.AllocateCSR("my_matrix", 345, 100, 100);

// Fill CSR matrix
// ...

int* csr_row_ptr = NULL;
int* csr_col_ind = NULL;
ValueType* csr_val = NULL;

// Get (steal) the data from the matrix, this will leave the local matrix object empty
mat.LeaveDataPtrCSR(&csr_row_ptr, &csr_col_ind, &csr_val);

void rocalution::LocalMatrix::Zeros (void)
Set all matrix values to zero.

void rocalution::LocalMatrix::Scale (ValueType alpha)
Scale all values in the matrix.

void rocalution::LocalMatrix::ScaleDiagonal (ValueType alpha)
Scale the diagonal entries of the matrix with alpha, all diagonal elements must exist.

void rocalution::LocalMatrix::ScaleOffDiagonal (ValueType alpha)
Scale the off-diagonal entries of the matrix with alpha, all diagonal elements must exist.

void rocalution::LocalMatrix::AddScalar (ValueType alpha)
Add a scalar to all matrix values.

void rocalution::LocalMatrix::AddScalarDiagonal (ValueType alpha)
Add alpha to the diagonal entries of the matrix, all diagonal elements must exist.

void rocalution::LocalMatrix::AddScalarOffDiagonal (ValueType alpha)
Add alpha to the off-diagonal entries of the matrix, all diagonal elements must exist.
void rocalution::LocalMatrix::ExtractSubMatrix(int row_offset, int col_offset, int row_size, int col_size, LocalMatrix<ValueType> *mat) const

Extract a sub-matrix with row/col_offset and row/col_size.

void rocalution::LocalMatrix::ExtractSubMatrices(int row_num_blocks, int col_num_blocks, const int *row_offset, const int *col_offset, LocalMatrix<ValueType> ***mat) const

Extract array of non-overlapping sub-matrices (row/col_num_blocks define the blocks for rows/columns; row/col_offset have sizes col/row_num_blocks+1, where [i+1]-[i] defines the i-th size of the sub-matrix)

void rocalution::LocalMatrix::ExtractDiagonal(LocalVector<ValueType> *vec_diag) const

Extract the diagonal values of the matrix into a LocalVector.

void rocalution::LocalMatrix::ExtractInverseDiagonal(LocalVector<ValueType> *vec_inv_diag) const

Extract the inverse (reciprocal) diagonal values of the matrix into a LocalVector.

void rocalution::LocalMatrix::ExtractU(LocalMatrix<ValueType> *U, bool diag) const

Extract the upper triangular matrix.

void rocalution::LocalMatrix::ExtractL(LocalMatrix<ValueType> *L, bool diag) const

Extract the lower triangular matrix.

void rocalution::LocalMatrix::Permute(const LocalVector<int> &permutation)

Perform (forward) permutation of the matrix.

void rocalution::LocalMatrix::PermuteBackward(const LocalVector<int> &permutation)

Perform (backward) permutation of the matrix.

void rocalution::LocalMatrix::CMK(LocalVector<int> *permutation) const

Create permutation vector for CMK reordering of the matrix.

The Cuthill-McKee ordering minimize the bandwidth of a given sparse matrix.

Example

LocalVector<int> cmk;
mat.CMK(&cmk);
mat.Permute(cmk);

Parameters

* [out] permutation: permutation vector for CMK reordering

void rocalution::LocalMatrix::RCMK(LocalVector<int> *permutation) const

Create permutation vector for reverse CMK reordering of the matrix.

The Reverse Cuthill-McKee ordering minimize the bandwidth of a given sparse matrix.

Example

LocalVector<int> rcmk;
mat.RCMK(&rcmk);
mat.Permute(rcmk);

Parameters
• [out] permutation: permutation vector for reverse CMK reordering

```cpp
void rocalution::LocalMatrix::ConnectivityOrder(LocalVector<int> *permutation) const
```
Create permutation vector for connectivity reordering of the matrix.

Connectivity ordering returns a permutation, that sorts the matrix by non-zero entries per row.

**Example**

```cpp
LocalVector<int> conn;
mat.ConnectivityOrder(&conn);
mat.Permute(conn);
```

**Parameters**

- [out] permutation: permutation vector for connectivity reordering

```cpp
void rocalution::LocalMatrix::MultiColoring(int &num_colors, int **size_colors, LocalVector<int> *permutation) const
```
Perform multi-coloring decomposition of the matrix.

The Multi-Coloring algorithm builds a permutation (coloring of the matrix) in a way such that no two adjacent nodes in the sparse matrix have the same color.

**Example**

```cpp
LocalVector<int> mc;
int num_colors;
int* block_colors = NULL;
mat.MultiColoring(num_colors, &block_colors, &mc);
mat.Permute(mc);
```

**Parameters**

- [out] num_colors: number of colors
- [out] size_colors: pointer to array that holds the number of nodes for each color
- [out] permutation: permutation vector for multi-coloring reordering

```cpp
void rocalution::LocalMatrix::MaximalIndependentSet(int &size, LocalVector<int> *permutation) const
```
Perform maximal independent set decomposition of the matrix.

The Maximal Independent Set algorithm finds a set with maximal size, that contains elements that do not depend on other elements in this set.

**Example**

```cpp
LocalVector<int> mis;
int size;
mat.MaximalIndependentSet(size, &mis);
mat.Permute(mis);
```

**Parameters**

- [out] size: number of independent sets
• [out] permutation: permutation vector for maximal independent set reordering

```cpp
void rocalution::LocalMatrix::ZeroBlockPermutation (int &size, LocalVector<int> *permutation) const
```

Return a permutation for saddle-point problems (zero diagonal entries)

For Saddle-Point problems, (i.e. matrices with zero diagonal entries), the Zero Block Permutation maps all zero-diagonal elements to the last block of the matrix.

**Example**

```cpp
LocalVector<int> zbp;
int size;
mat.ZeroBlockPermutation(size, &zbp);
mat.Permute(zbp);
```

**Parameters**

• [out] size:
• [out] permutation: permutation vector for zero block permutation

```cpp
void rocalution::LocalMatrix::ILUFactorize (void)
Perform ILU(0) factorization.
```

```cpp
void rocalution::LocalMatrix::LUFactorize (void)
Perform LU factorization.
```

```cpp
void rocalution::LocalMatrix::ILUTFactorize (double t, int maxrow)
Perform ILU(t,m) factorization based on threshold and maximum number of elements per row.
```

```cpp
void rocalution::LocalMatrix::ILUpFactorize (int p, bool level = true)
Perform ILU(p) factorization based on power.
```

```cpp
void rocalution::LocalMatrix::LUAnalyse (void)
Analyse the structure (level-scheduling)
```

```cpp
void rocalution::LocalMatrix::LUAnalyseClear (void)
Delete the analysed data (see LUAnalyse)
```

```cpp
void rocalution::LocalMatrix::LUSolve (const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
Solve LU out = in; if level-scheduling algorithm is provided then the graph traversing is performed in parallel.
```

```cpp
void rocalution::LocalMatrix::ICFactorize (LocalVector<ValueType> *inv_diag)
Perform IC(0) factorization.
```

```cpp
void rocalution::LocalMatrix::LLAnalyse (void)
Analyse the structure (level-scheduling)
```

```cpp
void rocalution::LocalMatrix::LLAnalyseClear (void)
Delete the analysed data (see LLAnalyse)
```

---

**Warning:** doxygenfunction: Unable to resolve multiple matches for function "rocalution::LocalMatrix::LLSolve" with arguments (const LocalVector<ValueType>&, LocalVector<ValueType> *) in doxygen xml output for project "rocALUTION" from directory: rocALUTIONxml/. Potential matches:

- void LLSolve(const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
- void LLSolve(const LocalVector<ValueType> &in, const LocalVector<ValueType> &inv_˓
→diag, LocalVector<ValueType> *out) const

484 Chapter 2. Solid Compilation Foundation and Language Support
**Warning:** doxygenfunction: Unable to resolve multiple matches for function "rocalution::LocalMatrix::LLSolve" with arguments (const LocalVector<ValueType>&, const LocalVector<ValueType>&, LocalVector<ValueType> *) cons) in doxygen xml output for project "rocALUTION" from directory: rocALUTIONxml/. Potential matches:

- void LLSolve(const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
- void LLSolve(const LocalVector<ValueType> &in, const LocalVector<ValueType> &inv_ →diag, LocalVector<ValueType> *out) const

```cpp
definition rocalution::LocalMatrix::LAnalyse(bool diag_unit = false)
    Analyse the structure (level-scheduling) L-part.
    - diag_unit == true the diag is 1;
    - diag_unit == false the diag is 0;

definition rocalution::LocalMatrix::LAnalyseClear(void)
    Delete the analysed data (see LAnalyse) L-part.

definition rocalution::LocalMatrix::LSolve(const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
    Solve L out = in; if level-scheduling algorithm is provided then the graph traversing is performed in parallel.

definition rocalution::LocalMatrix::UAnalyse(bool diag_unit = false)
    Analyse the structure (level-scheduling) U-part:
    - diag_unit == true the diag is 1;
    - diag_unit == false the diag is 0;

definition rocalution::LocalMatrix::UAnalyseClear(void)
    Delete the analysed data (see UAnalyse) U-part.

definition rocalution::LocalMatrix::USolve(const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
    Solve U out = in; if level-scheduling algorithm is provided then the graph traversing is performed in parallel.

definition rocalution::LocalMatrix::Householder(int idx, ValueType &beta, LocalVector<ValueType> *vec) const
    Compute Householder vector.

definition rocalution::LocalMatrix::QRDecompose(void)
    QR Decomposition.

definition rocalution::LocalMatrix::QRSolve(const LocalVector<ValueType> &in, LocalVector<ValueType> *out) const
    Solve QR out = in.

definition rocalution::LocalMatrix::Invert(void)
    Matrix inversion using QR decomposition.

definition rocalution::LocalMatrix::ReadFileMTX(const std::string filename)
    Read a matrix from Matrix Market Format file.
```
Example

```cpp
LocalMatrix<ValueType> mat;
mat.ReadFileMTX("my_matrix.mtx");
```

Parameters

• `[in] filename`: name of the file containing the MTX data.

```cpp
void rocalution::LocalMatrix::WriteFileMTX(const std::string filename) const
Write matrix to MTX (Matrix Market Format) file.
Write a matrix to Matrix Market Format file.
```

Example

```cpp
LocalMatrix<ValueType> mat;
// Allocate and fill mat
// ...
mat.WriteFileMTX("my_matrix.mtx");
```

Parameters

• `[in] filename`: name of the file to write the MTX data to.

```cpp
void rocalution::LocalMatrix::ReadFileCSR(const std::string filename)
Read matrix from CSR (rocALUTION binary format) file.
Read a CSR matrix from binary file. For details on the format, see `WriteFileCSR()`.
```

Example

```cpp
LocalMatrix<ValueType> mat;
mat.ReadFileCSR("my_matrix.csr");
```

Parameters

• `[in] filename`: name of the file containing the data.

```cpp
void rocalution::LocalMatrix::WriteFileCSR(const std::string filename) const
Write CSR matrix to binary file.
Write a CSR matrix to binary file.
```

The binary format contains a header, the rocALUTION version and the matrix data as follows

```cpp
// Header
out << "#rocALUTION binary csr file" << std::endl;
// rocALUTION version
out.write((char*) &version, sizeof(int));
// CSR matrix data
out.write((char*) &m, sizeof(int));
out.write((char*) &n, sizeof(int));
out.write((char*) &nnz, sizeof(int));
out.write((char*) csr_row_ptr, (m + 1) * sizeof(int));
```
out.write((char*)csr_col_ind, nnz * sizeof(int));
out.write((char*)csr_val, nnz * sizeof(double));

Note Vector values array is always stored in double precision (e.g. double or std::complex<double>).

Example

```
LocalMatrix<ValueType> mat;
// Allocate and fill mat
// ...
mat.WriteFileCSR("my_matrix.csr");
```

Parameters

- [in] filename: name of the file to write the data to.

```cpp
void rocalution::LocalMatrix::CopyFrom(const LocalMatrix<ValueType> &src)
```

Copy matrix from another LocalMatrix.

CopyFrom copies values and structure from another local matrix. Source and destination matrix should be in the same format.

Note This function allows cross platform copying. One of the objects could be allocated on the accelerator backend.

Example

```
LocalMatrix<ValueType> mat1, mat2;
// Allocate and initialize mat1 and mat2
// ...
// Move mat1 to accelerator
// mat1.MoveToAccelerator();

// Now, mat1 is on the accelerator (if available)
// and mat2 is on the host

// Copy mat1 to mat2 (or vice versa) will move data between host and accelerator backend
mat1.CopyFrom(mat2);
```

Parameters

- [in] src: Local matrix where values and structure should be copied from.

```cpp
void rocalution::LocalMatrix::CopyFromAsync(const LocalMatrix<ValueType> &src)
```

Async copy matrix (values and structure) from another LocalMatrix.

```cpp
void rocalution::LocalMatrix::CloneFrom(const LocalMatrix<ValueType> &src)
```

Clone the matrix.

CloneFrom clones the entire matrix, including values, structure and backend descriptor from another LocalMatrix.

Example

2.12. ROCm Libraries
LocalMatrix<ValueType> mat;

// Allocate and initialize mat (host or accelerator)
// ...

LocalMatrix<ValueType> tmp;

// By cloning mat, tmp will have identical values and structure and will be on
// the same backend as mat
tmp.CloneFrom(mat);

Parameters

- [in] src: LocalMatrix to clone from.

void rocalution::LocalMatrix::UpdateValuesCSR (ValueType *val)
  Update CSR matrix entries only, structure will remain the same.

void rocalution::LocalMatrix::CopyFromCSR (const int *row_offets, const int *col, const ValueType *val)
  Copy (import) CSR matrix described in three arrays (offsets, columns, values). The object data has to be allocated (call AllocateCSR first)

void rocalution::LocalMatrix::CopyToCSR (int *row_offsets, int *col, ValueType *val) const
  Copy (export) CSR matrix described in three arrays (offsets, columns, values). The output arrays have to be allocated.

void rocalution::LocalMatrix::CopyFromCOO (const int *row, const int *col, const ValueType *val)
  Copy (import) COO matrix described in three arrays (rows, columns, values). The object data has to be allocated (call AllocateCOO first)

void rocalution::LocalMatrix::CopyToCOO (int *row, int *col, ValueType *val) const
  Copy (export) COO matrix described in three arrays (rows, columns, values). The output arrays have to be allocated.

void rocalution::LocalMatrix::CopyFromHostCSR (const int *row_offset, const int *col, const ValueType *val, const std::string name, int nnz, int nrow, int ncol)
  Allocates and copies (imports) a host CSR matrix.

If the CSR matrix data pointers are only accessible as constant, the user can create a LocalMatrix object and pass const CSR host pointers. The LocalMatrix will then be allocated and the data will be copied to the corresponding backend, where the original object was located at.

Parameters

- [in] val: CSR matrix values array.
- [in] name: Matrix object name.
- [in] nnz: Number of non-zero elements.
- [in] nrow: Number of rows.
- [in] ncol: Number of columns.
void rocALUTION::LocalMatrix::ConvertToCSR(void)
Convert the matrix to CSR structure.

void rocALUTION::LocalMatrix::ConvertToMCSR(void)
Convert the matrix to MCSR structure.

void rocALUTION::LocalMatrix::ConvertToBCSR(void)
Convert the matrix to BCSR structure.

void rocALUTION::LocalMatrix::ConvertToCOO(void)
Convert the matrix to COO structure.

void rocALUTION::LocalMatrix::ConvertToELL(void)
Convert the matrix to ELL structure.

void rocALUTION::LocalMatrix::ConvertToDIA(void)
Convert the matrix to DIA structure.

void rocALUTION::LocalMatrix::ConvertToHYB(void)
Convert the matrix to HYB structure.

void rocALUTION::LocalMatrix::ConvertToDENSE(void)
Convert the matrix to DENSE structure.

void rocALUTION::LocalMatrix::ConvertTo(unsigned int matrix_format)
Convert the matrix to specified matrix ID format.

void rocALUTION::LocalMatrix::SymbolicPower(int p)
Perform symbolic computation (structure only) of |this|^p.

void rocALUTION::LocalMatrix::MatrixAdd(const LocalMatrix<ValueType> &mat, ValueType alpha = static_cast<ValueType>(1), ValueType beta = static_cast<ValueType>(1), bool structure = false)
Perform matrix addition, this = alpha*this + beta*mat.

  • if structure==false the sparsity pattern of the matrix is not changed;
  • if structure==true a new sparsity pattern is computed
void rocalution::LocalMatrix::MatrixMult(const LocalMatrix<ValueType> &A, const LocalMatrix<ValueType> &B)
    Multiply two matrices, this = A * B.

void rocalution::LocalMatrix::DiagonalMatrixMult(const LocalMatrix<ValueType> &A, const LocalVector<ValueType> &diag)
    Multiply the matrix with diagonal matrix (stored in LocalVector), as DiagonalMatrixMultR()

void rocalution::LocalMatrix::DiagonalMatrixMultL(const LocalVector<ValueType> &diag)
    Multiply the matrix with diagonal matrix (stored in LocalVector), this=diag*this.

void rocalution::LocalMatrix::DiagonalMatrixMultR(const LocalVector<ValueType> &diag)
    Multiply the matrix with diagonal matrix (stored in LocalVector), this=this*diag.

void rocalution::LocalMatrix::Gershgorin(ValueType &lambda_min, ValueType &lambda_max) const
    Compute the spectrum approximation with Gershgorin circles theorem.

void rocalution::LocalMatrix::Compress(double drop_off)
    Delete all entries in the matrix which abs(a_ij) <= drop_off; the diagonal elements are never deleted.

void rocalution::LocalMatrix::Transpose(void)
    Transpose the matrix.

void rocalution::LocalMatrix::Sort(void)
    Sort the matrix indices.
    Sorts the matrix by indices.
    • For CSR matrices, column values are sorted.
    • For COO matrices, row indices are sorted.

void rocalution::LocalMatrix::Key(long int &row_key, long int &col_key, long int &val_key) const
    Compute a unique hash key for the matrix arrays.

    Typically, it is hard to compare if two matrices have the same structure (and values). To do so, rocALUTION provides a keying function, that generates three keys, for the row index, column index and values array.

    Parameters

    • [out] row_key: row index array key
    • [out] col_key: column index array key
    • [out] val_key: values array key

void rocalution::LocalMatrix::ReplaceColumnVector(int idx, const LocalVector<ValueType> &vec)
    Replace a column vector of a matrix.

void rocalution::LocalMatrix::ReplaceRowVector(int idx, const LocalVector<ValueType> &vec)
    Replace a row vector of a matrix.

void rocalution::LocalMatrix::ExtractColumnVector(int idx, LocalVector<ValueType> *vec) const
    Extract values from a column of a matrix to a vector.

void rocalution::LocalMatrix::ExtractRowVector(int idx, LocalVector<ValueType> *vec) const
    Extract values from a row of a matrix to a vector.
void rocalution::LocalMatrix::AMGConnect (ValueType eps, LocalVector<int> *connections) const

Strong couplings for aggregation-based AMG.

void rocalution::LocalMatrix::AMGAggregate (const LocalVector<int> &connections, LocalVector<int> *aggregates) const

Plain aggregation - Modification of a greedy aggregation scheme from Vanek (1996)

void rocalution::LocalMatrix::AMGSmoothedAggregation (ValueType relax, const LocalVector<int> &aggregates, const LocalVector<int> &connections, LocalMatrix<ValueType> *prolong, LocalMatrix<ValueType> *restrict) const

Interpolation scheme based on smoothed aggregation from Vanek (1996)

void rocalution::LocalMatrix::AGMAggregation (const LocalVector<int> &aggregates, LocalMatrix<ValueType> *prolong, LocalMatrix<ValueType> *restrict) const

Aggregation-based interpolation scheme.

void rocalution::LocalMatrix::RugeStueben (ValueType eps, LocalMatrix<ValueType> *prolong, LocalMatrix<ValueType> *restrict) const

Ruge Stueben coarsening.

void rocalution::LocalMatrix::FSAI (int power, const LocalMatrix<ValueType> *pattern)

Factorized Sparse Approximate Inverse assembly for given system matrix power pattern or external sparsity pattern.

void rocalution::LocalMatrix::SPAI (void)

Sparse Approximate Inverse assembly for given system matrix pattern.
Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::LocalMatrix::FurtherPairwiseAggregation” with arguments (ValueType, int&, LocalVector<int> *, int&, int**, int&, int) cons) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void FurtherPairwiseAggregation(ValueType beta, int &nc, LocalVector<int> *G, int &Gsize, int **rG, int &rGsize, int ordering) const
- void FurtherPairwiseAggregation(const LocalMatrix<ValueType> &mat, ValueType beta, int &nc, LocalVector<int> *G, int &Gsize, int **rG, int &rGsize, int ordering) const

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::LocalMatrix::FurtherPairwiseAggregation” with arguments (const LocalMatrix<ValueType>&, ValueType, int&, LocalVector<int>* , int&, int**, int&, int) cons) in doxygen xml output for project "rocALUTION" from directory: rocALUTIONxml/. Potential matches:

- void FurtherPairwiseAggregation(ValueType beta, int &nc, LocalVector<int> *G, int &Gsize, int **rG, int &rGsize, int ordering) const
- void FurtherPairwiseAggregation(const LocalMatrix<ValueType> &mat, ValueType beta, int &nc, LocalVector<int> *G, int &Gsize, int **rG, int &rGsize, int ordering) const

void rocalution::LocalMatrix::CoarsenOperator (LocalMatrix<ValueType> *Ac, int nrow, int ncol, const LocalVector<int> &G, int Gsize, const int *rG, int rGsize) const

Build coarse operator for pairwise aggregation scheme.

2.12.8.12.7 Local Stencil

template<typename ValueType>
class LocalStencil : public rocalution::Operator<ValueType>

LocalStencil class.

A LocalStencil is called local, because it will always stay on a single system. The system can contain several CPUs via UMA or NUMA memory system or it can contain an accelerator.

Template Parameters

- ValueType: - can be int, float, double, std::complex<float> and std::complex<double>

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::LocalStencil::LocalStencil” with arguments (unsigned int) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- LocalStencil()
- LocalStencil(unsigned int type)

int rocalution::LocalStencil::GetNDim (void) const

Return the dimension of the stencil.
void rocalution::LocalStencil::SetGrid(int size)
Set the stencil grid size.

2.12.8.12.8 Global Matrix

template<typename ValueType>
class GlobalMatrix : public rocalution::Operator<ValueType>
GlobalMatrix class.

A GlobalMatrix is called global, because it can stay on a single or on multiple nodes in a network. For this type of communication, MPI is used.

Template Parameters

- ValueType: - can be int, float, double, std::complex<float> and std::complex<double>

<table>
<thead>
<tr>
<th>Warning:</th>
<th>doxygenfunction: Unable to resolve multiple matches for function “rocalution::GlobalMatrix::GlobalMatrix” with arguments (const ParallelManager&amp;) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- GlobalMatrix()</td>
<td></td>
</tr>
<tr>
<td>- GlobalMatrix(const ParallelManager &amp;pm)</td>
<td></td>
</tr>
</tbody>
</table>

bool rocalution::GlobalMatrix::Check (void) const
Return true if the matrix is ok (empty matrix is also ok) and false if there is something wrong with the structure or some of values are NaN.

void rocalution::GlobalMatrix::AllocateCSR (std::string name, int local_nnz, int ghost_nnz)
Allocate CSR Matrix.

void rocalution::GlobalMatrix::AllocateCOO (std::string name, int local_nnz, int ghost_nnz)
Allocate COO Matrix.

void rocalution::GlobalMatrix::SetParallelManager (const ParallelManager &pm)
Set the parallel manager of a global vector.

void rocalution::GlobalMatrix::SetDataPtrCSR (int **local_row_offset, int **local_col, ValueType **local_val, int **ghost_row_offset, int **ghost_col, ValueType **ghost_val, std::string name, int local_nnz, int ghost_nnz)
Initialize a CSR matrix on the host with externally allocated data.

void rocalution::GlobalMatrix::SetDataPtrCOO (int **local_row, int **local_col, ValueType **local_val, int **ghost_row, int **ghost_col, ValueType **ghost_val, std::string name, int local_nnz, int ghost_nnz)
Initialize a COO matrix on the host with externally allocated data.

void rocalution::GlobalMatrix::SetLocalDataPtrCSR (int **row_offset, int **col, ValueType **val, std::string name, int nnz)
Initialize a CSR matrix on the host with externally allocated local data.

void rocalution::GlobalMatrix::SetLocalDataPtrCOO (int **row, int **col, ValueType **val, std::string name, int nnz)
Initialize a COO matrix on the host with externally allocated local data.
void rocalution::GlobalMatrix::SetGhostDataPtrCSR (int **row_offset, int **col, ValueType **val, std::string name, int nnz)
Initialize a CSR matrix on the host with externally allocated ghost data.

void rocalution::GlobalMatrix::SetGhostDataPtrCOO (int **row, int **col, ValueType **val, std::string name, int nnz)
Initialize a COO matrix on the host with externally allocated ghost data.

void rocalution::GlobalMatrix::LeaveDataPtrCSR (int **local_row_offset, int **local_col, ValueType **local_val, int **ghost_row_offset, int **ghost_col, ValueType **ghost_val)
Leave a CSR matrix to host pointers.

void rocalution::GlobalMatrix::LeaveDataPtrCOO (int **local_row, int **local_col, ValueType **local_val, int **ghost_row, int **ghost_col, ValueType **ghost_val)
Leave a COO matrix to host pointers.

void rocalution::GlobalMatrix::LeaveLocalDataPtrCSR (int **row_offset, int **col, ValueType **val)
Leave a local CSR matrix to host pointers.

void rocalution::GlobalMatrix::LeaveLocalDataPtrCOO (int **row, int **col, ValueType **val)
Leave a local COO matrix to host pointers.

void rocalution::GlobalMatrix::LeaveGhostDataPtrCSR (int **row_offset, int **col, ValueType **val)
Leave a CSR ghost matrix to host pointers.

void rocalution::GlobalMatrix::LeaveGhostDataPtrCOO (int **row, int **col, ValueType **val)
Leave a COO ghost matrix to host pointers.

void rocalution::GlobalMatrix::CloneFrom (const GlobalMatrix<ValueType> &src)
Clone the entire matrix (values,structure+backend descr) from another GlobalMatrix.

void rocalution::GlobalMatrix::CopyFrom (const GlobalMatrix<ValueType> &src)
Copy matrix (values and structure) from another GlobalMatrix.

void rocalution::GlobalMatrix::ConvertToCSR (void)
Convert the matrix to CSR structure.

void rocalution::GlobalMatrix::ConvertToMCSR (void)
Convert the matrix to MCSR structure.

void rocalution::GlobalMatrix::ConvertToBCSR (void)
Convert the matrix to BCSR structure.

void rocalution::GlobalMatrix::ConvertToCOO (void)
Convert the matrix to COO structure.

void rocalution::GlobalMatrix::ConvertToELL (void)
Convert the matrix to ELL structure.

void rocalution::GlobalMatrix::ConvertToDIA (void)
Convert the matrix to DIA structure.

void rocalution::GlobalMatrix::ConvertToHYB (void)
Convert the matrix to HYB structure.

void rocalution::GlobalMatrix::ConvertToDENSE (void)
Convert the matrix to DENSE structure.
void rocalution::GlobalMatrix::ConvertTo(unsigned int matrix_format)
    Convert the matrix to specified matrix ID format.

void rocalution::GlobalMatrix::ReadFileMTX(const std::string filename)
    Read matrix from MTX (Matrix Market Format) file.

void rocalution::GlobalMatrix::WriteFileMTX(const std::string filename) const
    Write matrix to MTX (Matrix Market Format) file.

void rocalution::GlobalMatrix::ReadFileCSR(const std::string filename)
    Read matrix from CSR (ROCALUTION binary format) file.

void rocalution::GlobalMatrix::WriteFileCSR(const std::string filename) const
    Write matrix to CSR (ROCALUTION binary format) file.

void rocalution::GlobalMatrix::Sort(void)
    Sort the matrix indices.

void rocalution::GlobalMatrix::ExtractInverseDiagonal(GlobalVector<ValueType>* vec_inv_diag) const
    Extract the inverse (reciprocal) diagonal values of the matrix into a GlobalVector.

void rocalution::GlobalMatrix::Scale(ValueType alpha)
    Scale all the values in the matrix.

void rocalution::GlobalMatrix::InitialPairwiseAggregation(ValueType beta, int &nc,
    LocalVector<int>* G, int &Gsize, int **rG, int &rGsize, int ordering) const
    Initial Pairwise Aggregation scheme.

void rocalution::GlobalMatrix::FurtherPairwiseAggregation(ValueType beta, int &nc,
    LocalVector<int>* G, int &Gsize, int **rG, int &rGsize, int ordering) const
    Further Pairwise Aggregation scheme.

void rocalution::GlobalMatrix::CoarsenOperator(GlobalMatrix<ValueType>* Ac,
    ParallelManager *pm, int nrow, int ncol, const LocalVector<int> &G, int Gsize, const
    int *rG, int rGsize) const
    Build coarse operator for pairwise aggregation scheme.

2.12.8.12.9 Local

```cpp
template<typename ValueType>
class LocalVector : public rocalution::Vector<ValueType>
    LocalVector class.
```

A LocalVector is called local, because it will always stay on a single system. The system can contain several CPUs via UMA or NUMA memory system or it can contain an accelerator.

**Template Parameters**

- **ValueType**: can be int, float, double, std::complex<float> and std::complex<double>

```cpp
void rocalution::LocalVector::Allocate(std::string name, IndexType2 size)
    Allocate a local vector with name and size.
```
The local vector allocation function requires a name of the object (this is only for information purposes) and corresponding size description for vector objects.

Example

```cpp
LocalVector<ValueType> vec;
vec.Allocate("my vector", 100);
vec.Clear();
```

Parameters

- [in] name: object name
- [in] size: number of elements in the vector

```cpp
void rocalution::LocalVector::SetDataPtr(ValueType **ptr, std::string name, int size)
```

Initialize a `LocalVector` on the host with externally allocated data.

`SetDataPtr` has direct access to the raw data via pointers. Already allocated data can be set by passing the pointer.

**Note** Setting data pointer will leave the original pointer empty (set to `NULL`).

Example

```cpp
// Allocate vector
ValueType* ptr_vec = new ValueType[200];

// Fill vector
// ...

// rocALUTION local vector object
LocalVector<ValueType> vec;

// Set the vector data, ptr_vec will become invalid
vec.SetDataPtr(&ptr_vec, "my_vector", 200);
```

```cpp
void rocalution::LocalVector::LeaveDataPtr(ValueType **ptr)
```

Leave a `LocalVector` to host pointers.

`LeaveDataPtr` has direct access to the raw data via pointers. A `LocalVector` object can leave its raw data to a host pointer. This will leave the `LocalVector` empty.

Example

```cpp
// rocALUTION local vector object
LocalVector<ValueType> vec;

// Allocate the vector
vec.Allocate("my_vector", 100);

// Fill vector
// ...

ValueType* ptr_vec = NULL;
```

(continues on next page)
// Get (steal) the data from the vector, this will leave the local vector object empty
vec.LeaveDataPtr(&ptr_vec);

Warning: doxygenfunction: Unable to resolve multiple matches for function
"rocalution::LocalVector::operator[]" with arguments (int) in doxygen xml output for project "rocALUTION"
from directory: rocALUTIONxml/. Potential matches:
- ValueType &operator[](int i)
- const ValueType &operator[](int i) const

Warning: doxygenfunction: Unable to resolve multiple matches for function
"rocalution::LocalVector::operator[]" with arguments (int) const in doxygen xml output for project "rocALUTION"
from directory: rocALUTIONxml/. Potential matches:
- ValueType &operator[](int i)
- const ValueType &operator[](int i) const

void rocalution::LocalVector::CopyFromPermute(const LocalVector<ValueType> &src, const LocalVector<int> &permutation)
Copy a vector under permutation (forward permutation)

void rocalution::LocalVector::CopyFromPermuteBackward(const LocalVector<ValueType> &src, const LocalVector<int> &permutation)
Copy a vector under permutation (backward permutation)

void rocalution::LocalVector::CopyFromData(const ValueType *data)
Copy (import) vector.

Parameters
- [in] data: data to be imported.

void rocalution::LocalVector::CopyToData(ValueType *data) const
Copy (export) vector.

Parameters
- [out] data: exported data.

void rocalution::LocalVector::Permute(const LocalVector<int> &permutation)
Perform in-place permutation (forward) of the vector.

void rocalution::LocalVector::PermuteBackward(const LocalVector<int> &permutation)
Perform in-place permutation (backward) of the vector.
void rocalution::LocalVector::Restriction(const LocalVector<ValueType> &vec_fine, const LocalVector<int> &map)

Restriction operator based on restriction mapping vector.

void rocalution::LocalVector::Prolongation(const LocalVector<ValueType> &vec_coarse, const LocalVector<int> &map)

Prolongation operator based on restriction mapping vector.

void rocalution::LocalVector::SetIndexArray(int size, const int *index)

Set index array.

void rocalution::LocalVector::GetIndexValues(ValueType *values) const

Get indexed values.

void rocalution::LocalVector::SetIndexValues(const ValueType *values)

Set indexed values.

void rocalution::LocalVector::GetContinuousValues(int start, int end, ValueType *values) const

Get continuous indexed values.

void rocalution::LocalVector::SetContinuousValues(int start, int end, const ValueType *values)

Set continuous indexed values.

void rocalution::LocalVector::ExtractCoarseMapping(int start, int end, const int *index, int nc, int *size, int *map) const

Extract coarse boundary mapping.

void rocalution::LocalVector::ExtractCoarseBoundary(int start, int end, const int *index, int nc, int *size, int *boundary) const

Extract coarse boundary index.

2.12.8.12.10 Global Vector

class GlobalVector : public rocalution::Vector<ValueType>

GlobalVector class.

A GlobalVector is called global, because it can stay on a single or on multiple nodes in a network. For this type of communication, MPI is used.

Template Parameters

- ValueType: - can be int, float, double, std::complex<float> and std::complex<double>

Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::GlobalVector::GlobalVector” with arguments (const ParallelManager&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- GlobalVector()
- GlobalVector(const ParallelManager &pm)

void rocalution::GlobalVector::Allocate (std::string name, IndexType2 size)

Allocate a global vector with name and size.
void rocalution::GlobalVector::SetParallelManager(const ParallelManager &pm)
    Set the parallel manager of a global vector.

    **Warning:** doxygenfunction: Unable to resolve multiple matches for function
    “rocalution::GlobalVector::operator[]” with arguments (int) in doxygen xml output for project “rocALUTION”
    from directory: rocALUTIONxml/. Potential matches:

    - ValueType &operator[](int i)
    - const ValueType &operator[](int i) const

    **Warning:** doxygenfunction: Unable to resolve multiple matches for function
    “rocalution::GlobalVector::operator[]” with arguments (int) const in doxygen xml output for project
    “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

    - ValueType &operator[](int i)
    - const ValueType &operator[](int i) const

void rocalution::GlobalVector::SetDataPtr(ValueType **ptr, std::string name, IndexType2 size)
    Initialize the local part of a global vector with externally allocated data.

void rocalution::GlobalVector::LeaveDataPtr(ValueType **ptr)
    Get a pointer to the data from the local part of a global vector and free the global vector object.

void rocalution::GlobalVector::Restriction(const GlobalVector<ValueType> &vec_fine, const LocalVector<int> &map)
    Restriction operator based on restriction mapping vector.

void rocalution::GlobalVector::Prolongation(const GlobalVector<ValueType> &vec_coarse, const LocalVector<int> &map)
    Prolongation operator based on restriction mapping vector.

## 2.12.8.12.11 Parallel Manager

class ParallelManager : public rocalution::RocalutionObj
    Parallel Manager class.

    The parallel manager class handles the communication and the mapping of the global operators. Each global
    operator and vector need to be initialized with a valid parallel manager in order to perform any operation. For
    many distributed simulations, the underlying operator is already distributed. This information need to be passed
    to the parallel manager.

void rocalution::ParallelManager::SetMPICommunicator(const void *comm)
    Set the MPI communicator.

void rocalution::ParallelManager::Clear(void)
    Clear all allocated resources.

IndexType2 rocalution::ParallelManager::GetGlobalSize(void) const
    Return the global size.

int rocalution::ParallelManager::GetLocalSize(void) const
    Return the local size.
int rocalution::ParallelManager::GetNumReceivers (void) const
    Return the number of receivers.

int rocalution::ParallelManager::GetNumSenders (void) const
    Return the number of senders.

int rocalution::ParallelManager::GetNumProcs (void) const
    Return the number of involved processes.

void rocalution::ParallelManager::SetGlobalSize (IndexType2 size)
    Initialize the global size.

void rocalution::ParallelManager::SetLocalSize (int size)
    Initialize the local size.

void rocalution::ParallelManager::SetBoundaryIndex (int size, const int *index)
    Set all boundary indices of this ranks process.

void rocalution::ParallelManager::SetReceivers (int nrecv, const int *recvs, const int *recv_offset)
    Number of processes, the current process is receiving data from, array of the processes, the current process is receiving data from and offsets, where the boundary for process ‘receiver’ starts.

void rocalution::ParallelManager::SetSenders (int nsend, const int *sends, const int *send_offset)
    Number of processes, the current process is sending data to, array of the processes, the current process is sending data to and offsets where the ghost part for process ‘sender’ starts.

void rocalution::ParallelManager::LocalToGlobal (int proc, int local, int &global)
    Mapping local to global.

void rocalution::ParallelManager::GlobalToLocal (int global, int &proc, int &local)
    Mapping global to local.

bool rocalution::ParallelManager::Status (void) const
    Check sanity status of parallel manager.

void rocalution::ParallelManager::ReadFileASCII (const std::string filename)
    Read file that contains all relevant parallel manager data.

void rocalution::ParallelManager::WriteFileASCII (const std::string filename) const
    Write file that contains all relevant parallel manager data.

2.12.8.12 Solvers

template<class OperatorType, class VectorType, typename ValueType>
class Solver : public rocalution::RocalutionObj
    Base class for all solvers and preconditioners.

    Most of the solvers can be performed on linear operators LocalMatrix, LocalStencil and GlobalMatrix - i.e. the solvers can be performed locally (on a shared memory system) or in a distributed manner (on a cluster) via MPI. The only exception is the AMG (Algebraic Multigrid) solver which has two versions (one for LocalMatrix and one for GlobalMatrix class). The only pure local solvers (which do not support global/MPI operations) are the mixed-precision defect-correction solver and all direct solvers.

    All solvers need three template parameters - Operators, Vectors and Scalar type.

    The Solver class is purely virtual and provides an interface for

        • SetOperator() to set the operator A, i.e. the user can pass the matrix here.
• **Build()** to build the solver (including preconditioners, sub-solvers, etc.). The user need to specify the operator first before calling **Build().**

• **Solve()** to solve the system $Ax = b$. The user need to pass a right-hand-side $b$ and a vector $x$, where the solution will be obtained.

• **Print()** to show solver information.

• **ReBuildNumeric()** to only re-build the solver numerically (if possible).

• **MoveToHost()** and **MoveToAccelerator()** to offload the solver (including preconditioners and sub-solvers) to the host/accelerator.

**Template Parameters**

- **OperatorType**: - can be *LocalMatrix*, *GlobalMatrix* or *LocalStencil*
- **VectorType**: - can be *LocalVector* or *GlobalVector*
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by `rocalution::DirectLinearSolver< OperatorType, VectorType, ValueType >`, `rocalution::IterativeLinearSolver< OperatorType, VectorType, ValueType >`, `rocalution::Preconditioner< OperatorType, VectorType, ValueType >`

```c++
void rocalution::Solver::SetOperator(const OperatorType &op)  
Set the Operator of the solver.

void rocalution::Solver::ResetOperator(const OperatorType &op)  
Reset the operator; see ReBuildNumeric()

void rocalution::Solver::Print(void) const = 0  
Print information about the solver.

void rocalution::Solver::Solve(const VectorType &rhs, VectorType *x) = 0  
Solve Operator $x = rhs$.

void rocalution::Solver::SolveZeroSol(const VectorType &rhs, VectorType *x)  
Solve Operator $x = rhs$, setting initial $x = 0$.

void rocalution::Solver::Clear(void)  
Clear (free all local data) the solver.

void rocalution::Solver::Build(void)  
Build the solver (data allocation, structure and numerical computation)

void rocalution::Solver::BuildMoveToAcceleratorAsync(void)  
Build the solver and move it to the accelerator asynchronously.

void rocalution::Solver::Sync(void)  
Synchronize the solver.

void rocalution::Solver::ReBuildNumeric(void)  
Rebuild the solver only with numerical computation (no allocation or data structure computation)

void rocalution::Solver::MoveToHost(void)  
Move all data (i.e. move the solver) to the host.

void rocalution::Solver::MoveToAccelerator(void)  
Move all data (i.e. move the solver) to the accelerator.

void rocalution::Solver::Verbose(int verb = 1)  
Provide verbose output of the solver.
```
• verb = 0 -> no output
• verb = 1 -> print info about the solver (start, end);
• verb = 2 -> print (iter, residual) via iteration control;

template<class OperatorType, class VectorType, typename ValueType>
class IterativeLinearSolver : public rocakution::Solver<OperatorType, VectorType, ValueType>
Base class for all linear iterative solvers.

The iterative solvers are controlled by an iteration control object, which monitors the convergence properties of
the solver, i.e. maximum number of iteration, relative tolerance, absolute tolerance and divergence tolerance.
The iteration control can also record the residual history and store it in an ASCII file.

• Init(), InitMinIter(), InitMaxIter() and InitTol() initialize the solver and set the stopping criteria.
• RecordResidualHistory() and RecordHistory() start the recording of the residual and write it into a file.
• Verbose() sets the level of verbose output of the solver (0 - no output, 2 - detailed output, including residual
and iteration information).
• SetPreconditioner() sets the preconditioning.

All iterative solvers are controlled based on

• Absolute stopping criteria, when $|r_k|_{L_p} \leq \epsilon_{abs}$
• Relative stopping criteria, when $|r_k|_{L_p} / |r_1|_{L_p} \leq \epsilon_{rel}$
• Divergence stopping criteria, when $|r_k|_{L_p} / |r_1|_{L_p} \geq \epsilon_{div}$
• Maximum number of iteration $N$, when $k = N$

where $k$ is the current iteration, $r_k$ the residual for the current iteration $k$ (i.e. $r_k = b - Ax_k$) and $r_1$ the starting
residual (i.e. $r_1 = b - Ax_{init}$). In addition, the minimum number of iterations $M$ can be specified. In this case,
the solver will not stop to iterate, before $k \geq M$.

The $L_p$ norm is used for the computation, where $p$ could be 1, 2 and $\infty$. The norm computation can be set
with SetResidualNorm() with 1 for $L_1$, 2 for $L_2$ and 3 for $L_\infty$. For the computation with $L_\infty$, the index of
the maximum value can be obtained with GetAmaxResidualIndex(). If this function is called and $L_\infty$ was not
selected, this function will return -1.

The reached criteria can be obtained with GetSolverStatus(), returning

• 0, if no criteria has been reached yet
• 1, if absolute tolerance has been reached
• 2, if relative tolerance has been reached
• 3, if divergence tolerance has been reached
• 4, if maximum number of iteration has been reached

Template Parameters

• OperatorType: - can be LocalMatrix, GlobalMatrix or LocalStencil
• VectorType: - can be LocalVector or GlobalVector
• ValueType: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by rocakution::BaseMultiGrid< OperatorType, VectorType, ValueType >, rocakution::BiCGStab<
OperatorType, VectorType, ValueType >, rocakution::BiCGStabl< OperatorType, VectorType, ValueType
Warning: doxygenfunction: Unable to resolve multiple matches for function 
"rocalution::IterativeLinearSolver::Init" with arguments (double, double, double, int) in doxygen xml output for 
project "rocALUTION" from directory: rocALUTIONxml/. Potential matches:

- void Init(double abs_tol, double rel_tol, double div_tol, int max_iter)
- void Init(double abs_tol, double rel_tol, double div_tol, int min_iter, int max_iter)

Warning: doxygenfunction: Unable to resolve multiple matches for function 
"rocalution::IterativeLinearSolver::Init" with arguments (double, double, double, int, int) in doxygen xml output 
for project "rocALUTION" from directory: rocALUTIONxml/. Potential matches:

- void Init(double abs_tol, double rel_tol, double div_tol, int max_iter)
- void Init(double abs_tol, double rel_tol, double div_tol, int min_iter, int max_iter)

void rocalution::IterativeLinearSolver::InitMinIter(int min_iter)
Set the minimum number of iterations.

void rocalution::IterativeLinearSolver::InitMaxIter(int max_iter)
Set the maximum number of iterations.

void rocalution::IterativeLinearSolver::InitTol(double abs, double rel, double div)
Set the absolute/relative/divergence tolerance.

void rocalution::IterativeLinearSolver::SetResidualNorm(int resnorm)
Set the residual norm to $L_1$, $L_2$ or $L_\infty$ norm.

- resnorm = 1 -> $L_1$ norm
- resnorm = 2 -> $L_2$ norm
- resnorm = 3 -> $L_\infty$ norm

void rocalution::IterativeLinearSolver::RecordResidualHistory(void)
Record the residual history.

void rocalution::IterativeLinearSolver::RecordHistory(const std::string & filename)
Write the history to file.

void rocalution::IterativeLinearSolver::Verbose(int verb = 1)
Set the solver verbosity output.

void rocalution::IterativeLinearSolver::Solve(const VectorType & rhs, VectorType * x)
Solve $\text{Operator } x = \text{rhs}$. 

2.12. ROCm Libraries
void rocalution::IterativeLinearSolver::SetPreconditioner(Solver<OperatorType, VectorType, ValueType>& precond)

Set a preconditioner of the linear solver.

int rocalution::IterativeLinearSolver::GetIterationCount(void)

Return the iteration count.

double rocalution::IterativeLinearSolver::GetCurrentResidual(void)

Return the current residual.

int rocalution::IterativeLinearSolver::GetSolverStatus(void)

Return the current status.

int rocalution::IterativeLinearSolver::GetAmaxResidualIndex(void)

Return absolute maximum index of residual vector when using $L_{\infty}$ norm.

template<class OperatorType, class VectorType, typename ValueType>
class FixedPoint : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
Fixed-Point Iteration Scheme.

The Fixed-Point iteration scheme is based on additive splitting of the matrix $A = M + N$. The scheme reads

$$x_{k+1} = M^{-1}(b - N x_k).$$

It can also be reformulated as a weighted defect correction scheme

$$x_{k+1} = x_k - \omega M^{-1}(Ax_k - b).$$

The inversion of $M$ can be performed by preconditioners (Jacobi, Gauss-Seidel, ILU, etc.) or by any type of solvers.

**Template Parameters**

- OperatorType: - can be LocalMatrix, GlobalMatrix or LocalStencil
- VectorType: - can be LocalVector or GlobalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

void rocalution::FixedPoint::SetRelaxation(ValueType omega)

Set relaxation parameter $\omega$.

template<class OperatorTypeH, class VectorTypeH, typename ValueTypeH, class OperatorTypeL, class VectorTypeL, typename ValueTypeL>
class MixedPrecisionDC : public rocalution::IterativeLinearSolver<OperatorTypeH, VectorTypeH, ValueTypeH>
Mixed-Precision Defect Correction Scheme.

The Mixed-Precision solver is based on a defect-correction scheme. The current implementation of the library is using host based correction in double precision and accelerator computation in single precision. The solver is implementing the scheme

$$x_{k+1} = x_k + A^{-1} r_k,$$

where the computation of the residual $r_k = b - Ax_k$ and the update $x_{k+1} = x_k + d_k$ are performed on the host in double precision. The computation of the residual system $Ad_k = r_k$ is performed on the accelerator in single precision. In addition to the setup functions of the iterative solver, the user need to specify the inner ($Ad_k = r_k$) solver.

**Template Parameters**

- OperatorTypeH: - can be LocalMatrix

504 Chapter 2. Solid Compilation Foundation and Language Support
• VectorTypeH: - can be \textit{LocalVector}
• ValueTypeH: - can be double
• OperatorTypeL: - can be \textit{LocalMatrix}
• VectorTypeL: - can be \textit{LocalVector}
• ValueTypeL: - can be float

\begin{verbatim}
void rocalution::MixedPrecisionDC::Set(Solver<OperatorTypeL, VectorTypeL, ValueTypeL> &Solver_L)
\end{verbatim}

Set the inner solver for $A_d k = r_k$.

\begin{verbatim}
template<class OperatorType, class VectorType, typename ValueType>
class Chebyshev : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
\end{verbatim}

\textit{Chebyshev} Iteration Scheme.

The \textit{Chebyshev} Iteration scheme (also known as acceleration scheme) is similar to the \textit{CG} method but requires minimum and maximum eigenvalues of the operator.

\textbf{Template Parameters}

• OperatorType: - can be \textit{LocalMatrix, GlobalMatrix} or \textit{LocalStencil}
• VectorType: - can be \textit{LocalVector} or \textit{GlobalVector}
• ValueType: - can be float, double, std::complex<float> or std::complex<double>

\begin{verbatim}
void rocalution::Chebyshev::Set(ValueType lambda_min, ValueType lambda_max)
\end{verbatim}

Set the minimum and maximum eigenvalues of the operator.

\begin{verbatim}
template<class OperatorType, class VectorType, typename ValueType>
class BiCGStab : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
\end{verbatim}

\textit{Bi-Conjugate Gradient Stabilized Method.}

The Bi-Conjugate Gradient Stabilized method is a variation of CGS and solves sparse (non) symmetric linear systems $A x = b$. SAAD

\textbf{Template Parameters}

• OperatorType: - can be \textit{LocalMatrix, GlobalMatrix} or \textit{LocalStencil}
• VectorType: - can be \textit{LocalVector} or \textit{GlobalVector}
• ValueType: - can be float, double, std::complex<float> or std::complex<double>

\begin{verbatim}
template<class OperatorType, class VectorType, typename ValueType>
class BiCGStabl : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
\end{verbatim}

\textit{Bi-Conjugate Gradient Stabilized (l) Method.}

The Bi-Conjugate Gradient Stabilized (l) method is a generalization of \textit{BiCGStab} for solving sparse (non) symmetric linear systems $A x = b$. It minimizes residuals over $l$-dimensional Krylov subspaces. The degree $l$ can be set with \textit{SetOrder()}. bicgstabl

\textbf{Template Parameters}

• OperatorType: - can be \textit{LocalMatrix} or \textit{GlobalMatrix}
• VectorType: - can be \textit{LocalVector} or \textit{GlobalVector}
• ValueType: - can be float, double, std::complex<float> or std::complex<double>
void rocalution::BiCGStabl::SetOrder(int l)
Set the order.

template<class OperatorType, class VectorType, typename ValueType>
class CG : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
    Conjugate Gradient Method.

The Conjugate Gradient method is the best known iterative method for solving sparse symmetric positive definite (SPD) linear systems $Ax = b$. It is based on orthogonal projection onto the Krylov subspace $K_m(r_0, A)$, where $r_0$ is the initial residual. The method can be preconditioned, where the approximation should also be SPD. SAAD

**Template Parameters**

- **OperatorType**: - can be `LocalMatrix`, `GlobalMatrix` or `LocalStencil`
- **VectorType**: - can be `LocalVector` or `GlobalVector`
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class CR : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
    Conjugate Residual Method.

The Conjugate Residual method is an iterative method for solving sparse symmetric semi-positive definite linear systems $Ax = b$. It is a Krylov subspace method and differs from the much more popular Conjugate Gradient method that the system matrix is not required to be positive definite. The method can be preconditioned where the approximation should also be SPD or semi-positive definite. SAAD

**Template Parameters**

- **OperatorType**: - can be `LocalMatrix`, `GlobalMatrix` or `LocalStencil`
- **VectorType**: - can be `LocalVector` or `GlobalVector`
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class FCG : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
    Flexible Conjugate Gradient Method.

The Flexible Conjugate Gradient method is an iterative method for solving sparse symmetric positive definite linear systems $Ax = b$. It is similar to the Conjugate Gradient method with the only difference, that it allows the preconditioner $M^{-1}$ to be not a constant operator. This can be especially helpful if the operation $M^{-1}x$ is the result of another iterative process and not a constant operator. fcg

**Template Parameters**

- **OperatorType**: - can be `LocalMatrix` or `GlobalMatrix`
- **VectorType**: - can be `LocalVector` or `GlobalVector`
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class GMRES : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
    Generalized Minimum Residual Method.

The Generalized Minimum Residual method (GMRES) is a projection method for solving sparse (non) symmetric linear systems $Ax = b$, based on restarting technique. The solution is approximated in a Krylov subspace...
\( \mathcal{K} = \mathcal{K}_m \) and \( \mathcal{L} = A\mathcal{K}_m \) with minimal residual, where \( \mathcal{K}_m \) is the \( m \)-th Krylov subspace with \( v_1 = r_0/\|r_0\|_2 \).

SAAD

The Krylov subspace basis size can be set using \( \text{SetBasisSize()} \). The default size is 30.

**Template Parameters**

- \( \text{OperatorType} \): can be \( \text{LocalMatrix} \), \( \text{GlobalMatrix} \) or \( \text{LocalStencil} \)
- \( \text{VectorType} \): can be \( \text{LocalVector} \) or \( \text{GlobalVector} \)
- \( \text{ValueType} \): can be float, double, std::complex<float> or std::complex<double>

```cpp
template<class OperatorType, class VectorType, typename ValueType>
void rocalution::GMRES::SetBasisSize(int size_basis)
Set the size of the Krylov subspace basis.
```

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class FGMRES : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
Flexible Generalized Minimum Residual Method.
```

The Flexible Generalized Minimum Residual method (FGMRES) is a projection method for solving sparse (non) symmetric linear systems \( Ax = b \). It is similar to the GMRES method with the only difference, the FGMRES is based on a window shifting of the Krylov subspace and thus allows the preconditioner \( M^{-1} \) to be not a constant operator. This can be especially helpful if the operation \( M^{-1}x \) is the result of another iterative process and not a constant operator. SAAD

The Krylov subspace basis size can be set using \( \text{SetBasisSize()} \). The default size is 30.

**Template Parameters**

- \( \text{OperatorType} \): can be \( \text{LocalMatrix} \), \( \text{GlobalMatrix} \) or \( \text{LocalStencil} \)
- \( \text{VectorType} \): can be \( \text{LocalVector} \) or \( \text{GlobalVector} \)
- \( \text{ValueType} \): can be float, double, std::complex<float> or std::complex<double>

```cpp
template<class OperatorType, class VectorType, typename ValueType>
void rocalution::FGMRES::SetBasisSize(int size_basis)
Set the size of the Krylov subspace basis.
```

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class IDR : public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
Induced Dimension Reduction Method.
```

The Induced Dimension Reduction method is a Krylov subspace method for solving sparse (non) symmetric linear systems \( Ax = b \). IDR(s) generates residuals in a sequence of nested subspaces. IDR1 IDR2

The dimension of the shadow space can be set by \( \text{SetShadowSpace()} \). The default size of the shadow space is 4.

**Template Parameters**

- \( \text{OperatorType} \): can be \( \text{LocalMatrix} \), \( \text{GlobalMatrix} \) or \( \text{LocalStencil} \)
- \( \text{VectorType} \): can be \( \text{LocalVector} \) or \( \text{GlobalVector} \)
- \( \text{ValueType} \): can be float, double, std::complex<float> or std::complex<double>

```cpp
template<class OperatorType, class VectorType, typename ValueType>
void rocalution::IDR::SetShadowSpace(int s)
Set the size of the Shadow Space.
```

```cpp
template<class OperatorType, class VectorType, typename ValueType>
void rocalution::IDR::SetRandomSeed(unsigned long long seed)
Set random seed for ONB creation (seed must be greater than 0)
```

---

2.12. ROCm Libraries 507
class QMRCGStab: public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
Quasi-Minimal Residual Conjugate Gradient Stabilized Method.

The Quasi-Minimal Residual Conjugate Gradient Stabilized method is a variant of the Krylov subspace BiCGStab method for solving sparse (non) symmetric linear systems $Ax = b$. qmrcgstab

Template Parameters
- OperatorType: - can be LocalMatrix or GlobalMatrix
- VectorType: - can be LocalVector or GlobalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class BaseMultiGrid: public rocalution::IterativeLinearSolver<OperatorType, VectorType, ValueType>
Base class for all multigrid solvers Trottenberg2003.

Template Parameters
- OperatorType: - can be LocalMatrix or GlobalMatrix
- VectorType: - can be LocalVector or GlobalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by rocalution::BaseAMG< OperatorType, VectorType, ValueTyp e >, rocalution::MultiGrid< OperatorType, VectorType, ValueType >

void rocalution::BaseMultiGrid::SetSolver (Solver<OperatorType, VectorType, ValueType> &solver)
Set the coarse grid solver.

void rocalution::BaseMultiGrid::SetSmoother (IterativeLinearSolver<OperatorType, VectorType, ValueType> **smoother)
Set the smoother for each level.

void rocalution::BaseMultiGrid::SetSmootherPreIter (int iter)
Set the number of pre-smoothing steps.

void rocalution::BaseMultiGrid::SetSmootherPostIter (int iter)
Set the number of post-smoothing steps.

void rocalution::BaseMultiGrid::SetRestrictOperator (OperatorType **op) = 0
Set the restriction operator for each level.

void rocalution::BaseMultiGrid::SetProlongOperator (OperatorType **op) = 0
Set the prolongation operator for each level.

void rocalution::BaseMultiGrid::SetOperatorHierarchy (OperatorType **op) = 0
Set the operator for each level.

void rocalution::BaseMultiGrid::SetScaling (bool scaling)
Enable/disable scaling of intergrid transfers.

void rocalution::BaseMultiGrid::SetHostLevels (int levels)
Force computation of coarser levels on the host backend.

void rocalution::BaseMultiGrid::SetCycle (unsigned int cycle)
Set the MultiGrid Cycle (default: Vcycle)

void rocalution::BaseMultiGrid::SetKcycleFull (bool kcycle_full)
Set the MultiGrid Kcycle on all levels or only on finest level.
void rocalution::BaseMultiGrid::InitLevels (int levels)
Set the depth of the multigrid solver.

template<class OperatorType, class VectorType, typename ValueType>
class MultiGrid : public rocalution::BaseMultiGrid<OperatorType, VectorType, ValueType>
 MultiGrid Method.

The MultiGrid method can be used with external data, such as externally computed restriction, prolongation and operator hierarchy. The user need to pass all this information for each level and for its construction. This includes smoothing step, prolongation/restriction, grid traversing and coarse grid solver. This data need to be passed to the solver. Trottenberg2003

- Restriction and prolongation operations can be performed in two ways, based on Restriction() and Prolongation() of the LocalVector class, or by matrix-vector multiplication. This is configured by a set function.
- Smoothers can be of any iterative linear solver. Valid options are Jacobi, Gauss-Seidel, ILU, etc. using a FixedPoint iteration scheme with pre-defined number of iterations. The smoothers could also be a solver such as CG, BiCGStab, etc.
- Coarse grid solver could be of any iterative linear solver type. The class also provides mechanisms to specify, where the coarse grid solver has to be performed, on the host or on the accelerator. The coarse grid solver can be preconditioned.
- Grid scaling based on a $L_2$ norm ratio.
- Operator matrices need to be passed on each grid level.

Template Parameters
- OperatorType: - can be LocalMatrix or GlobalMatrix
- VectorType: - can be LocalVector or GlobalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class BaseAMG : public rocalution::BaseMultiGrid<OperatorType, VectorType, ValueType>
 Base class for all algebraic multigrid solvers.

The Algebraic MultiGrid solver is based on the BaseMultiGrid class. The coarsening is obtained by different aggregation techniques. The smoothers can be constructed inside or outside of the class.

All parameters in the Algebraic MultiGrid class can be set externally, including smoothers and coarse grid solver.

Template Parameters
- OperatorType: - can be LocalMatrix or GlobalMatrix
- VectorType: - can be LocalVector or GlobalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by rocalution::GlobalPairwiseAMG< OperatorType, VectorType, ValueType >, rocalution::PairwiseAMG< OperatorType, VectorType, ValueType >, rocalution::RugeStuebenAMG< OperatorType, VectorType, ValueType >, rocalution::SAAMG< OperatorType, VectorType, ValueType >, rocalution::UAAMG< OperatorType, VectorType, ValueType >

void rocalution::BaseAMG::ClearLocal (void)
Clear all local data.
void rocalution::BaseAMG::BuildHierarchy (void)
Create AMG hierarchy.

void rocalution::BaseAMG::BuildSmoothers (void)
Create AMG smoothers.

void rocalution::BaseAMG::SetCoarsestLevel (int coarse_size)
Set coarsest level for hierarchy creation.

void rocalution::BaseAMG::SetManualSmoothers (bool sm_manual)
Set flag to pass smoothers manually for each level.

void rocalution::BaseAMG::SetManualSolver (bool s_manual)
Set flag to pass coarse grid solver manually.

void rocalution::BaseAMG::SetDefaultSmootherFormat (unsigned int op_format)
Set the smoother operator format.

void rocalution::BaseAMG::SetOperatorFormat (unsigned int op_format)
Set the operator format.

int rocalution::BaseAMG::GetNumLevels (void)
Returns the number of levels in hierarchy.

template<class OperatorType, class VectorType, typename ValueType>
class UAAMG : public rocalution::BaseAMG<OperatorType, VectorType, ValueType>
Unsmoothed Aggregation Algebraic MultiGrid Method.

The Unsmoothed Aggregation Algebraic MultiGrid method is based on unsmoothed aggregation based interpolation scheme. stuben

Template Parameters

- OperatorType: - can be LocalMatrix
- VectorType: - can be LocalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

void rocalution::UAAMG::SetCouplingStrength (ValueType eps)
Set coupling strength.

void rocalution::UAAMG::SetOverInterp (ValueType overInterp)
Set over-interpolation parameter for aggregation.

template<class OperatorType, class VectorType, typename ValueType>
class SAAMG : public rocalution::BaseAMG<OperatorType, VectorType, ValueType>
Smoothed Aggregation Algebraic MultiGrid Method.

The Smoothed Aggregation Algebraic MultiGrid method is based on smoothed aggregation based interpolation scheme. vanek

Template Parameters

- OperatorType: - can be LocalMatrix
- VectorType: - can be LocalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

void rocalution::SAAMG::SetCouplingStrength (ValueType eps)
Set coupling strength.
void rocalsion::SAAMG::SetInterpRelax (ValueType relax)
  Set the relaxation parameter.

template<class OperatorType, class VectorType, typename ValueType>
class RugeStuebenAMG : public rocalsion::BaseAMG<OperatorType, VectorType, ValueType>
Ruge-Stueben Algebraic MultiGrid Method.

The Ruge-Stueben Algebraic MultiGrid method is based on the classic Ruge-Stueben coarsening with direct interpolation. The solver provides high-efficiency in terms of complexity of the solver (i.e. number of iterations). However, most of the time it has a higher building step and requires higher memory usage.

Template Parameters
  • OperatorType: - can be LocalMatrix
  • VectorType: - can be LocalVector
  • ValueType: - can be float, double, std::complex<float> or std::complex<double>

void rocalsion::RugeStuebenAMG::SetCouplingStrength (ValueType eps)
  Set coupling strength.

template<class OperatorType, class VectorType, typename ValueType>
class PairwiseAMG : public rocalsion::BaseAMG<OperatorType, VectorType, ValueType>
Pairwise Aggregation Algebraic MultiGrid Method.

The Pairwise Aggregation Algebraic MultiGrid method is based on a pairwise aggregation matching scheme. It delivers very efficient building phase which is suitable for Poisson-like equation. Most of the time it requires K-cycle for the solving phase to provide low number of iterations.

Template Parameters
  • OperatorType: - can be LocalMatrix
  • VectorType: - can be LocalVector
  • ValueType: - can be float, double, std::complex<float> or std::complex<double>

void rocalsion::PairwiseAMG::SetBeta (ValueType beta)
  Set beta for pairwise aggregation.

void rocalsion::PairwiseAMG::SetOrdering (unsigned int ordering)
  Set re-ordering for aggregation.

void rocalsion::PairwiseAMG::SetCoarseningFactor (double factor)
  Set target coarsening factor.

template<class OperatorType, class VectorType, typename ValueType>
class GlobalPairwiseAMG : public rocalsion::BaseAMG<OperatorType, VectorType, ValueType>
Pairwise Aggregation Algebraic MultiGrid Method (multi-node)

The Pairwise Aggregation Algebraic MultiGrid method is based on a pairwise aggregation matching scheme. It delivers very efficient building phase which is suitable for Poisson-like equation. Most of the time it requires K-cycle for the solving phase to provide low number of iterations. This version has multi-node support.

Template Parameters
  • OperatorType: - can be GlobalMatrix
  • VectorType: - can be GlobalVector
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

```cpp
void rocalution::GlobalPairwiseAMG::SetBeta (ValueType beta)
Set beta for pairwise aggregation.
```

```cpp
void rocalution::GlobalPairwiseAMG::SetOrdering (const _aggregation_ordering ordering)
Set re-ordering for aggregation.
```

```cpp
void rocalution::GlobalPairwiseAMG::SetCoarseningFactor (double factor)
Set target coarsening factor.
```

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>
### 2.12. ROCm Libraries

#### class QR

```cpp
class QR : public rocalution::DirectLinearSolver<OperatorType, VectorType, ValueType>
```

**QR Decomposition.**

The **QR** Decomposition decomposes a given matrix into $A = QR$, such that $Q$ is an orthogonal matrix and $R$ an upper triangular matrix.

**Template Parameters**

- **OperatorType**: - can be `LocalMatrix`
- **VectorType**: - can be `LocalVector`
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

#### 2.12.8.12.13 Preconditioners

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class Preconditioner : public rocalution::Solver<OperatorType, VectorType, ValueType>
```

Base class for all preconditioners.

**Template Parameters**

- **OperatorType**: - can be `LocalMatrix` or `GlobalMatrix`
- **VectorType**: - can be `LocalVector` or `GlobalVector`
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by rocalution::AIChebyshev< OperatorType, VectorType, ValueType >, rocalution::AS< OperatorType, VectorType, ValueType >, rocalution::BlockJacobi< OperatorType, VectorType, ValueType >, rocalution::BlockPreconditioner< OperatorType, VectorType, ValueType >, rocalution::DiagJacobiSaddlePointPrecond< OperatorType, VectorType, ValueType >, rocalution::FSAI< OperatorType, VectorType, ValueType >, rocalution::IC< OperatorType, VectorType, ValueType >, rocalution::ILU< OperatorType, VectorType, ValueType >, rocalution::ILUT< OperatorType, VectorType, ValueType >, rocalution::Jacobi< OperatorType, VectorType, ValueType >, rocalution::JacobiSaddlePointPrecond< OperatorType, VectorType, ValueType >, rocalution::MultiColored< OperatorType, VectorType, ValueType >, rocalution::MultiElimination< OperatorType, VectorType, ValueType >, rocalution::MultiPoint< OperatorType, VectorType, ValueType >, rocalution::Preconditioner< OperatorType, VectorType, ValueType >, rocalution::SGS< OperatorType, VectorType, ValueType >, rocalution::SPAI< OperatorType, VectorType, ValueType >, rocalution::TNS< OperatorType, VectorType, ValueType >, rocalution::VariablePreconditioner< OperatorType, VectorType, ValueType >

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class AIChebyshev : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>
```

Approximate Inverse - Chebyshev Preconditioner.

The Approximate Inverse - Chebyshev Preconditioner is an inverse matrix preconditioner with values from a linear combination of matrix-valued Chebyshev polynomials. cheb

**Template Parameters**

- **OperatorType**: - can be `LocalMatrix`
- **VectorType**: - can be `LocalVector`
- **ValueType**: - can be float, double, std::complex<float> or std::complex<double>

```cpp
void rocalution::AIChebyshev::Set (int p, ValueType lambda_min, ValueType lambda_max)
```

Set order, min and max eigenvalues.
class FSAI : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>

   Factorized Approximate Inverse Preconditioner.

   The Factorized Sparse Approximate Inverse preconditioner computes a direct approximation of \( M^{-1} \) by minimizing the Frobenius norm \( ||IGL||_F \), where \( L \) denotes the exact lower triangular part of \( A \) and \( G := M^{-1} \).
   The FSAI preconditioner is initialized by \( q \), based on the sparsity pattern of \( |A^q| \). However, it is also possible to supply external sparsity patterns in form of the LocalMatrix class. kolotilina

   Note The FSAI preconditioner is only suited for symmetric positive definite matrices.

   Template Parameters

   - OperatorType: - can be LocalMatrix
   - VectorType: - can be LocalVector
   - ValueType: - can be float, double, std::complex<float> or std::complex<double>

   Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::FSAI::Set” with arguments (int) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

   - void Set(const OperatorType &pattern)
   - void Set(int power)

   Warning: doxygenfunction: Unable to resolve multiple matches for function “rocalution::FSAI::Set” with arguments (const OperatorType&) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

   - void Set(const OperatorType &pattern)
   - void Set(int power)

void rocalution::FSAI::SetPrecondMatrixFormat (unsigned int mat_format)

Set the matrix format of the preconditioner.

template<class OperatorType, class VectorType, typename ValueType>
class SPAI : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>

   SParse Approximate Inverse Preconditioner.

   The SParse Approximate Inverse algorithm is an explicitly computed preconditioner for general sparse linear systems. In its current implementation, only the sparsity pattern of the system matrix is supported. The SPAI computation is based on the minimization of the Frobenius norm \( ||AMI||_F \). grote

   Template Parameters

   - OperatorType: - can be LocalMatrix
   - VectorType: - can be LocalVector
   - ValueType: - can be float, double, std::complex<float> or std::complex<double>

void rocalution::SPAI::SetPrecondMatrixFormat (unsigned int mat_format)

Set the matrix format of the preconditioner.

template<class OperatorType, class VectorType, typename ValueType>
class TNS : public roculation::Preconditioner<OperatorType, VectorType, ValueType>
Truncated Neumann Series Preconditioner.

The Truncated Neumann Series (TNS) preconditioner is based on \( M^{-1} = K^T D^{-1} K \), where \( K = (I - LD^{-1} + (LD^{-1})^2) \), with the diagonal \( D \) of \( A \) and the strictly lower triangular part \( L \) of \( A \). The preconditioner can be computed in two forms - explicitly and implicitly. In the implicit form, the full construction of \( M \) is performed via matrix-matrix operations, whereas in the explicit form, the application of the preconditioner is based on matrix-vector operations only. The matrix format for the stored matrices can be specified.

Template Parameters

- OperatorType: can be LocalMatrix
- VectorType: can be LocalVector
- ValueType: can be float, double, std::complex<float> or std::complex<double>

void roculation::TNS::Set (bool imp)
Set implicit (true) or explicit (false) computation.

void roculation::TNS::SetPrecondMatrixFormat (unsigned int mat_format)
Set the matrix format of the preconditioner.

template<class OperatorType, class VectorType, typename ValueType>
class AS : public roculation::Preconditioner<OperatorType, VectorType, ValueType>
Additive Schwarz Preconditioner.

The Additive Schwarz preconditioner relies on a preconditioning technique, where the linear system \( Ax = b \) can be decomposed into small sub-problems based on \( A_i = R_i^T A R_i \), where \( R_i \) are restriction operators. Those restriction operators produce sub-matrices which overlap. This leads to contributions from two preconditioners on the overlapped area which are scaled by 1/2. RAS

Template Parameters

- OperatorType: can be LocalMatrix
- VectorType: can be LocalVector
- ValueType: can be float, double, std::complex<float> or std::complex<double>

void roculation::AS::Set (int nb, int overlap, Solver<OperatorType, VectorType, ValueType> **preconds)
Set number of blocks, overlap and array of preconditioners.

template<class OperatorType, class VectorType, typename ValueType>
class RAS : public roculation::AS<OperatorType, VectorType, ValueType>
Restricted Additive Schwarz Preconditioner.

The Restricted Additive Schwarz preconditioner relies on a preconditioning technique, where the linear system \( Ax = b \) can be decomposed into small sub-problems based on \( A_i = R_i^T A R_i \), where \( R_i \) are restriction operators. The RAS method is a mixture of block Jacobi and the AS scheme. In this case, the sub-matrices contain overlapped areas from other blocks, too. RAS

Template Parameters

- OperatorType: can be LocalMatrix
- VectorType: can be LocalVector
- ValueType: can be float, double, std::complex<float> or std::complex<double>
template<class OperatorType, class VectorType, typename ValueType>
class BlockJacobi : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>
Block-Jacobi Preconditioner.

The Block-Jacobi preconditioner is designed to wrap any local preconditioner and apply it in a global block fashion locally on each interior matrix.

Template Parameters

- **OperatorType**: can be *GlobalMatrix*
- **VectorType**: can be *GlobalVector*
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

```cpp
void rocalution::BlockJacobi::Set(Solver<LocalMatrix<ValueType>, LocalVector<ValueType>, ValueType> &precond)
Set local preconditioner.
```

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class BlockPreconditioner : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>
Block-Preconditioner.

When handling vector fields, typically one can try to use different preconditioners and/or solvers for the different blocks. For such problems, the library provides a block-type preconditioner. This preconditioner builds the following block-type matrix

\[
P = \begin{pmatrix}
A_d & 0 & \cdots & 0 \\
B_1 & B_d & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots \\
Z_1 & Z_2 & \cdots & Z_d
\end{pmatrix}
\]

The solution of \( P \) can be performed in two ways. It can be solved by block-lower-triangular sweeps with inversion of the blocks \( A_d \ldots Z_d \) and with a multiplication of the corresponding blocks. This is set by *SetL-Solver()*. (which is the default solution scheme). Alternatively, it can be used only with an inverse of the diagonal \( A_d \ldots Z_d \) (Block-Jacobi type) by using *SetDiagonalSolver()*. 

Template Parameters

- **OperatorType**: can be *LocalMatrix*
- **VectorType**: can be *LocalVector*
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

```cpp
void rocalution::BlockPreconditioner::Set(int n, const int *size, Solver<OperatorType, VectorType, ValueType> **D_solver)
Set number, size and diagonal solver.
```

```cpp
void rocalution::BlockPreconditioner::SetDiagonalSolver(void)
Set diagonal solver mode.
```

```cpp
void rocalution::BlockPreconditioner::SetLSolver(void)
Set lower triangular sweep mode.
```

```cpp
void rocalution::BlockPreconditioner::SetExternalLastMatrix(const OperatorType &mat)
Set external last block matrix.
```

```cpp
void rocalution::BlockPreconditioner::SetPermutation(const LocalVector<int> &perm)
Set permutation vector.
```
template<class OperatorType, class VectorType, typename ValueType>
class Jacobi : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>

Jacobi Method.

The Jacobi method is for solving a diagonally dominant system of linear equations $Ax = b$. It solves for each diagonal element iteratively until convergence, such that

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)} \right)$$

Template Parameters

- **OperatorType**: can be LocalMatrix or GlobalMatrix
- **VectorType**: can be LocalVector or GlobalVector
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class GS : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>

Gauss-Seidel / Successive Over-Relaxation Method.

The Gauss-Seidel / SOR method is for solving system of linear equations $Ax = b$. It approximates the solution iteratively with

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)} \right),$$

with $\omega \in (0, 2)$.

Template Parameters

- **OperatorType**: can be LocalMatrix
- **VectorType**: can be LocalVector
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class SGS : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>


The Symmetric Gauss-Seidel / SSOR method is for solving system of linear equations $Ax = b$. It approximates the solution iteratively.

Template Parameters

- **OperatorType**: can be LocalMatrix
- **VectorType**: can be LocalVector
- **ValueType**: can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class ILU : public rocalution::Preconditioner<OperatorType, VectorType, ValueType>

Incomplete LU Factorization based on levels.

The Incomplete LU Factorization based on levels computes a sparse lower and sparse upper triangular matrix such that $A = LU - R$. 

2.12. ROCm Libraries
Template Parameters

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, `std::complex<float>` or `std::complex<double>`

```cpp
void rocalution::ILU::Set(int p, bool level = true)
Initialize ILU(p) factorization.
```

Initialize ILU(p) factorization based on power. SAAD
- level = true build the structure based on levels
- level = false build the structure only based on the power(p+1)

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class ILUT: public rocalution::Preconditioner<OperatorType, VectorType, ValueType>
Incomplete LU Factorization based on threshold.
```

The Incomplete LU Factorization based on threshold computes a sparse lower and sparse upper triangular matrix such that $A = LU - R$. Fill-in values are dropped depending on a threshold and number of maximal fill-ins per row. SAAD

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
- **VectorType**: can be `LocalVector`
- **ValueType**: can be float, double, `std::complex<float>` or `std::complex<double>`

**Warning**: doxygenfunction: Unable to resolve multiple matches for function “rocalution::ILUT::Set” with arguments (double) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/.
Potential matches:

- void Set(double t)
- void Set(double t, int maxrow)

**Warning**: doxygenfunction: Unable to resolve multiple matches for function “rocalution::ILUT::Set” with arguments (double, int) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/.
Potential matches:

- void Set(double t)
- void Set(double t, int maxrow)

```cpp
template<class OperatorType, class VectorType, typename ValueType>
class IC: public rocalution::Preconditioner<OperatorType, VectorType, ValueType>
Incomplete Cholesky Factorization without fill-ins.
```

The Incomplete Cholesky Factorization computes a sparse lower triangular matrix such that $A = LL^T - R$. Additional fill-ins are dropped and the sparsity pattern of the original matrix is preserved.

**Template Parameters**

- **OperatorType**: can be `LocalMatrix`
• VectorType: can be \textit{LocalVector}
• ValueType: can be float, double, std::complex<float> or std::complex<double>

\begin{verbatim}
template<class OperatorType, class VectorType, typename ValueType>
class VariablePreconditioner : public roculation::Preconditioner<OperatorType, VectorType, ValueType>

Variable Preconditioner can hold a selection of preconditioners. Thus, any type of preconditioners can be combined. As example, the variable preconditioner can combine Jacobi, GS and ILU – then, the first iteration of the iterative solver will apply Jacobi, the second iteration will apply GS and the third iteration will apply ILU. After that, the solver will start again with Jacobi, GS, ILU.

Template Parameters

• OperatorType: can be LocalMatrix
• VectorType: can be LocalVector
• ValueType: can be float, double, std::complex<float> or std::complex<double>

void roculation::VariablePreconditioner::SetPreconditioner(int n, Solver<OperatorType, VectorType, ValueType> **precond)

Set the preconditioner sequence.
\end{verbatim}

\begin{verbatim}
template<class OperatorType, class VectorType, typename ValueType>
class MultiColored : public roculation::Preconditioner<OperatorType, VectorType, ValueType>

Base class for all multi-colored preconditioners.

Template Parameters

• OperatorType: can be LocalMatrix
• VectorType: can be LocalVector
• ValueType: can be float, double, std::complex<float> or std::complex<double>

Subclassed by roculation::MultiColoredILU< OperatorType, VectorType, ValueType >, roculation::MultiColoredSGS< OperatorType, VectorType, ValueType >

void roculation::MultiColored::SetPrecondMatrixFormat(unsigned int mat_format)
Set a specific matrix type of the decomposed block matrices.

void roculation::MultiColored::SetDecomposition(bool decomp)
Set if the preconditioner should be decomposed or not.
\end{verbatim}

\begin{verbatim}
template<class OperatorType, class VectorType, typename ValueType>
class MultiColoredSGS : public roculation::Preconditioner<OperatorType, VectorType, ValueType>

Multi-Colored Symmetric Gauss-Seidel / SSOR Preconditioner.

The Multi-Colored Symmetric Gauss-Seidel / SSOR preconditioner is based on the splitting of the original matrix. Higher parallelism in solving the forward and backward substitution is obtained by performing a multi-colored decomposition. Details on the Symmetric Gauss-Seidel / SSOR algorithm can be found in the SGS preconditioner.

Template Parameters

• OperatorType: can be LocalMatrix
• VectorType: can be LocalVector
\end{verbatim}
ValueType: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by rocalution::MultiColoredGS< OperatorType, VectorType, ValueType >

void rocalution::MultiColoredSGS::SetRelaxation (ValueType omega)
Set the relaxation parameter for the SOR/SSOR scheme.

template<class OperatorType, class VectorType, typename ValueType>
class MultiColoredGS: public rocalution::MultiColoredSGS<OperatorType, VectorType, ValueType>
Multi-Colored Gauss-Seidel / SOR Preconditioner.

The Multi-Colored Symmetric Gauss-Seidel / SOR preconditioner is based on the splitting of the original matrix. Higher parallelism in solving the forward substitution is obtained by performing a multi-colored decomposition. Details on the Gauss-Seidel / SOR algorithm can be found in the GS preconditioner.

Template Parameters
- OperatorType: - can be LocalMatrix
- VectorType: - can be LocalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

template<class OperatorType, class VectorType, typename ValueType>
class MultiColoredILU: public rocalution::MultiColored<OperatorType, VectorType, ValueType>
Multi-Colored Incomplete LU Factorization Preconditioner.

Multi-Colored Incomplete LU Factorization based on the ILU(p) factorization with a power(q)-pattern method. This method provides a higher degree of parallelism of forward and backward substitution compared to the standard ILU(p) preconditioner. Lukarski2012

Template Parameters
- OperatorType: - can be LocalMatrix
- VectorType: - can be LocalVector
- ValueType: - can be float, double, std::complex<float> or std::complex<double>

---

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::MultiColoredILU::Set” with arguments (int) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void Set(int p)
- void Set(int p, int q, bool level = true)

---

**Warning:** doxygenfunction: Unable to resolve multiple matches for function “rocalution::MultiColoredILU::Set” with arguments (int, int, bool) in doxygen xml output for project “rocALUTION” from directory: rocALUTIONxml/. Potential matches:

- void Set(int p)
- void Set(int p, int q, bool level = true)
class MultiElimination: public roc::Preconditioner<OperatorType, VectorType, ValueType>

Multi-Elimination Incomplete LU Factorization Preconditioner.

The Multi-Elimination Incomplete LU preconditioner is based on the following decomposition

\[
A = \begin{pmatrix}
    D & E \\
    E & C
\end{pmatrix} = \begin{pmatrix}
    I & 0 \\
    ED^{-1} & I
\end{pmatrix} \times \begin{pmatrix}
    D & F \\
    0 & \hat{A}
\end{pmatrix},
\]

where \( \hat{A} = C - ED^{-1}F \). To make the inversion of \( D \) easier, we permute the preconditioning before the factorization with a permutation \( P \) to obtain only diagonal elements in \( D \). The permutation here is based on a maximal independent set. This procedure can be applied to the block matrix \( \hat{A} \), in this way we can perform the factorization recursively. In the last level of the recursion, we need to provide a solution procedure. By the design of the library, this can be any kind of solver. SAAD

Template Parameters

- OperatorType: can be LocalMatrix
- VectorType: can be LocalVector
- ValueType: can be float, double, std::complex<float> or std::complex<double>

int roc::MultiElimination::GetSizeDiagBlock (void) const

Returns the size of the first (diagonal) block of the preconditioner.

int roc::MultiElimination::GetLevel (void) const

Return the depth of the current level.

void roc::MultiElimination::Set (Solver<OperatorType, VectorType, ValueType> &AA_Solver, int level, double drop_off = 0.0)

Initialize (recursively) ME-ILU with level (depth of recursion)

AA_Solvers - defines the last-block solver drop_off - defines drop-off tolerance

void roc::MultiElimination::SetPrecondMatrixFormat (unsigned int mat_format)

Set a specific matrix type of the decomposed block matrices.

template<class OperatorType, class VectorType, typename ValueType>
class DiagJacobiSaddlePointPrecond: public roc::Preconditioner<OperatorType, VectorType, ValueType>

Diagonal Preconditioner for Saddle-Point Problems.

Consider the following saddle-point problem

\[
A = \begin{pmatrix}
    K & F \\
    E & 0
\end{pmatrix}.
\]

For such problems we can construct a diagonal Jacobi-type preconditioner of type

\[
P = \begin{pmatrix}
    K & 0 \\
    0 & S
\end{pmatrix},
\]

with \( S = ED^{-1}F \), where \( D \) are the diagonal elements of \( K \). The matrix \( S \) is fully constructed (via sparse matrix-matrix multiplication). The preconditioner needs to be initialized with two external solvers/preconditioners - one for the matrix \( K \) and one for the matrix \( S \).

Template Parameters

- OperatorType: can be LocalMatrix
- VectorType: can be LocalVector
- ValueType: can be float, double, std::complex<float> or std::complex<double>
void rocakution::DiagJacobiSaddlePointPrecond::Set(Solver<OperatorType, VectorType, ValueType> &K_Solver, Solver<OperatorType, VectorType, ValueType> &S_Solver)

Initialize solver for \( K \) and \( S \).

2.12.9 Tensile

2.12.9.1 Introduction

Tensile is a tool for creating a benchmark-driven backend library for GEMMs, GEMM-like problems (such as batched GEMM), N-dimensional tensor contractions, and anything else that multiplies two multi-dimensional objects together on an AMD GPU.

Overview for creating a custom TensileLib backend library for your application:

1. Install the PyYAML and cmake dependency (mandatory), git clone and cd Tensile
2. Create a benchmark config.yaml file in ./Tensile/Configs/
3. Run the benchmark. After the benchmark is finished. Tensile will dump 4 directories: 1 & 2 is about benchmarking. 3 & 4 is the summarized results from your library (like rocBLAS) viewpoints.
   1. 1_BenchmarkProblems: has all the problems descriptions and executables generated during benchmarking, where you can re-launch exe to reproduce results.
   2. 2_BenchmarkData: has the raw performance results.
   3. 3_LibraryLogic: has optimal kernel configurations yaml file and Winner*.csv. Usually rocBLAS takes the yaml files from this folder.
   4. 4_LibraryClient: has a client exe, so you can launch from a library viewpoint.
4. Add the Tensile library to your application’s CMake target. The Tensile library will be written, compiled and linked to your application at application-compile-time.
   - GPU kernels, written in HIP, OpenCL, or AMD GCN assembly.
   - Solution classes which enqueue the kernels.
   - APIs which call the fastest solution for a problem.

2.12.9.1.1 Quick Example (Ubuntu):

```
sudo apt-get install python-yaml
mkdir Tensile
cd Tensile
git clone https://github.com/ROCmSoftwarePlatform/Tensile repo
cd repo
git checkout master
mkdir build
cd build
python ../Tensile/Tensile.py ../Tensile/Configs/test_sgemm.yaml ./
```

After about 10 minutes of benchmarking, Tensile will print out the path to the client you can run.

```
./4_LibraryClient/build/client -h
./4_LibraryClient/build/client --sizes 5760 5760 1 5760
```
2.12.9.2 Benchmark Config example

Tensile uses an incremental and “programmable” benchmarking protocol.

2.12.9.2.1 Example Benchmark config.yaml as input file to Tensile

GlobalParameters:
- PrintLevel: 1
- ForceRedoBenchmarkProblems: False
- ForceRedoLibraryLogic: True
- ForceRedoLibraryClient: True
- CMakeBuildType: Release
- EnqueuesPerSync: 1
- SyncsPerBenchmark: 1
- LibraryPrintDebug: False
- NumElementsToValidate: 128
- ValidationMaxToPrint: 16
- ValidationPrintValids: False
- ShortNames: False
- MergeFiles: True
- PlatformIdx: 0
- DeviceIdx: 0
- DataInitTypeAB: 0

BenchmarkProblems:
- # sgemm NN
  - # ProblemType
    - OperationType: GEMM
    - DataType: s
    - TransposeA: False
    - TransposeB: False
    - UseBeta: True
    - Batched: True
  - # BenchmarkProblemSizeGroup
    - InitialSolutionParameters:
      - BenchmarkCommonParameters:
        - ProblemSizes:
          - Range: [ [5760], 0, [1], 0 ]
          - LoopDoWhile: False
          - NumLoadsCoalescedA: [-1]
          - NumLoadsCoalescedB: [1]
          - WorkGroupMapping: [1]
        - ForkParameters:
          - ThreadTile:
            - [ 8, 8 ]
            - [ 4, 8 ]
            - [ 4, 4 ]
          - WorkGroup:
            - [ 8, 16, 1 ]
            - [ 16, 16, 1 ]
          - LoopTail: [False, True]
          - EdgeType: ["None", "Branch", "ShiftPtr"]
          - DepthU: [ 8, 16]
          - VectorWidth: [1, 2, 4]
        - BenchmarkForkParameters:
JoinParameters:
  - MacroTile
BenchmarkJoinParameters:
BenchmarkFinalParameters:
  - ProblemSizes:
    - Range: [5760], 0, [1], 0

LibraryLogic:

LibraryClient:

2.12.9.2.2 Structure of config.yaml

Top level data structure whose keys are Parameters, BenchmarkProblems, LibraryLogic and LibraryClient.

- **Parameters** contains a dictionary storing global parameters used for all parts of the benchmarking.

- **BenchmarkProblems** contains a list of dictionaries representing the benchmarks to conduct; each element, i.e. dictionary, in the list is for benchmarking a single **ProblemType**. The keys for these dictionaries are **ProblemType**, **InitialSolutionParameters**, **BenchmarkCommonParameters**, **ForkParameters**, **BenchmarkForkParameters**, **JoinParameters**, **BenchmarkJoinParameters** and **BenchmarkFinalParameters**. See **Benchmark Protocol** for more information on these steps.

- **LibraryLogic** contains a dictionary storing parameters for analyzing the benchmark data and designing how the backend library will select which Solution for certain ProblemSizes.

- **LibraryClient** contains a dictionary storing parameters for actually creating the library and creating a client which calls into the library.

2.12.9.2.3 Global Parameters

- **Name**: Prefix to add to API function names; typically name of device.

- **MinimumRequiredVersion**: Which version of Tensile is required to interpret this yaml file

- **RuntimeLanguage**: Use HIP or OpenCL runtime.

- **KernelLanguage**: For OpenCL runtime, kernel language must be set to OpenCL. For HIP runtime, kernel language can be set to HIP or assembly (gfx803, gfx900).

- **PrintLevel**: 0=Tensile prints nothing, 1=prints some, 2=prints a lot.

- **ForceRedoBenchmarkProblems**: False means don’t redo a benchmark phase if results for it already exist.

- **ForceRedoLibraryLogic**: False means don’t re-generate library logic if it already exist.

- **ForceRedoLibraryClient**: False means don’t re-generate library client if it already exist.

- **CMakeBuildType**: Release or Debug

- **EnqueuesPerSync**: Num enqueues before syncing the queue.

- **SyncsPerBenchmark**: Num queue syncs for each problem size.

- **LibraryPrintDebug**: True means Tensile solutions will print kernel enqueue info to stdout

- **NumElementsToValidate**: Number of elements to validate; 0 means no validation.

- **ValidationMaxToPrint**: How many invalid results to print.
- **ValidationPrintValids**: True means print validation comparisons that are valid, not just invalids.
- **ShortNames**: Convert long kernel, solution and files names to short serial ids.
- **MergeFiles**: False means write each solution and kernel to its own file.
- **PlatformIdx**: OpenCL platform id.
- **DeviceIdx**: OpenCL or HIP device id.
- **DataInitType[AB,C]**: Initialize validation data with 0=0’s, 1=1’s, 2=serial, 3=random.
- **KernelTime**: Use kernel time reported from runtime rather than api times from cpu clocks to compare kernel performance.

The exhaustive list of global parameters and their defaults is stored in Common.py.

### 2.12.9.2.4 Problem Type Parameters

- **OperationType**: GEMM or TensorContraction.
- **DataType**: s, d, c, z, h
- **UseBeta**: False means library/solutions/kernel won’t accept a beta parameter; thus beta=0.
- **UseInitialStrides**: False means data is contiguous in memory.
- **HighPrecisionAccumulate**: For tmpC += a*b, use twice the precision for tmpC as for DataType. Not yet implemented.
- **ComplexConjugateA**: True or False; ignored for real precision.
- **ComplexConjugateB**: True or False; ignored for real precision.

For **OperationType=GEMM** only:

- **TransposeA**: True or False.
- **TransposeB**: True or False.
- **Batched**: True (False has been deprecated). For **OperationType=TensorContraction** only (showing batched gemm NT: C[ijk] = Sum[l] A[ilk] * B[jlk])
- **IndexAssignmentsA**: [0, 3, 2]
- **IndexAssignmentsB**: [1, 3, 2]
- **NumDimensionsC**: 3.

### 2.12.9.2.5 Solution / Kernel Parameters

See: Kernel Parameters.
2.12.9.2.6 Defaults

Because of the flexibility / complexity of the benchmarking process and, therefore, of the config.yaml files; Tensile has a default value for every parameter. If you neglect to put LoopUnroll anywhere in your benchmark, rather than crashing or complaining, Tensile will put the default LoopUnroll options into the default phase (common, fork, join...). This guarantees ease of use and more importantly backward compatibility; every time we add a new possible solution parameter, you don’t necessarily need to update your configs; we’ll have a default figured out for you.

However, this may cause some confusion. If your config fork 2 parameters, but you see that 3 were forked during benchmarking, that’s because you didn’t specify the 3rd parameter anywhere, so Tensile stuck it in its default phase, which was forking (for example). Also, specifying ForkParameters: and leaving it empty isn’t the same as leaving JoinParameter out of your config. If you leave ForkParameters out of your config, Tensile will add a ForkParameters step and put the default parameters into it (unless you put all the parameters elsewhere), but if you specify ForkParameters and leave it empty, then you won’t work anything.

Therefore, it is safest to specify all parameters in your config.yaml files; that way you’ll guarantee the behavior you want. See /Tensile/Common.py for the current list of parameters.

2.12.9.3 Benchmark Protocol

2.12.9.3.1 Old Benchmark Architecture was Intractable

The benchmarking strategy from version 1 was vanilla flavored brute force: (8 WorkGroups)* (12 ThreadTiles)* (4 NumLoadsCoalescedAs)* (4 NumLoadsCoalescedBs)* (3 LoopUnrolls)* (5 BranchTypes)* ... *(1024 ProblemSizes)=23,592,960 is a multiplicative series which grows very quickly. Adding one more boolean parameter doubles the number of kernel enqueues of the benchmark.

2.12.9.3.2 Incremental Benchmark is Faster

Tensile version 2 allows the user to manually interrupt the multiplicative series with “additions” instead of “multiplies”, i.e., (8 WorkGroups)* (12 ThreadTiles)+ (4 NumLoadsCoalescedAs)* (4 NumLoadsCoalescedBs)* (3 LoopUnrolls)+ (5 BranchTypes)* ... +(1024 ProblemSizes)=1,151 is a dramatically smaller number of enqueues. Now, adding one more boolean parameter may only add on 2 more enqueues.

2.12.9.3.3 Phases of Benchmark

To make the Tensile’s programability more manageable for the user and developer, the benchmarking protocol has been split up into several steps encoded in a config.yaml file. The below sections reference the following config.yaml. Note that this config.yaml has been created to be a simple illustration and doesn’t not represent an actual good benchmark protocol. See the configs included in the repository (/Tensile/Configs) for examples of good benchmarking configs.

BenchmarkProblems:
- # sgemm
  - # Problem Type
    OperationType: GEMM
    Batched: True
  - # Benchmark Size-Group
    InitialSolutionParameters:
    - WorkGroup: [ [ 16, 16, 1 ] ]
    - NumLoadsCoalescedA: [ 1 ]
    - NumLoadsCoalescedB: [ 1 ]

(continues on next page)
- ThreadTile: [[4, 4]]

BenchmarkCommonParameters:
  - ProblemSizes:
    - Range: [[512], [512], [1], [512]]
    - EdgeType: ["Branch", "ShiftPtr"]
      PrefetchGlobalRead: [False, True]

ForkParameters:
  - WorkGroup: [[8, 32, 1], [16, 16, 1], [32, 8, 1]]
    ThreadTile: [[2, 8], [4, 4], [8, 2]]

BenchmarkForkParameters:
  - ProblemSizes:
    - Exact: [2880, 2880, 1, 2880]
    - NumLoadsCoalescedA: [1, 2, 4, 8]
    - NumLoadsCoalescedB: [1, 2, 4, 8]

JoinParameters:
  - MacroTile

BenchmarkJoinParameters:
  - LoopUnroll: [8, 16]

BenchmarkFinalParameters:
  - ProblemSizes:
    - Range: [[16, 128], [16, 128], [1], [256]]

2.12.9.3.4 Initial Solution Parameters

A Solution is comprised of ~20 parameters, and all are needed to create a kernel. Therefore, during the first benchmark which determines which WorkGroupShape is fastest, what are the other 19 solution parameters which are used to describe the kernels that we benchmark? That’s what InitialSolutionParameters are for. The solution used for benchmarking WorkGroupShape will use the parameters from InitialSolutionParameters. The user must choose good default solution parameters in order to correctly identify subsequent optimal parameters.

2.12.9.3.5 Problem Sizes

Each step of the benchmark can override what problem sizes will be benchmarked. A ProblemSizes entry of type Range is a list whose length is the number of indices in the ProblemType. A GEMM ProblemSizes must have 3 elements while a batched-GEMM ProblemSizes must have 4 elements. So, for a ProblemType of C[ij] = Sum[k] A[ik]*B[jk], the ProblemSizes elements represent [SizeI, SizeJ, SizeK]. For each index, there are 5 ways of specifying the sizes of that index:

1. [1968]
   - Benchmark only size 1968; n = 1.
2. [16, 1920]
   - Benchmark sizes 16 to 1968 using the default step size (=16); n = 123.
3. [16, 32, 1968]
   - Benchmark sizes 16 to 1968 using a step size of 32; n = 61.
4. [64, 32, 16, 1968]
   • Benchmark sizes from 64 to 1968 with a step size of 32. Also, increase the step size by 16 each iteration.
   • This causes fewer sizes to be benchmarked when the sizes are large, and more benchmarks where the sizes are small; this is typically desired behavior.
   • $n = 16$ (64, 96, 144, 208, 288, 384, 496, 624, 768, 928, 1104, 1296, 1504, 1728, 1968). The stride at the beginning is 32, but the stride at the end is 256.

5. 0
   • The size of this index is just whatever size index 0 is. For a 3-dimensional ProblemType, this allows benchmarking only a 2- dimensional or 1-dimensional slice of problem sizes.

Here are a few examples of valid ProblemSizes for 3D GEMMs:

<table>
<thead>
<tr>
<th>Range</th>
<th># n = 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ [16, 128], [16, 128], [16, 128] ]</td>
<td></td>
</tr>
<tr>
<td>[ [16, 128], 0, 0]</td>
<td># n = 8</td>
</tr>
<tr>
<td>[ [16, 16, 16, 5760], 0, [1024, 1024, 4096] ]</td>
<td># n = 108</td>
</tr>
</tbody>
</table>

2.12.9.3.6 Benchmark Parameters

During this first phase of benchmarking, we examine parameters which will be the same for all solutions for this ProblemType. During each step of benchmarking, there is only 1 winner. In the above example we are benchmarking the dictionary \{EdgeType: [Branch, ShiftPtr], PrefetchGlobalRead: [False, True]\}; therefore, this benchmark step generates 4 solution candidates, and the winner will be the fastest EdgeType/PrefetchGlobalRead combination. Assuming the winner is ET=SP and PGR=T, then all solutions for this ProblemType will have ET=SP and PGR=T. Also, once a parameter has been determined, all subsequent benchmarking steps will use this determined parameter rather than pulling values from InitialSolutionParameters. Because the common parameters will apply to all kernels, they are typically the parameters which are compiler-dependent or hardware-dependent rather than being tile-dependent.

2.12.9.3.7 Fork Parameters

If we continued to determine every parameter in the above manner, we’d end up with a single fastest solution for the specified ProblemSizes; we usually desire multiple different solutions with varying parameters which may be fastest for different groups of ProblemSizes. One simple example of this is small tiles sizes are fastest for small problem sizes, and large tiles are fastest for large tile sizes.

Therefore, we allow “forking” parameters; this means keeping multiple winners after each benchmark steps. In the above example we fork \{WorkGroup: […], ThreadTile: […]\}. This means that in subsequent benchmarking steps, rather than having one winning parameter, we’ll have one winning parameter per fork permutation; we’ll have 9 winners.
2.12.9.3.8 Benchmark Fork Parameters

When we benchmark the fork parameters, we retain one winner per permutation. Therefore, we first determine the fastest NumLoadsCoalescedA for each of the WG,TT permutations, then we determine the fastest NumLoadsCoalescedB for each permutation.

2.12.9.3.9 Join Parameters

After determining fastest parameters for all the forked solution permutations, we have the option of reducing the number of winning solutions. When a parameter is listed in the JoinParameters section, that means that of the kept winning solutions, each will have a different value for that parameter. Listing more parameters to join results in more winners being kept, while having a JoinParameters section with no parameters listed results on only 1 fastest solution.

In our example we join over the MacroTile (work-group x thread-tile). After forking tiles, there were 9 solutions that we kept. After joining MacroTile, we’ll only keep six: 16x256, 32x128, 64x64, 128x32 and 256x16. The solutions that are kept are based on their performance during the last BenchmarkForkParameters benchmark, or, if there weren’t any, JoinParameters will conduct a benchmark of all solution candidates then choose the fastest.

2.12.9.3.10 Benchmark Join Parameters

After narrowing the list of fastest solutions through joining, you can continue to benchmark parameters, keeping one winning parameter per solution permutation.

2.12.9.3.11 Benchmark Final Parameters

After all the parameter benchmarking has been completed and the final list of fastest solution has been assembled, we can benchmark all the solution over a large set of ProblemSizes. This benchmark represent the final output of benchmarking; it outputs a .csv file where the rows are all the problem sizes and the columns are all the solutions. This is the information which gets analysed to produce the library logic.

2.12.9.4 Contributing

We’d love your help, but . . .

1. Never check in a tab (t); use 4 spaces.
2. Follow the coding style of the file you’re editing.
3. Make pull requests against develop branch.
4. Rebase your develop branch against ROCmSoftwarePlatform::Tensile::develop branch right before pull-requesting.
5. In your pull request, state what you tested (which OS, what drivers, what devices, which config.yaml’s) so we can ensure that your changes haven’t broken anything.
2.12.9.5 Dependencies

2.12.9.5.1 CMake

- CMake 2.9

2.12.9.5.2 Python

(One time only)

- Ubuntu: sudo apt install python2.7 python-yaml
- CentOS: sudo yum install python PyYAML
- Fedora: sudo dnf install python PyYAML

2.12.9.5.3 Compilers

- For Tensile_BACKEND = OpenCL1.2 (untested)
  - Visual Studio 14 (2015). (VS 2012 may also be supported; c++11 should no longer be required by Tensile. Need to verify.)
  - GCC 4.8 and above
- For Tensile_BACKEND = HIP
  - Public ROCm

2.12.9.6 Installation

Tensile can be installed via:

1. Download repo and don’t install; install PyYAML dependency manually and call python scripts manually:

   ```
   git clone https://github.com/ROCmSoftwarePlatform/Tensile.git
   python Tensile/Tensile/Tensile.py your_custom_config.yaml your_benchmark_path
   ```

2. Install develop branch directly from repo using pip:

   ```
   pip install git+https://github.com/ROCmSoftwarePlatform/Tensile.git@develop
tensile your_custom_config.yaml your_benchmark_path
   ```

3. Download repo and install manually: (deprecated)

   ```
   git clone https://github.com/ROCmSoftwarePlatform/Tensile.git
   cd Tensile
   sudo python setup.py install
tensile your_custom_config.yaml your_benchmark_path
   ```
2.12.9.7 Kernel Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoopDoWhile</td>
<td>True=DoWhile loop, False=While or For loop</td>
</tr>
<tr>
<td>LoopTail</td>
<td>Additional loop with LoopUnroll=1.</td>
</tr>
<tr>
<td>EdgeType</td>
<td>Branch, ShiftPtr or None</td>
</tr>
<tr>
<td>WorkGroup</td>
<td>[dim0, dim1, LocalSplitU]</td>
</tr>
<tr>
<td>ThreadTile</td>
<td>[dim0, dim1]</td>
</tr>
<tr>
<td>GlobalSplitU</td>
<td>Split up summation among work-groups to create more concurrency. This option launches a kernel to handle the beta scaling, then a second kernel where the writes to global memory are atomic.</td>
</tr>
<tr>
<td>PrefetchGlobalRead</td>
<td>True means outer loop should prefetch global data one iteration ahead.</td>
</tr>
<tr>
<td>PrefetchLocalRead</td>
<td>True means inner loop should prefetch lds data one iteration ahead.</td>
</tr>
<tr>
<td>WorkGroupMapping</td>
<td>In what order will work-groups compute C; affects caching.</td>
</tr>
<tr>
<td>LoopUnroll</td>
<td>How many iterations to unroll inner loop; helps loading coalesced memory.</td>
</tr>
<tr>
<td>MacroTile</td>
<td>Derived from WorkGroup*ThreadTile.</td>
</tr>
<tr>
<td>DepthU</td>
<td>Derived from LoopUnroll*SplitU.</td>
</tr>
<tr>
<td>NumLoadsCoalescedA,B</td>
<td>Number of loads from A in coalesced dimension.</td>
</tr>
<tr>
<td>GlobalReadCoalesceGroupA,B</td>
<td>True means adjacent threads map to adjacent global read elements (but, if transposing data then write to lds is scattered).</td>
</tr>
<tr>
<td>GlobalReadCoalesceVectorA,B</td>
<td>True means vector components map to adjacent global read elements (but, if transposing data then write to lds is scattered).</td>
</tr>
<tr>
<td>VectorWidth</td>
<td>Thread tile elements are contiguous for faster memory accesses. For example VW=4 means a thread will read a float4 from memory rather than 4 non-contiguous floats.</td>
</tr>
<tr>
<td>KernelLanguage</td>
<td>Whether kernels should be written in source code (HIP, OpenCL) or assembly (gfx803, gfx900, ...).</td>
</tr>
</tbody>
</table>

The exhaustive list of solution parameters and their defaults is stored in Common.py.

2.12.9.7.2 Kernel Parameters Affect Performance

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Affect</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>The kernel parameters affect many aspects of performance. Changing a parameter may help address one performance bottleneck but worsen another. That is why searching through the parameter space is vital to discovering the fastest kernel for a given problem.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.12.9.7.3 How N-Dimensional Tensor Contractions Are Mapped to Finite-Dimensional GPU Kernels

For a traditional GEMM, the 2-dimensional output, \( C[i,j] \), is mapped to launching a 2-dimensional grid of work groups, each of which has a 2-dimensional grid of work items; one dimension belongs to \( i \) and one dimension belongs to \( j \). The 1-dimensional summation is represented by a single loop within the kernel body.

2.12.9.7.4 Special Dimensions: D0, D1 and DU

To handle arbitrary dimensionality, Tensile begins by determining 3 special dimensions: D0, D1 and DU.

D0 and D1 are the free indices of A and B (one belongs to A and one to B) which have the shortest strides. This allows the inner-most loops to read from A and B the fastest via coalescing. In a traditional GEMM, every matrix has a dimension with a shortest stride of 1, but Tensile doesn’t make that assumption. Of these two dimensions, D0 is the dimension which has the shortest tensor C stride which allows for fast writing.

DU represents the summation index with the shortest combined stride (stride in A + stride in B); it becomes the inner most loop which gets “U”nrolled. This assignment is also mean’t to assure fast reading in the inner-most summation loop. There can be multiple summation indices (i.e. embedded loops) and DU will be iterated over in the inner most loop.
2.12.9.5 GPU

OpenCL allows for 3-dimensional grid of work-groups, and each work-group can be a 3-dimensional grid of
work-items. Tensile assigns D0 to be dimension-0 of the work-group and work-item grid; it assigns D1 to be
dimension-1 of the work-group and work-item grids. All other free or batch dimensions are flattened down into the
final dimension-2 of the work-group and work-item grids. Within the GPU kernel, dimensions-2 is reconstituted
back into whatever dimensions it represents.

2.12.9.8 Languages

2.12.9.8.1 Tensile Benchmarking is Python3

The benchmarking module, Tensile.py, is written in python3. The python scripts write solution, kernels, cmake files
and all other C/C++ files used for benchmarking. Please note that Tensile is not compatible with Python2.

2.12.9.8.2 Tensile Library

The Tensile API, Tensile.h, is confined to C89 so that it will be usable by most software. The code behind the API is
allowed to be c++11.

2.12.9.8.3 Device Languages

The device languages Tensile supports for the gpu kernels is

- OpenCL 1.2
- HIP
- Assembly
  - gfx803
  - gfx900

2.12.9.9 Library Logic

Running the LibraryLogic phase of benchmarking analyses the benchmark data and encodes a mapping for each
problem type. For each problem type, it maps problem sizes to best solution (i.e. kernel).

When you build Tensile.lib, you point the TensileCreateLibrary function to a directory where your library
logic yaml files are.

2.12.9.10 Problem Nomenclature

2.12.9.10.1 Example Problems

- Standard GEMM has 4 variants (2 free indices (i, j) and 1 summation index l)

- \( C[i,j,k] = \text{Sum}[l] A[i,l,k] \times B[j,l,k] \) (batched-GEMM; 2 free indices, 1 batched index \( k \) and 1 summation index \( l \))
- \( C[i,j] = \text{Sum}[k,l] A[i,k,l] \times B[j,l,k] \) (2D summation)
- \( C[i,j,k,l,m] = \text{Sum}[n] A[i,k,m,l,n] \times B[j,k,l,n,m] \) (GEMM with 3 batched indices)
- \( C[i,j,k,l,m] = \text{Sum}[n,o] A[i,k,m,o,n] \times B[j,m,l,n,o] \) (4 free indices, 2 summation indices and 1 batched index)
- \( C[i,j,k,l] = \text{Sum}[m,n] A[i,j,m,n,l] \times B[m,n,k,j,l] \) (batched image convolution mapped to 7D tensor contraction)
- and even crazier

2.12.9.10.2 **Nomenclature**

The indices describe the dimensionality of the problem being solved. A GEMM operation takes 2 2-dimensional matrices as input (totaling 4 input dimensions) and contracts them along one dimension (which cancels out 2 of the dimensions), resulting in a 2-dimensional result.

Whenever an index shows up in multiple tensors, those tensors must be the same size along that dimension but they may have different strides.

There are 3 categories of indices/dimensions that Tensile deals with: free, batch and bound.

**Free Indices**

Free indices are the indices of tensor \( C \) which come in pairs; one of the pair shows up in tensor \( A \) while the other shows up in tensor \( B \). In the really crazy example above, \( i/j/k/l \) are the 4 free indices of tensor \( C \). Indices \( i \) and \( k \) come from tensor \( A \) and indices \( j \) and \( l \) come from tensor \( B \).

**Batch Indices**

Batch indices are the indices of tensor \( C \) which shows up in both tensor \( A \) and tensor \( B \). For example, the difference between the GEMM example and the batched-GEMM example above is the additional index. In the batched-GEMM example, the index \( K \) is the batch index which is batching together multiple independent GEMMs.

**Bound/Summation Indices**

The final type of indices are called bound indices or summation indices. These indices do not show up in tensor \( C \); they show up in the summation symbol (\( \text{Sum}[k] \)) and in tensors \( A \) and \( B \). It is along these indices that we perform the inner products (pairwise multiply then sum).

2.12.9.10.3 **Limitations**

Problem supported by Tensile must meet the following conditions:

There must be at least one pair of free indices.
2.12.9.11 Tensile.lib

After running the benchmark and generating library config files, you’re ready to add Tensile.lib to your project. Tensile provides a TensileCreateLibrary function, which can be called:

```c
set(Tensile_BACKEND "HIP")
set( Tensile_LOGIC_PATH "~/LibraryLogic" CACHE STRING "Path to Tensile logic.yaml files")
option( Tensile_MERGE_FILES "Tensile to merge kernels and solutions files?" OFF)
option( Tensile_SHORT_NAMES "Tensile to use short file/function names? Use if compiler complains they're too long." OFF)
option( Tensile_PRINT_DEBUG "Tensile to print runtime debug info?" OFF)

find_package(Tensile) # use if Tensile has been installed
TensileCreateLibrary(
   ${Tensile_LOGIC_PATH}
   ${Tensile_BACKEND}
   ${Tensile_MERGE_FILES}
   ${Tensile_SHORT_NAMES}
   ${Tensile_PRINT_DEBUG}
   Tensile_ROOT ${Tensile_ROOT} # optional; use if tensile not installed )
target_link_libraries( TARGET Tensile )
```

TODO: Where is the Tensile include directory?

2.12.9.12 Versioning


- **Major**: Tensile increments the major version if the public API changes, or if either the benchmark.yaml or library-config.yaml files change format in a non-backwards-compatible manner.
- **Minor**: Tensile increments the minor version when new kernel, solution or benchmarking features are introduced in a backwards-compatible manner.
- **Patch**: Bug fixes or minor improvements.

2.12.10 rocThrust

HIP back-end for Thrust(alpha release)

2.12.10.1 Introduction

Thrust is a parallel algorithm library. This library has been ported to HIP/ROCm platform, which uses the rocPRIM library. The HIP ported library works on HIP/ROCm platforms. Currently there is no CUDA backend in place.
2.12.10.2 Requirements

Software

- Git
- CMake (3.5.1 or later)
- **AMD ROCm platform (1.8.0 or later)**
  - Including HCC compiler, which must be set as C++ compiler on ROCm platform.
- rocPRIM library
  - It will be automatically downloaded and built by CMake script.

Optional:

- GTest
  - Required only for tests. Building tests is enabled by default.
  - It will be automatically downloaded and built by CMake script.

2.12.10.3 Hardware

Visit the following link for ROCm hardware requirements:

2.12.10.4 Build and Install

For build and install:

```
git clone https://github.com/ROCmSoftwarePlatform/rocThrust

cd rocThrust; mkdir build; cd build

# Configure rocThrust, setup options for your system.
# Build options:
# BUILD_TEST - ON by default,
#
# ! IMPORTANT !
# On ROCm platform set C++ compiler to HCC. You can do it by adding 'CXX=<path-to-hcc>
# before 'cmake' or setting cmake option 'CMAKE_CXX_COMPILER' with the path to the
# HCC compiler.
# [CXX=hcc] cmake ../. # or cmake-gui ../.

# Build
make -j4
# Optionally, run tests if they're enabled.
cmake --output-on-failure

# Package
make package

# Install
[sudo] make install
```
2.12.10.5 Using rocThrust In A Project

Recommended way of including rocThrust into a CMake project is by using its package configuration files.

```bash
# On ROCm rocThrust requires rocPRIM
find_package(rocprim REQUIRED CONFIG PATHS "/opt/rocm/rocprim")

# "/opt/rocm" - default install prefix
find_package(rocthrust REQUIRED CONFIG PATHS "/opt/rocm/rocthrust")

... includes rocThrust headers and roc::rocprim_hip target
target_link_libraries(<your_target> rocthrust)
```

2.12.10.6 Running Unit Tests

```bash
# Go to rocThrust build directory
cd rocThrust; cd build

# To run all tests
ctest

# To run unit tests for rocThrust.
./test/<unit-test-name>
```

2.12.10.7 Documentation

Documentation is available [here](#).

2.12.10.8 Support

Bugs and feature requests can be reported through the issue tracker.

2.12.11 ROCm SMI library

The ROCm System Management Interface Library, or ROCm SMI library, is part of the Radeon Open Compute ROCm software stack. It is a C library for Linux that provides a user space interface for applications to monitor and control GPU applications.

2.12.11.1 Important note about Versioning and Backward Compatibility

The ROCm SMI library is currently under development, and therefore subject to change either at the ABI or API level. The intention is to keep the API as stable as possible even while in development, but in some cases we may need to break backwards compatibility in order to ensure future stability and usability. Following Semantic Versioning rules, while the ROCm SMI library is in high state of change, the major version will remain 0, and backward compatibility is not ensured.

Once new development has leveled off, the major version will become greater than 0, and backward compatibility will be enforced between major versions.
2.12.11.2 Building ROCm SMI

2.12.11.2.1 Additional Required software for building

In order to build the ROCm SMI library, the following components are required. Note that the software versions listed are what was used in development. Earlier versions are not guaranteed to work:

- CMake (v3.5.0)
- g++ (5.4.0)

In order to build the latest documentation, the following are required:

- DOxygen (1.8.11)
- latex (pdfTeX 3.14159265-2.6-1.40.16)

The source code for ROCm SMI is available on Github.

After the ROCm SMI library git repository has been cloned to a local Linux machine, building the library is achieved by following the typical CMake build sequence. Specifically,

```
$ mk -p build
$ cd build
$ cmake <location of root of ROCm SMI library CMakeLists.txt>
$ make
# Install library file and header; default location is /opt/rocm
$ make install
```

The built library will appear in the build folder.

2.12.11.3 Building the Documentation

The documentation PDF file can be built with the following steps (continued from the steps above):

```
$ make doc
$ cd latex
$ make
```

The reference manual, refman.pdf will be in the latex directory upon a successful build.

2.12.11.4 Building the Tests

In order to verify the build and capability of ROCm SMI on your system and to see an example of how ROCm SMI can be used, you may build and run the tests that are available in the repo. To build the tests, follow these steps:

```
# Set environment variables used in CMakeLists.txt file
$ ROCM_DIR=<location of ROCm SMI library>
$ mkdir <location for test build>
$ cd <location for test build>
$ cmake -DROCM_DIR=<location of ROCM SMI library .so> <ROCm SMI source root>/tests/rocm_smi_test
$ make
```

To run the test, execute the program rsmitst that is built from the steps above.
2.12.11.5 Usage

2.12.11.5.1 Device Indices

Many of the functions in the library take a “device index”. The device index is a number greater than or equal to 0, and less than the number of devices detected, as determined by rsmi_num_monitor_devices(). The index is used to distinguish the detected devices from one another. It is important to note that a device may end up with a different index after a reboot, so an index should not be relied upon to be constant over reboots.

2.12.11.5.2 Hello ROCm SMI

The only required ROCm-SMI call for any program that wants to use ROCm-SMI is the rsmi_init() call. This call initializes some internal data structures that will be used by subsequent ROCm-SMI calls.

When ROCm-SMI is no longer being used, rsmi_shut_down() should be called. This provides a way to do any releasing of resources that ROCm-SMI may have held. In many cases, this may have no effect, but may be necessary in future versions of the library.

A simple “Hello World” type program that displays the device ID of detected devices would look like this:

```c
#include <stdint.h>
#include "rocm_smi/rocm_smi.h"
int main() {
    rsmi_status_t ret;
    uint32_t num_devices;
    uint64_t dev_id;

    // We will skip return code checks for this example, but it
    // is recommended to always check this as some calls may not
    // apply for some devices or ROCm releases
    ret = rsmi_init(0);
    ret = rsmi_num_monitor_devices(&num_devices);

    for (int i=0; i < num_devices; ++i) {
        ret = rsmi_dev_id_get(i, &dev_id);
        // dev_id holds the device ID of device i, upon a
        // successful call
    }
    ret = rsmi_shut_down();
    return 0;
}
```

2.12.12 RCCL

ROCm Communication Collectives Library
2.12.12.1 Introduction

RCCL (pronounced “Rickle”) is a stand-alone library of standard collective communication routines for GPUs, implementing all-reduce, all-gather, reduce, broadcast, and reduce-scatter. It has been optimized to achieve high bandwidth on platforms using PCIe, xGMI as well as networking using InfiniBand Verbs or TCP/IP sockets. RCCL supports an arbitrary number of GPUs installed in a single node, and can be used in either single- or multi-process (e.g., MPI) applications. Multi node support is planned for a future release.

The collective operations are implemented using ring algorithms and have been optimized for throughput and latency. For best performance, small operations can be either batched into larger operations or aggregated through the API.

2.12.12.2 Requirements

- ROCm supported GPUs
- ROCm stack installed on the system (HIP runtime & HCC)
- For building and running the unit tests, chrpath will need to be installed on your machine first. (sudo apt-get install chrpath)

2.12.12.3 Quickstart

RCCL directly depends on HIP runtime & HCC C++ compiler which are part of the ROCm software stack. In addition, HC Direct Function call support needs to be present on your machine. There are binaries for hcc and HIP that need to be installed to get HC Direct Function call support. These binaries are currently packaged with roc-master, and will be included in ROCm 2.4.

The root of this repository has a helper script ‘install.sh’ to build and install RCCL on Ubuntu with a single command. It does not take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it’s a great way to get started quickly and can serve as an example of how to build/install.

- ./install.sh – builds library including unit tests
- ./install.sh -i – builds and installs the library to /opt/rocm/rccl; installation path can be changed with --prefix argument (see below.)
- ./install.sh -h – shows help
- ./install.sh -t – builds library including unit tests
- ./install.sh -r – runs unit tests (must be already built)
- ./install.sh -p – builds RCCL package
- ./install.sh --prefix – specify custom path to install RCCL to (default:/opt/rocm)

2.12.12.4 Manual

2.12.12.4.1 To build the library:

```bash
$ git clone https://github.com/ROCmSoftwarePlatform/rccl.git
$ cd rccl
$ mkdir build
$ cd build
$ CXX=/opt/rocm/bin/hcc cmake
$ make -j 8
```
You may substitute a path of your own choosing for CMAKE_INSTALL_PREFIX. Note: ensure rocm-cmake is installed.

```bash
apt install rocm-cmake.
```

### 2.12.12.5 To build the RCCL package and install package:

Assuming you have already cloned this repository and built the library as shown in the previous section:

```bash
$ cd rccl/build
$ make package
$ sudo dpkg -i *.deb
```

RCCL package install requires sudo/root access because it creates a directory called “rccl” under /opt/rocm/. This is an optional step and RCCL can be used directly by including the path containing librcl.so.

### 2.12.12.6 Tests

There are unit tests implemented with the Googletest framework in RCCL, which are currently a work-in-progress. To invoke the unit tests, go to the rccl-install folder, then the test/ subfolder, and execute the appropriate unit test executable(s). Several notes for running the unit tests:

- The LD_LIBRARY_PATH environment variable will need to be set to include /path/to/rccl-install/lib/ in order to run the unit tests.
- The HSA_FORCE_FINE_GRAIN_PCIE environment variable will need to be set to 1 in order to run the unit tests.

An example call to the unit tests:

```bash
$ LD_LIBRARY_PATH=rccl-install/lib/ HSA_FORCE_FINE_GRAIN_PCIE=1 rccl-install/test/
...UnitTests
```

There are also other performance and error-checking tests for RCCL. These are maintained separately here. See the rccl-tests README for more information on how to build and run those tests.

### 2.12.12.7 Library and API Documentation

Please refer to the Library documentation for current documentation.

### 2.12.12.8 Copyright

All source code and accompanying documentation is copyright (c) 2015-2018, NVIDIA CORPORATION. All rights reserved.

All modifications are copyright (c) 2019 Advanced Micro Devices, Inc. All rights reserved.
2.12.13 hipCUB

hipCUB is a thin wrapper library on top of rocPRIM or CUB. It enables developers to port project using CUB library to the HIP layer and to run them on AMD hardware. In ROCm environment hipCUB uses rocPRIM library as the backend, however, on CUDA platforms it uses CUB instead.

2.12.13.1 Requirements

- Git
- CMake (3.5.1 or later)
- For AMD GPUs:
  - **AMD ROCm platform (1.8.0 or later)**
    * Including HCC compiler, which must be set as C++ compiler on ROCm platform.
  - **rocPRIM library**
    * It will be automatically downloaded and built by CMake script.
- For NVIDIA GPUs:
  - CUDA Toolkit
  - CUB library (automatically downloaded and by CMake script)

Optional:
- GTest
  - Required only for tests. Building tests is enabled by default.
  - It will be automatically downloaded and built by CMake script.

2.12.13.2 Build And Install

```bash
git clone https://github.com/ROCmSoftwarePlatform/hipCUB.git

# Go to hipCUB directory, create and go to the build directory.
cd hipCUB; mkdir build; cd build

# Configure hipCUB, setup options for your system.
# Build options:
# BUILD_TEST - ON by default,
#
# ! IMPORTANT !
# On ROCm platform set C++ compiler to HCC. You can do it by adding '
# before 'cmake' or setting cmake option 'CMAKE_CXX_COMPILER' to path to the HCC
# compiler.
# [CXX=hcc] cmake ../. # or cmake-gui ../.

# Build
make -j4

# Optionally, run tests if they're enabled.
cmake --output-on-failure
```

(continues on next page)
# Package
make package

# Install
[sudo] make install

## 2.12.13.3 Using hipCUB In A Project

Recommended way of including hipCUB into a CMake project is by using its package configuration files.

```bash
# On ROCm hipCUB requires rocPRIM
find_package(rocprim REQUIRED CONFIG PATHS "/opt/rocm/rocprim")

# "/opt/rocm" - default install prefix
find_package(hipcub REQUIRED CONFIG PATHS "/opt/rocm/hipcub")

... 
# On ROCm: includes hipCUB headers and roc::rocprim_hip target
# On CUDA: includes only hipCUB headers, user has to include CUB directory
target_link_libraries(<your_target> hip::hipcub)
```

Include only the main header file:

```bash
#include <hipcub/hipcub.hpp>
```

CUB or rocPRIM headers are included by hipCUB depending on the current HIP platform.

## 2.12.13.4 Running Unit Tests

```bash
# Go to hipCUB build directory
cd hipCUB; cd build

# To run all tests
c-test

# To run unit tests for hipCUB
./test/hipcub/<unit-test-name>
```

## 2.12.13.5 Documentation

```bash
# go to hipCUB doc directory
cd hipCUB; cd doc

# run doxygen
doxygen Doxyfile

# open html/index.html
```
2.12.13.6 Support

Bugs and feature requests can be reported through the issue tracker.

2.12.13.7 Contributions and License

Contributions of any kind are most welcome! More details are found at CONTRIBUTING and LICENSE.

2.12.14 AMD

MiGraphX

AMD’s graph optimization engine.

2.12.14.1 Prerequisites

• ROCm cmake modules required
• MIOpen for running on the GPU
• HIP for running on the GPU
• Protobuf for reading onnx files
• Half - IEEE 754-based half-precision floating point library
• pybind11 - for python bindings

Installing the dependencies

Dependencies can be installed using the ROCm build tool rbuild.

To install rbuild:

```
pip install https://github.com/RadeonOpenCompute/rbuild/archive/master.tar.gz
```

To build dependencies along with MiGraphX

```
rbuild build -d depend --cxx=/opt/rocm/bin/hcc
```

This builds dependencies in the subdirectory named depend and then builds MiGraphX using these dependencies.

2.12.14.2 Building MiGraphX from source

Configuring with cmake

First create a build directory:

```
mkdir build;
cd build;
```

Next configure cmake. The hcc compiler is required to build the MIOpen backend:

```
CXX=/opt/rocm/bin/hcc cmake ..
```

If the dependencies from install_deps.cmake was installed to another directory, the CMAKE_PREFIX_PATH needs to be set to what --prefix was set to from install_deps.cmake:
Changing the cmake configuration

The configuration can be changed after running cmake by using ccmake:

ccmake .. OR cmake-gui: cmake-gui ..

2.12.14.3 Building the library

The library can be built, from the build directory using the ‘Release’ configuration:

```bash
cmake --build . --config Release OR make
```

And can be installed by using the ‘install’ target:

```bash
cmake --build . --config Release --target install OR make install
```

This will install the library to the CMAKE_INSTALL_PREFIX path that was set.

2.12.14.4 Running the tests

The tests can be run by using the ‘check’ target:

```bash
cmake --build . --config Release --target check OR make check
```

2.12.14.5 Building the documentation

HTML and PDF documentation can be built using:

```bash
cmake --build . --config Release --target doc OR make doc
```

This will build a local searchable web site inside the doc/html folder.

Documentation is built using Doxygen, Sphinx, and Breathe.

Requirements for both Sphinx and Breathe can be installed with:

```bash
pip install -r doc/requirements.txt
```

Depending on your setup sudo may be required for the pip install.

2.12.14.6 Formatting the code

All the code is formatted using clang-format. To format a file, use:

```bash
clang-format-5.0 -style=file -i <path-to-source-file>
```

Also, githooks can be installed to format the code per-commit:

```
./.githooks/install
```
2.12.14.7 Using docker

The easiest way to setup the development environment is to use docker. You can build the top-level docker file:

docker build -t migraphx .

Then to enter the development environment use docker run:

docker run --device='/dev/kfd' --device='/dev/dri' -v=`pwd`:/data -w /data --group-˓ →add video -it migraphx

2.12.15 Deprecated

2.12.15.1 hCRNG

hCRNG has been deprecated and has been replaced by rocRAND

The hCRNG library is an implementation of uniform random number generators targeting the AMD heterogeneous hardware via HCC compiler runtime. The computational resources of underlying AMD heterogenous compute gets exposed and exploited through the HCC C++ frontend. Refer here for more details on HCC compiler.

For more information, please refer HCRG

2.12.15.2 hipeigen

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

For more information, please refer HIPE

2.12.15.3 clFFT

clFFT is a software library containing FFT functions written in OpenCL. In addition to GPU devices, the library also supports running on CPU devices to facilitate debugging and heterogeneous programming.

For more information, please refer CLFF

2.12.15.4 clBLAS

This repository houses the code for the OpenCL™ BLAS portion of clMath. The complete set of BLAS level 1, 2 & 3 routines is implemented. Please see Netlib BLAS for the list of supported routines. In addition to GPU devices, the library also supports running on CPU devices to facilitate debugging and multicore programming. APPML 1.12 is the most current generally available pre-packaged binary version of the library available for download for both Linux and Windows platforms.

The primary goal of clBLAS is to make it easier for developers to utilize the inherent performance and power efficiency benefits of heterogeneous computing. clBLAS interfaces do not hide nor wrap OpenCL interfaces, but rather leaves OpenCL state management to the control of the user to allow for maximum performance and flexibility.

The clBLAS library does generate and enqueue optimized OpenCL kernels, relieving the user from the task of writing, optimizing and maintaining kernel code themselves.

For more information, please refer CLB
2.12.15.5 clSPARSE

an OpenCL™ library implementing Sparse linear algebra routines. This project is a result of a collaboration between AMD Inc. and Vratis Ltd..

For more information, please refer CLS

2.12.15.6 clRNG

A library for uniform random number generation in OpenCL.

Streams of random numbers act as virtual random number generators. They can be created on the host computer in unlimited numbers, and then used either on the host or on computing devices by work items to generate random numbers. Each stream also has equally-spaced substreams, which are occasionally useful. The API is currently implemented for four different RNGs, namely the MRG31k3p, MRG32k3a, LFSR113 and Philox-4×32-10 generators.

For more information, please refer CLR

2.12.15.7 hcFFT

hcFFT has been deprecated and has been replaced by rocFFT

For more information, please refer HCF

2.13 Other API References

<table>
<thead>
<tr>
<th>2.13.1 HIP</th>
<th>Language</th>
<th>Runtime</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>• HIP-API</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2.13.2 HIP</th>
<th>Math</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>• HIP-MATH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2.13.3 ROCr</th>
<th>System</th>
<th>Runtime</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ROCr-API</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2.13.4 Math</th>
<th>Library</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>• hcRNG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• clBLAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• clSPARSE_api</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.14 Deep Learning

2.14.1 MIOpen API

• MIOpen API
• MIOpenGEMM API

2.14.2 TensorFlow

2.14.2.1 AMD ROCm TensorFlow v1.15 Release

We are excited to announce the release of ROCm enabled TensorFlow v1.15 for AMD GPUs.

In this release we have the following features enabled on top of upstream TF1.15 enhancements:

• We integrated ROCm RCCL library for mGPU communication, details in RCCL github repo
• XLA backend is enabled for AMD GPUs, the functionality is complete, performance optimization is in progress.

2.14.2.2 AMD ROCm TensorFlow v2.2.0-beta1 Release

In addition to TensorFlow v1.15 release, we also enabled TensorFlow v2.2.0-beta1 for AMD GPUs. The TF-ROCm 2.2.0-beta1 release supports Tensorflow V2 API. Both whl packages and docker containers are available below.

2.14.2.3 Tensorflow Installation

First, you’ll need to install the open-source AMD ROCm 3.3 stack. For details, see here

Then, install these other relevant ROCm packages:

```
sudo apt update
sudo apt install rocm-libs miopen-hip cxlactivitylogger rccl
```

And finally, install TensorFlow itself (via the Python Package Index):

```
sudo apt install wget python3-pip
# Pip3 install the whl package from PyPI
pip3 install --user tensorflow-rocm
```

Tensorflow v2.2.0 is installed.
2.14.2.4 Tensorflow

Tensorflow docker images are also publicly available, more details can be found here

The official github repository is here

2.14.3 MIOpen

2.14.3.1 ROCm MIOpen v2.0.1 Release

Announcing our new Foundation for Deep Learning acceleration MIOpen 2.0 which introduces support for Convolution Neural Network (CNN) acceleration — built to run on top of the ROCm software stack!

This release includes the following:

- This release contains bug fixes and performance improvements.
- Additionally, the convolution algorithm Implicit GEMM is now enabled by default
- **Known issues:**
  - Backward propagation for batch normalization in fp16 mode may trigger NaN in some cases
  - Softmax Log mode may produce an incorrect result in back propagation
- Source code
- Documentation
  - MIOpen
  - MIOpenGemm

**Changes:**

- Added Winograd multi-pass convolution kernel
- Fixed issue with hip compiler paths
- Fixed immediate mode behavior with auto-tuning environment variable
- Fixed issue with system find-db in-memory cache, the fix enable the cache by default
- Improved logging
- Improved how symbols are hidden in the library
- Updated default behavior to enable implicit GEMM

2.14.3.2 Porting from cuDNN to MIOpen

The porting guide highlights the key differences between the current cuDNN and MIOpen APIs.
2.14.3.3 The ROCm 3.3 has prebuilt packages for MiOpen

Install the ROCm MiOpen implementation (assuming you already have the ‘rocm’ and ‘rocm-opencl-dev’ package installed):

MiOpen can be installed on Ubuntu using

```
apt-get
```

For just OpenCL development

```
sudo apt-get install miopengemm miopen-opencl
```

For HIP development

```
sudo apt-get install miopengemm miopen-hip
```

Or you can build from source code

Currently both the backends cannot be installed on the same system simultaneously. If a different backend other than what currently exists on the system is desired, please uninstall the existing backend completely and then install the new backend.

2.14.4 PyTorch

2.14.4.1 Building PyTorch for ROCm

This is a quick guide to setup PyTorch with ROCm support inside a docker container. Assumes a .deb based system. See ROCm install for supported operating systems and general information on the ROCm software stack.

A ROCm install version 3.3 is required currently.

1. Install or update rocm-dev on the host system:

```
sudo apt-get install rocm-dev
or
sudo apt-get update
sudo apt-get upgrade
```

2.14.4.2 Recommended:Install using published PyTorch ROCm docker image:

2. Obtain docker image:

```
docker pull rocm/pytorch:rocm3.0_ubuntu16.04_py3.6_pytorch
```

3. Clone PyTorch repository on the host:

```
cd ~
git clone https://github.com/pytorch/pytorch.git
cd pytorch
git submodule init
git submodule update
```

4. Start a docker container using the downloaded image:
sudo docker run -it -v $HOME:/data --privileged --rm --device=/dev/kfd --device=/dev/dri --group-add video rocm/pytorch:rocm3.0_ubuntu16.04_py3.6_pytorch

Note: This will mount your host home directory on /data in the container.

5. Change to previous PyTorch checkout from within the running docker:

```
cd /data/pytorch
```

6. Build PyTorch for ROCm:

```
Unless you are running a gfx900/Vega10-type GPU (MI25, Vega56, Vega64,...), explicitly export the GPU architecture to build for, e.g.: export HCC_AMDGPU_TARGET=gfx906
then

.jenkins/pytorch/build.sh
```

This will first hipify the PyTorch sources and then compile using 4 concurrent jobs, needing 16 GB of RAM to be available to the docker image.

7. Confirm working installation:

```
PYTORCH_TEST_WITH_ROCM=1 python test/run_test.py --verbose
```

No tests will fail if the compilation and installation is correct.

8. Install torchvision:

```
pip install torchvision
```

This step is optional but most PyTorch scripts will use torchvision to load models. E.g., running the pytorch examples requires torchvision.

9. Commit the container to preserve the pytorch install (from the host):

```
sudo docker commit <container_id> -m 'pytorch installed'
```

2.14.4.3 Option 2: Install using PyTorch upstream docker file

2. Clone PyTorch repository on the host:

```
cd ~
git clone https://github.com/pytorch/pytorch.git
cd pytorch
git submodule init
git submodule update
```

3. Build PyTorch docker image:

```
cd pytorch/docker/caffe2/jenkins
./build.sh py2-clang7-rocmdeb-ubuntu16.04
```

This should complete with a message “Successfully built <image_id>” Note here that other software versions may be chosen, such setups are currently not tested though!

4. Start a docker container using the new image:
sudo docker run -it -v $HOME:/data --privileged --rm --device=/dev/kfd --device=/dev/dri --group-add video <image_id>

Note: This will mount your host home directory on /data in the container.

5. Change to previous PyTorch checkout from within the running docker:

   cd /data/pytorch

6. Build PyTorch for ROCm:

   Unless you are running a gfx900/Vega10-type GPU (MI25, Vega56, Vega64,…), explicitly export the GPU architecture to build for, e.g.: export HCC_AMDGPU_TARGET=gfx906

   then

   .jenkins/pytorch/build.sh

   This will first hipify the PyTorch sources and then compile using 4 concurrent jobs, needing 16 GB of RAM to be available to the docker image.

7. Confirm working installation:

   PYTORCH_TEST_WITH_ROCM=1 python test/run_test.py --verbose

   No tests will fail if the compilation and installation is correct.

8. Install torchvision:

   pip install torchvision

   This step is optional but most PyTorch scripts will use torchvision to load models. E.g., running the pytorch examples requires torchvision.

9. Commit the container to preserve the pytorch install (from the host):

   sudo docker commit <container_id> -m 'pytorch installed'

2.14.4.4 Option 3: Install using minimal ROCm docker file

2. Download pytorch dockerfile:

   Dockerfile

3. Build docker image:

   cd pytorch_docker
   sudo docker build .

   This should complete with a message “Successfully built <image_id>”

4. Start a docker container using the new image:

   sudo docker run -it -v $HOME:/data --privileged --rm --device=/dev/kfd --device=/dev/dri --group-add video <image_id>

   Note: This will mount your host home directory on /data in the container.

5. Clone pytorch master (on to the host):
cd ~
git clone https://github.com/pytorch/pytorch.git or git clone https://github.com/-ROCmSoftwarePlatform/pytorch.git
cd pytorch
git submodule init
git submodule update

6. Run “hipify” to prepare source code (in the container):

   cd /data/pytorch/
   python tools/amd_build/build_pytorch_amd.py
   python tools/amd_build/build_caffe2_amd.py

7. Build and install pytorch:

   Unless you are running a gfx900/Vega10-type GPU (MI25, Vega56, Vega64,...), explicitly export the GPU architecture to build for, e.g.: export HCC_AMDGPU_TARGET=gfx906
   then
   USE_ROCM=1 MAX_JOBS=4 python setup.py install --user

   Use MAX_JOBS=n to limit peak memory usage. If building fails try falling back to fewer jobs. 4 jobs assume available main memory of 16 GB or larger.

8. Confirm working installation:

   PYTORCH_TEST_WITH_ROCM=1 python test/run_test.py --verbose

   No tests will fail if the compilation and installation is correct.

9. Install torchvision:

   pip install torchvision

   This step is optional but most PyTorch scripts will use torchvision to load models. E.g., running the pytorch examples requires torchvision.

10. Commit the container to preserve the pytorch install (from the host):

   sudo docker commit <container_id> -m 'pytorch installed'

2.14.4.5 Try PyTorch examples

1. Clone the PyTorch examples repository:

   git clone https://github.com/pytorch/examples.git

2. Run individual example: MNIST

   cd examples/mnist

   Follow instructions in README.md, in this case:
   pip install -r requirements.txt python main.py

3. Run individual example: Try ImageNet training

2.14. Deep Learning 553
cd ../imagenet

Follow instructions in README.md.

### 2.14.5 Caffe2

#### 2.14.5.1 Building Caffe2 for ROCm

This is a quick guide to setup Caffe2 with ROCm support inside docker container and run on AMD GPUs. Caffe2 with ROCm support offers complete functionality on a single GPU achieving great performance on AMD GPUs using both native ROCm libraries and custom hip kernels. This requires your host system to have rocm-3.3s drivers installed. Please refer to ROCm install to install ROCm software stack. If your host system doesn’t have docker installed, please refer to docker install. It is recommended to add the user to the docker group to run docker as a non-root user, please refer here.

This guide provides two options to run Caffe2.

1. Launch the docker container using a docker image with Caffe2 installed.
2. Build Caffe2 from source inside a Caffe2 ROCm docker image.

#### 2.14.5.2 Option 1: Docker image with Caffe2 installed:

This option provides a docker image which has Caffe2 installed. Users can launch the docker container and train/run deep learning models directly. This docker image will run on both gfx900(Vega10-type GPU - MI25, Vega56, Vega64,...) and gfx906(Vega20-type GPU - MI50, MI60).

1. Launch the docker container

```bash
docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video -v rocm/pytorch:rocm3.0_ubuntu16.04_py3.6_caffe2
```

This will automatically download the image if it does not exist on the host. You can also pass -v argument to mount any data directories on to the container.

#### 2.14.5.3 Option 2: Install using Caffe2 ROCm docker image:

1. Clone PyTorch repository on the host:

```bash
cd ~
git clone --recurse-submodules https://github.com/pytorch/pytorch.git
cd pytorch
git submodule update --init --recursive
```

2. Launch the docker container

```bash
docker pull rocm/pytorch:pytorch:rocm3.0_ubuntu16.04_py3.6_caffe2
docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video -v $PWD:/pytorch rocm/pytorch:rocm3.0_ubuntu16.04_py3.6_caffe2
```

3. Build Caffe2 from source

```bash
cd /pytorch
```

If running on gfx900/vega10-type GPU(MI25, Vega56, Vega64,...)
2.14.5.4 Test Caffe2 Installation

To validate Caffe2 installation, run

1. Test Command

```bash
cd ~ && python -c 'from caffe2.python import core' 2>/dev/null && echo "Success" || echo "Failure"
```

2. Running unit tests in Caffe2

```bash
cd /pytorch
./jenkins/caffe2/test.sh
```

2.14.5.5 Run benchmarks

Caffe2 benchmarking script supports the following networks MLP, AlexNet, OverFeat, VGGA, Inception

To run benchmarks for networks MLP, AlexNet, OverFeat, VGGA, Inception run the command from pytorch home directory replacing `<name_of_the_network>` with one of the networks.

```bash
python caffe2/python/convnet_benchmarks.py --batch_size 64 --model <name_of_the_network> --engine MIOPEN
```

2.14.5.6 Running example scripts

Please refer to the example scripts in `caffe2/python/examples`. It currently has `resnet50_trainer.py` which can run ResNet’s, ResNeXt’s with various layer, groups, depth configurations and `char_rnn.py` which uses RNNs to do character level prediction.

2.14.5.7 Building own docker images

After cloning the pytorch repository, you can build your own Caffe2 ROCm docker image. Navigate to pytorch repo and run

```bash
cd docker/caffe2/jenkins
./build.sh py2-clang7-rocmdeb-ubuntu16.04
```

This should complete with a message “Successfully built <image_id>” which can then be used to install Caffe2 as in Option 2 above.
2.15 MIVisionX

MIVisionX toolkit is a set of comprehensive computer vision and machine intelligence libraries, utilities, and applications bundled into a single toolkit. AMD MIVisionX delivers highly optimized open source implementation of the Khronos OpenVX™ and OpenVX™ Extensions along with Convolution Neural Net Model Compiler & Optimizer supporting ONNX, and Khronos NNEF™ exchange formats. The toolkit allows for rapid prototyping and deployment of optimized workloads on a wide range of computer hardware, including small embedded x86 CPUs, APUs, discrete GPUs, and heterogeneous servers.

- **AMD OpenVX**

- **AMD OpenVX Extensions**
  - Loom 360 Video Stitch Library
  - Neural Net Library
  - OpenCV Extension
  - RPP Extension
  - WinML Extension

- **Applications**

- **Neural Net Model Compiler and Optimizer**

- **RALI**

- **Samples**

- **Toolkit**

- **Utilities**
  - Inference Generator
  - Loom Shell
  - RunCL
  - RunVX

- **Prerequisites**

- **Build and Install MIVisionX**

- **Verify the Installation**

- **Docker**

- **Release Notes**

AMD OpenVX [amd_openvx] is a highly optimized open source implementation of the Khronos OpenVX computer vision specification. It allows for rapid prototyping as well as fast execution on a wide range of computer hardware, including small embedded x86 CPUs and large workstation discrete GPUs.
The OpenVX framework provides a mechanism to add new vision functions to OpenVX by 3rd party vendors. This project has below mentioned OpenVX modules and utilities to extend amd_openvx project, which contains the AMD OpenVX Core Engine.

- **amd_looms:** AMD Radeon Loom stitching library for live 360 degree video applications.

- **amd_nn:** OpenVX neural network module

- **amd_opencv:** OpenVX module that implements a mechanism to access OpenCV functionality as OpenVX kernels

- **amd_winml:** WinML extension will allow developers to import a pre-trained ONNX model into an OpenVX graph and add hundreds of different pre & post processing vision/generic/user-defined functions, available in OpenVX and OpenCV interop, to the input and output of the neural net model. This will allow developers to build an end to end application for inference.
MIVisionX has a number of applications built on top of OpenVX modules, it uses AMD optimized libraries to build applications which can be used to prototype or used as models to develop a product.

- **Cloud Inference Application**: This sample application does inference using a client-server system.
- **Digit Test** This sample application is used to recognize hand written digits.
- **MIVisionX OpenVX Classification**: This sample application shows how to run supported pre-trained caffe models with MIVisionX RunTime.
- **MIVisionX WinML Classification**: This sample application shows how to run supported ONNX models with MIVisionX RunTime on Windows.
- **MIVisionX WinML YoloV2**: This sample application shows how to run tiny yolov2(20 classes) with MIVisionX RunTime on Windows.
- **External Applications**
Neural Net Model Compiler & Optimizer `model_compiler` converts pre-trained neural net models to MIVisionX runtime code for optimized inference.

The Radeon Augmentation Library RALI is designed to efficiently decode and process images and videos from a variety of storage formats and modify them through a processing graph programmable by the user.

MIVisionX samples using OpenVX and OpenVX extension libraries

**GDF - Graph Description Format**

MIVisionX samples using runvx with GDF

`skintonedetect.gdf`

**usage:**

```bash
runvx skintonedetect.gdf
```

`canny.gdf`
• RunVX: command-line utility to execute OpenVX graph described in GDF text file
• **RunCL**: command-line utility to build, execute, and debug OpenCL programs
• **CPU**: SSE4.1 or above CPU, 64-bit
• **GPU**: GFX7 or above [optional]
• **APU**: Carrizo or above [optional]

**Note:** Some modules in MIVisionX can be built for CPU only. To take advantage of advanced features and modules we recommend using AMD GPUs or AMD APUs.

### Windows

- Windows 10
- Windows SDK
- Visual Studio 2017
- Install the latest drivers and [OpenCL SDK](https://github.com/GPUOpen-LibrariesAndSDKs/CL-SDK/releases/tag/1.0)

**OpenCV**
- Set OpenCV_DIR environment variable to OpenCV/build folder
- Add %OpenCV_DIR%x64vc14bin or %OpenCV_DIR%x64vc15bin to your PATH

### Linux

- Install ROCm
- ROCm CMake, MIOpenGEMM & MIOpen for Neural Net Extensions (vx
- CMake 2.8 or newer [download](https://github.com/opencv/opencv/releases/tag/3.4.0)
- `Protobuf` for inference generator & model compiler
  - install libprotobuf-dev and protobuf-compiler needed for vx
  - `OpenCV` <https://github.com/opencv/opencv/releases/tag/3.4.0>_
    - Set OpenCV_DIR environment variable to OpenCV/build folder
- **FFMPEG** - Optional
  - FFMPEG is required for amd_media & mv_deploy modules

For the convenience of the developer, we here provide the setup script which will install all the dependencies required by this project.

**MIVisionX-setup.py** - This script builds all the prerequisites required by MIVisionX. The setup script creates a deps folder and installs all the prerequisites, this script only needs to be executed once. If -d option for directory is not given the script will install deps folder in ‘~/’ directory by default, else in the user specified folder.

### Prerequisites for running the scripts

- ubuntu 16.04/18.04 or CentOS 7.5/7.6
- ROCm supported hardware
- ROCm

usage:
python MIVisionX-setup.py --directory [setup directory - optional]  
  --installer [Package management tool - optional]
  → (default: apt-get) [options: Ubuntu: apt-get; CentOS: yum]
    → miopen [MIOpen Version - optional (default: 2.1.0)]
    → miopenengemm [MIOpenGEMM Version - optional (default: 1.1.5)]
    → ffmpeg [FFMPEG Installation - optional (default: no)]
  → [options: Install ffmpeg - yes]
    → rpp [RPP Installation - optional (default: yes)]
  → [options: yes/no]

Note: use --installer yum for CentOS

Windows

Using .msi packages

- MIVisionX-installer.msi: MIVisionX
- MIVisionX_WinML-installer.msi: MIVisionX for WinML

Using Visual Studio 2017 on 64-bit Windows 10

- Install OpenCL_SDK
- Install OpenCV with/without contrib to support camera capture, image display, & opencv extensions
  - Set OpenCV_DIR environment variable to OpenCV/build folder
  - Add %OpenCV_DIR%\x64\vc14\bin or %OpenCV_DIR%\x64\vc15\bin to your PATH
- Use MIVisionX.sln to build for x64 platform

NOTE: vx.nn is not supported on Windows in this release

Linux

Using apt-get/yum

Prerequisites

- Ubuntu 16.04/18.04 or CentOS 7.5/7.6
- ROCm supported hardware
- ROCm

Ubuntu

sudo apt-get install mivisionx

CentOS

sudo yum install mivisionx

Note:

- vx_winml is not supported on Linux
- source code will not available with apt-get/yum install
- executables placed in /opt/rocm/mivisionx/bin and libraries in /opt/rocm/mivisionx/lib
- OpenVX and module header files into /opt/rocm/mivisionx/include
- model compiler, toolkit, & samples placed in /opt/rocm/mivisionx
- Package (.deb & .rpm) install requires OpenCV v3.4.0 to execute AMD OpenCV extensions
Using MIVisionX-setup.py and CMake on Linux (Ubuntu 16.04/18.04 or CentOS 7.5/7.6) with ROCm

- Install ROCm
- Use the below commands to setup and build MIVisionX

```
python MIVisionX-setup.py --directory [setup directory - optional]
  --installer [Package management tool - optional (default: apt-get)]
  --miopen [MIOpen Version - optional (default: 2.1.0)]
  --miopengemm [MIOpenGEMM Version - optional (default: 1.1.5)]
  --ffmpeg [FFMPEG Installation - optional (default: no)]
  --rpp [RPP Installation - optional (default: yes)]
```

**Note:** Use `--installer yum` for CentOS

```
mkdir build
cd build
cmake ..
make -j8
sudo make install
```

**Note:**
- vx_winml is not supported on Linux
- the installer will copy all executables into `/opt/rocm/mivisionx/bin` and libraries into `/opt/rocm/mivisionx/lib`
- the installer also copies all the OpenVX and module header files into `/opt/rocm/mivisionx/include` folder

Using CMake on Linux (Ubuntu 16.04 64-bit or CentOS 7.5 / 7.6) with ROCm

- Install ROCm
- `git clone, build and install other ROCm projects (using cmake and % make install)` in the below order for `vx_nn`

  - rocm-cmake
  - MIOpenGEMM
  - MIOpen – make sure to use -DMIOPEN_BACKEND=OpenCL option with cmake

- install protobuf
- install OpenCV
- install FFMPEG n4.0.4 - Optional
- **build and install (using cmake and % make install)**
  - executables will be placed in bin folder
  - libraries will be placed in lib folder
  - the installer will copy all executables into `/opt/rocm/mivisionx/bin` and libraries into `/opt/rocm/mivisionx/lib`
  - the installer also copies all the OpenVX and module header files into `/opt/rocm/mivisionx/include` folder
• add the installed library path to LD_LIBRARY_PATH environment variable (default /opt/rocm/mivisionx/lib)
• add the installed executable path to PATH environment variable (default /opt/rocm/mivisionx/bin)

Linux

• The installer will copy all executables into /opt/rocm/mivisionx/bin and libraries into /opt/rocm/mivisionx/lib
• The installer also copies all the OpenVX and OpenVX module header files into /opt/rocm/mivisionx/include folder
• Apps, Samples, Documents, Model Compiler and Toolkit are placed into /opt/rocm/mivisionx
• Run samples to verify the installation
  – Canny Edge Detection

![Image of Canny Edge Detection]

```bash
export PATH=$PATH:/opt/rocm/mivisionx/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/rocm/mivisionx/lib
runvx /opt/rocm/mivisionx/samples/gdf/canny.gdf
```

Note: More samples are available here

MIVisionX provides developers with docker images for Ubuntu 16.04, Ubuntu 18.04, CentOS 7.5, & CentOS 7.6. Using docker images developers can quickly prototype and build applications without having to be locked into a single system setup or lose valuable time figuring out the dependencies of the underlying software.

MIVisionX Docker

• Ubuntu 16.04
• Ubuntu 18.04
• CentOS 7.5
• CentOS 7.6

Docker Workflow Sample on Ubuntu 16.04/18.04

Prerequisites

• Ubuntu 16.04/18.04
• rocm supported hardware
Workflow

Step 1 - Install rocm-dkms

```
sudo apt update
sudo apt dist-upgrade
sudo apt install libnuma-dev
sudo reboot
wget -qO - http://repo.radeon.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
echo 'deb [arch=amd64] http://repo.radeon.com/rocm/apt/debian/ xenial main' | sudo tee /etc/apt/sources.list.d/rocm.list
sudo apt update
sudo apt install rocm-dkms
sudo reboot
```

Step 2 - Setup Docker

```
sudo apt-get install curl
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
apt-cache policy docker-ce
sudo apt-get install -y docker-ce
sudo systemctl status docker
```

Step 3 - Get Docker Image

```
sudo docker pull mivisionx/ubuntu-16.04
```

Step 4 - Run the docker image

```
sudo docker run -it --device=/dev/kfd --device=/dev/dri --cap-add=SYS_RAWIO --device=/dev/mem --group-add video --network host mivisionx/ubuntu-16.04
```

- **Optional:** Map localhost directory on the docker image
  - option to map the localhost directory with trained caffe models to be accessed on the docker image.
  - usage: -v {LOCAL_HOST_DIRECTORY_PATH}:{DOCKER_DIRECTORY_PATH}

```
sudo docker run -it -v /home/:/root/hostDrive/ --device=/dev/kfd --device=/dev/dri --cap-add=SYS_RAWIO --device=/dev/mem --group-add video --network host mivisionx/ubuntu-16.04
```

Note: Display option with docker

- Using host display

```
xhost +local:root
sudo docker run -it --device=/dev/kfd --device=/dev/dri --cap-add=SYS_RAWIO --device=/dev/mem --group-add video --network host --env DISPLAY=unix$DISPLAY --privileged --volume $XAUTH:/root/.
--authority
```

- Test display with MIVisionX sample

2.15. MIVisionX
Known issues

• Package (.deb & .rpm) install requires OpenCV v3.4.0 to execute AMD OpenCV extensions

Tested configurations

• Windows 10
• Linux: Ubuntu - 16.04/18.04 & CentOS - 7.5/7.6
• ROCm: rocm-dkms - 2.9.6
• rocm-cmake - github master:ac45c6e
• MIOpenGEMM - 1.1.5
• MIOpen - 2.1.0
• Protobuf - V3.5.2
• OpenCV - 3.4.0
• Dependencies for all the above packages

2.16 AMD ROCm ROCProfiler

2.16.1 1. Overview

The rocProf is a command line tool implemented on the top of rocProfiler and rocTracer APIs. Source code for rocProf can be found at GitHub: https://github.com/ROCm-Developer-Tools/rocprof

This command line tool is implemented as a script which is setting up the environment for attaching the profiler and then run the provided application command line. The tool uses two profiling plugins loaded by ROC runtime and based on rocProfiler and rocTracer for collecting metrics/counters, HW traces and runtime API/activity traces. The tool consumes an input XML or text file with counters list or trace parameters and provides output profiling data and statistics in various formats as text, CSV and JSON traces. Google Chrome tracing can be used to visualize the JSON traces with runtime API/activity timelines and per kernel counters data.

2.16.2 2. Profiling Modes

‘rocprof’ can be used for GPU profiling using HW counters and application tracing

2.16.3 2.1. GPU profiling

GPU profiling is controlled with input file which defines a list of metrics/counters and a profiling scope. An input file is provided using option ‘-i’. Output CSV file with a line per submitted kernel is generated. Each line has kernel name, kernel parameters and counter values. By option ‘—stats’ the kernel execution stats can be generated in CSV format. Currently profiling has limitation of serializing submitted kernels. An example of input file:
# Perf counters group 1

dmc : Wavefronts VALUInsts SALUInsts SFetchInsts

# Perf counters group 2

dmc : TCC_HIT[0], TCC_MISS[0]

# Filter by dispatches range, GPU index and kernel names
# supported range formats: "3:9", "3:", "3"

range: 1 : 4
gpu: 0 1 2 3
kernel: simple Pass1 simpleConvolutionPass2

An example of profiling command line for `MatrixTranspose` application

$ rocprof -i input.txt MatrixTranspose

RPL: on '191018_011134' from '/..../rocprofiler_pkg' in '/..../MatrixTranspose'
RPL: profiling './MatrixTranspose'
RPL: input file 'input.txt'
RPL: output dir '/tmp/rpl_data_191018_011134_9695'
RPL: result dir '/tmp/rpl_data_191018_011134_9695/input0_results_191018_011134'
ROCProfiler: rc-file '/..../rpl_rc.xml'
ROCProfiler: input from '/tmp/rpl_data_191018_011134_9695/input0.xml'

  gpu_index =
  kernel =
  range =
  4 metrics
    L2CacheHit, VFetchInsts, VWriteInsts, MemUnitStalled
  0 traces
Device name Ellesmere [Radeon RX 470/480/570/580/580X]
PASSED!

ROCProfiler: 1 contexts collected, output directory /tmp/rpl_data_191018_011134_9695/
    input0_results_191018_011134
RPL: '/..../MatrixTranspose/input.csv' is generated

2.1.1. Counters and metrics

There are two profiling features, metrics and traces. Hardware performance counters are treated as the basic metrics
and the formulas can be defined for derived metrics. Counters and metrics can be dynamically configured using XML
configuration files with counters and metrics tables:

- Counters table entry, basic metric: counter name, block name, event id
- Derived metrics table entry: metric name, an expression for calculation the metric from the counters

Metrics XML File Example:

```xml
<gfx8>
  <metric name=L1_CYCLES_COUNTER block=L1 event=0 descr="L1 cache cycles"/>
  <metric name=L1_MISS_COUNTER block=L1 event=33 descr="L1 cache misses"/>
  ...
</gfx8>
<gfx9>
  ...
</gfx9>
<global>
  <metric
    name=L1_MISS_RATIO
    expr=L1_CYCLES_COUNT/L1_MISS_COUNTER
</global>
```

(continues on next page)
2.1.1. Metrics query

Available counters and metrics can be queried by options ‘--list-basic’ for counters and ‘--list-derived’ for derived metrics. The output for counters indicates number of block instances and number of block counter registers. The output for derived metrics prints the metrics expressions. Examples:

```
$ rocprof --list-basic
RPL: on '191018_014450' from '/opt/rocm/rocprofiler' in '/..../MatrixTranspose'
ROCProfiler: rc-file '/..../rpl_rc.xml'
Basic HW counters:
gpu-agent0 : GRBM_COUNT : Tie High - Count Number of Clocks
    block GRBM has 2 counters
    gpu-agent0 : GRBM_GUI_ACTIVE : The GUI is Active
        block GRBM has 2 counters
    . . .
gpu-agent0 : TCC_HIT[0-15] : Number of cache hits.
    block TCC has 4 counters
    gpu-agent0 : TCC_MISS[0-15] : Number of cache misses. UC reads count as misses.
        block TCC has 4 counters
    . . .
```

```
$ rocprof --list-derived
RPL: on '191018_015911' from '/opt/rocm/rocprofiler' in '/home/evgeny/work/BUILD/0_
    → MatrixTranspose'
ROCProfiler: rc-file '/home/evgeny/rpl_rc.xml'
Derived metrics:
gpu-agent0 : TCC_HIT_sum : Number of cache hits. Sum over TCC instances.
    TCC_HIT_sum = sum(TCC_HIT,16)
gpu-agent0 : TCC_MISS_sum : Number of cache misses. Sum over TCC instances.
    TCC_MISS_sum = sum(TCC_MISS,16)
gpu-agent0 : TCC_MC_RDREQ_sum : Number of 32-byte reads. Sum over TCC instaces.
    TCC_MC_RDREQ_sum = sum(TCC_MC_RDREQ,16)
    . . .
```

2.1.1.2. Metrics collecting

Counters and metrics accumulated per kernel can be collected using input file with a list of metrics, see an example in 2.1. Currently profiling has limitation of serializing submitted kernels. The number of counters which can be dumped by one run is limited by GPU HW by number of counter registers per block. The number of counters can be different for different blocks and can be queried, see 2.1.1.1.

2.1.1.2.1. Blocks instancing

GPU blocks are implemented as several identical instances. To dump counters of specific instance square brackets can be used, see an example in 2.1. The number of block instances can be queried, see 2.1.1.1.

2.1.1.2.2. HW limitations

The number of counters which can be dumped by one run is limited by GPU HW by number of counter registers per block. The number of counters can be different for different blocks and can be queried, see 2.1.1.1.

- Metrics groups

To dump a list of metrics exceeding HW limitations the metrics list can be split on groups. The tool supports automatic splitting on optimal metric groups:
$ rocprof -i input.txt ./MatrixTranspose
RPL: on '191018_032645' from '/opt/roc/rocprofiler' in '/....MatrixTranspose'
RPL: profiling './MatrixTranspose'
RPL: input file 'input.txt'
RPL: output dir '/tmp/rpl_data_191018_032645_12106'
RPL: result dir '/tmp/rpl_data_191018_032645_12106/input0_results_191018_032645'
ROCProfiler: rc-file '/....rpl_rc.xml'
ROCProfiler: input from '/tmp/rpl_data_191018_032645_12106/input0.xml"
gpu_index =
kernel =
range =
  20 metrics
    Wavefronts, VALUInsts, SALUInsts, SFetchInsts, FlatVMemInsts, LDSInsts,
    FlatLDSInsts, GDSInsts, VALUUtilization, FetchSize, WriteSize, L2CacheHit,
    VWriteInsts, GPUBusy, VALUBusy, SALUBusy, MemUnitStalled, WriteUnitStalled,
    LDSBankConflict, MemUnitBusy
  0 traces
Device name Ellesmere [Radeon RX 470/480/570/570X/580/580X]

Input metrics out of HW limit. Proposed metrics group set:
  group1: L2CacheHit VWriteInsts MemUnitStalled WriteUnitStalled MemUnitBusy FetchSize,
               FlatVMemInsts LDSInsts VALUInsts SFetchInsts FlatLDSInsts GPUBusy,
               Wavefronts
  group2: WriteSize GDSInsts VALUUtilization VALUBusy SALUBusy LDSBankConflict

ERROR: rocprofiler_open(), Construct(), Metrics list exceeds HW limits
Aborted (core dumped)
Error found, profiling aborted.

- Collecting with multiple runs

To collect several metric groups a full application replay is used by defining several ‘pmc:’ lines in the input file, see 2.1.

2.16.4 2.2. Application tracing

Supported application tracing includes runtime API and GPU activity tracing’ Supported runtimes are: ROCr (HSA API) and HIP Supported GPU activity: kernel execution, async memory copy, barrier packets. The trace is generated in JSON format compatible with Chrome tracing. The trace consists of several sections with timelines for API trace per thread and GPU activity. The timelines events show event name and parameters. Supported options:
  ‘—hsa-trace’, ‘—hip-trace’, ‘—sys-trace’, where ‘sys trace’ is for HIP and HSA combined trace.

2.2.1. HIP runtime trace

The trace is generated by option ‘—hip-trace’ and includes HIP API timelines and GPU activity at the runtime level.

2.2.2. ROCr runtime trace

The trace is generated by option ‘—hsa-trace’ and includes ROCr API timelines and GPU activity at AQL queue level. Also, can provide counters per kernel.

2.2.3. KFD driver trace

The trace is generated by option ‘—kfd-trace’ and includes KFD Thunk API timelines.

It is planned to include memory allocations/migration activity tracing.
2.2.4. Code annotation

Support for application code annotation. Start/stop API is supported to programmatically control the profiling. A ‘roctx’ library provides annotation API. Annotation is visualized in JSON trace as a separate “Markers and Ranges” timeline section.

2.2.4.1. Start/stop API

```c
// Tracing start API
void roctracer_start();

// Tracing stop API
void roctracer_stop();
```

2.2.4.2. rocTX basic markers API

```c
// A marker created by given ASCII message
void roctxMark(const char* message);

// Returns the 0 based level of a nested range being started by given message.
// A negative value is returned on the error.
int roctxRangePush(const char* message);

// Marks the end of a nested range.
// Returns the 0 based level the range.
// A negative value is returned on the error.
int roctxRangePop();
```

2.3. Multiple GPUs profiling

The profiler supports multiple GPU’s profiling and provide GPU id for counters and kernels data in CSV output file. Also, GPU id is indicating for respective GPU activity timeline in JSON trace.

2.16.5 3. Profiling control

Profiling can be controlled by specifying a profiling scope, by filtering trace events and specifying interesting time intervals.

2.16.6 3.1. Profiling scope

Counters profiling scope can be specified by GPU id list, kernel name substrings list and dispatch range. Supported range formats examples: “3:9”, “3:”, “3”. You can see an example of input file in 2.1.

2.16.7 3.2. Tracing control

Tracing can be filtered by events names using profiler input file and by enabling interesting time intervals by command line option.

3.2.1. Filtering traced APIs

A list of traced API names can be specified in profiler input file. An example of input file line for ROCr runtime trace (HAS API): hsa: hsa_queue_create hsa_amd_memory_pool_allocate

3.2.2. Tracing period

Tracing can be disabled on start so it can be enabled with start/stop API:
2.16.8 3.3. Concurrent kernels

Currently concurrent kernels profiling is not supported which is a planned feature. Kernels are serialized.

2.16.9 3.4. Multi-processes profiling

Multi-processes profiling is not currently supported.

2.16.10 3.5. Errors logging

Profiler errors are logged to global logs:

/tmp/aql_profile_log.txt
/tmp/rocprofiler_log.txt
/tmp/roctracer_log.txt

2.16.11 4. 3rd party visualization tools

‘rocprof’ is producing JSON trace compatible with Chrome Tracing, which is an internal trace visualization tool in Google Chrome.

2.16.12 4.1. Chrome tracing

Good review can be found by the link: https://aras-p.info/blog/2017/01/23/Chrome-Tracing-as-Profiler-Frontend/

2.16.13 5. Command line options

The command line options can be printed with option ‘-h’:

rocprof [-h] [--list-basic] [--list-derived] [-i <input .txt/.xml file>] [-o <output CSV file>] <app command line>

Options:
- h - this help
--verbose - verbose mode, dumping all base counters used in the input metrics
--list-basic - to print the list of basic HW counters
--list-derived - to print the list of derived metrics with formulas
--cmd-qts <on|off> - quoting profiled cmd line [on]

-i <.txt|.xml file> - input file
    Input file .txt format, automatically rerun application for every pmc line:

(continues on next page)
# Perf counters group 1

```
# Perf counters group 1
pmc : Wavefronts VALUInsts SALUInsts SFetchInsts FlatVMemInsts LDSInsts
→ FlatLDSInsts GDSInsts FetchSize
```

# Perf counters group 2

```
# Perf counters group 2
pmc : VALUUtilization,WriteSize L2CacheHit
# Filter by dispatches range, GPU index and kernel names
# supported range formats: "3:9", "3:", "3"
range: 1 : 4
gpu: 0 1 2 3
kernel: simple Pass1 simpleConvolutionPass2
```

Input file .xml format, for single profiling run:

```
# Metrics list definition, also the form "<block-name>:<event-id>" can be used
# All defined metrics can be found in the 'metrics.xml'
# There are basic metrics for raw HW counters and high-level metrics for derived counters
<metric name=SQ:4,SQ_WAVES,VFetchInsts</metric>
```

```
# Filter by dispatches range, GPU index and kernel names
<metric
# range formats: "3:9", "3:", "3"
range=""
# list of gpu indexes "0,1,2,3"
gpu_index=""
# list of matched sub-strings "Simple1,Conv1,SimpleConvolution"
kernel=""
</metric>
```

-o <output file> - output CSV file [<input file base>.csv]
The output CSV file columns meaning in the columns order:
  Index - kernels dispatch order index
  KernelName - the dispatched kernel name
  gpu-id - GPU id the kernel was submitted to
  queue-id - the ROCm queue unique id the kernel was submitted to
  queue-index - The ROCm queue write index for the submitted AQL packet
  tid - system application thread id which submitted the kernel
  grd - the kernel's grid size
  wgr - the kernel's work group size
  lds - the kernel's LDS memory size
  scr - the kernel's scratch memory size
  vgpr - the kernel's VGPR size
  sgpr - the kernel's SGPR size
  fbar - the kernel's barriers limitation
  sig - the kernel's completion signal
  ... - The columns with the counters values per kernel dispatch
  DispatchNs/BeginNs/EndNs/CompleteNs - timestamp columns if time-stamping was enabled

-d <data directory> - directory where profiler store profiling data including thread traces [/tmp]
The data directory is removing automatically if the directory is matching the temporary one, which is the default.
-t <temporary directory> - to change the temporary directory [/tmp]
By changing the temporary directory you can prevent removing the profiling data from /tmp or enable removing from not '/tmp' directory.
--basenames <on|off> - to turn on/off truncating of the kernel full function names, till the base ones [off]
--timestamp <on|off> - to turn on/off the kernel dispatches timestamps, dispatch/begin/end/complete [off]
   Four kernel timestamps in nanoseconds are reported:
   DispatchNs - the time when the kernel AQL dispatch packet was written to the queue
   BeginNs - the kernel execution begin time
   EndNs - the kernel execution end time
   CompleteNs - the time when the completion signal of the AQL dispatch packet was received
--ctx-limit <max number> - maximum number of outstanding contexts [0 - unlimited]
--heartbeat <rate sec> - to print progress heartbeats [0 - disabled]
--stats - generating kernel execution stats, file <output name>.stats.csv
--roctx-trace - to enable rocTX application code annotation trace
   Will show the application code annotation in JSON trace "Markers and Ranges" section.
--sys-trace - to trace HIP/HSA APIs and GPU activity, generates stats and JSON trace
   chrome-tracing compatible
--hip-trace - to trace HIP, generates API execution stats and JSON file chrome-tracing compatible
--hsa-trace - to trace HSA, generates API execution stats and JSON file chrome-tracing compatible
--kfd-trace - to trace KFD, generates API execution stats and JSON file chrome-tracing compatible
   Generated files: <output name>_<domain>_stats.txt <output name>.json
Traced API list can be set by input .txt or .xml files.
Input .txt:
   hsa: hsa_queue_create hsa_amd_memory_poolAllocate
Input .xml:
   <trace name="HSA">
     <parameters list="hsa_queue_create, hsa_amd_memory_poolAllocate">
   </parameters>
   </trace>
--trace-start <on|off> - to enable tracing on start [on]
--trace-period <dealy:length:rate> - to enable trace with initial delay, with periodic sample length and rate
   Supported time formats: <number(m|s|ms|us)>
--obj-tracking <on|off> - to turn on/off kernels code objects tracking [off]
   To support V3 code objects.

Configuration file:
You can set your parameters defaults preferences in the configuration file 'rpl_rc.xml'. The search path sequence: .:/home/ evgeny:<package path>
First the configuration file is looking in the current directory, then in your home, and then in the package directory.
Configurable options: 'basenames', 'timestamp', 'ctx-limit', 'heartbeat', 'obj-tracking'.
An example of 'rpl_rc.xml':
<defaults
   basenames=off
   timestamp=off
   ctx-limit=0
   (continues on next page)
2.16.14 6. Publicly available counters and metrics

The following counters are publicly available for commercially available VEGA10/20 GPUs.

Counters:

- GRBMD_COUNT : Tie High - Count Number of Clocks
- GRBMD_GUI_ACTIVE : The GUI is Active
- SQ_WAVES : Count number of waves sent to SQs. (per-simd, emulated, global)
- SQ_INSTS_VALU : Number of VALU instructions issued. (per-simd, emulated)
- SQ_INSTS_VMEM_WR : Number of VMEM write instructions issued (including FLAT). (per-simd, emulated)
- SQ_INSTS_VMEM_RD : Number of VMEM read instructions issued (including FLAT). (per-simd, emulated)
- SQ_INSTS_SALU : Number of SALU instructions issued. (per-simd, emulated)
- SQ_INSTS_SMEM : Number of SMEM instructions issued. (per-simd, emulated)
- SQ_INSTS_FLAT : Number of FLAT instructions issued. (per-simd, emulated)
- SQ_INSTS_FLAT_LDSONLY : Number of FLAT instructions issued that read/wrote only from/to LDS (only works if EARLY_TA_DONE is enabled). (per-simd, emulated)
- SQ_INSTS_LDS : Number of LDS instructions issued (including FLAT). (per-simd, emulated)
- SQ_WAIT_INST_LDS : Number of wave-cycles spent waiting for LDS instruction issue. In units of 4 cycles. (per-simd, nondeterministic)
- SQ_ACTIVE_INST_VALU : regspec 71? Number of cycles the SQ instruction arbiter is working on a VALU instruction. (per-simd, nondeterministic)
- SQ_INST_CYCLES_SALU : Number of cycles needed to execute non-memory read scalar operations. (per-simd, emulated)
- SQ_THREAD_CYCLES_VALU : Number of thread-cycles used to execute VALU operations (similar to INST_CYCLES_VALU but multiplied by # of active threads). (per-simd, emulated)
- SQ_LDS_BANK_CONFLICT : Number of cycles LDS is stalled by bank conflicts. (per-simd, emulated)
- TA_TA_BUSY[0-15] : TA block is busy. Perf_Windowing not supported for this counter.
- TA_FLAT_READ_WAVEFRONTS[0-15] : Number of flat opcode reads processed by the TA.
- TA_FLAT_WRITE_WAVEFRONTS[0-15] : Number of flat opcode writes processed by the TA.
- TCC_HIT[0-15] : Number of cache hits.
- TCC_MISS[0-15] : Number of cache misses. UC reads count as misses.
- TCC_EA_WRREQ[0-15] : Number of transactions (either 32-byte or 64-byte) going over the TC_EA_wrreq interface. Atomics may travel over the same interface and are generally classified as write requests. This does not include probe commands.
- TCC_EA_WRREQ_64B[0-15] : Number of 64-byte transactions going (64-byte write or COMPRESSWRITE) over the TC_EA_wrreq interface.
- TCC_EA_WRREQ_STALL[0-15] : Number of cycles a write request was stalled.
- TCC_EA_RDREQ[0-15] : Number of TCC/Ea read requests (either 32-byte or 64-byte)
- TCC_EA_RDREQ_32B[0-15] : Number of 32-byte TCC/Ea read requests

The following derived metrics have been defined and the profiler metrics XML specification can be found at:

Metrics:
- **TA_BUSY_avr**: TA block is busy. Average over TA instances.
- **TA_BUSY_max**: TA block is busy. Max over TA instances.
- **TA_BUSY_min**: TA block is busy. Min over TA instances.
- **TA_FLAT_READ_WAVEFRONTS_sum**: Number of flat opcode reads processed by the TA.
  - Sum over TA instances.
- **TA_FLAT_WRITE_WAVEFRONTS_sum**: Number of flat opcode writes processed by the TA.
  - Sum over TA instances.
- **TCC_HIT_sum**: Number of cache hits. Sum over TCC instances.
- **TCC_MISS_sum**: Number of cache misses. Sum over TCC instances.
- **TCC_EA_RDREQ_32B_sum**: Number of 32-byte TCC/EA read requests. Sum over TCC instances.
- **TCC_EA_RDREQ_sum**: Number of TCC/EA read requests (either 32-byte or 64-byte).
  - Sum over TCC instances.
- **TCC_EA_WRREQ_sum**: Number of transactions (either 32-byte or 64-byte) going over the TC_EA_wrreq interface. Sum over TCC instances.
- **TCC_EA_WRREQ_64B_sum**: Number of 64-byte transactions going (64-byte write or CMPSWAP) over the TC_EA_wrreq interface. Sum over TCC instances.
- **TCC_WRREQ_STALL_max**: Number of cycles a write request was stalled. Max over TCC instances.
- **TCC_MC_WRREQ_sum**: Number of 32-byte effective writes. Sum over TCC instances.
- **FETCH_SIZE**: The total kilobytes fetched from the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
- **WRITE_SIZE**: The total kilobytes written to the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
- **GPUBusy**: The percentage of time GPU was busy.
- **Wavefronts**: Total wavefronts.
- **VALUInsts**: The average number of vector ALU instructions executed per work-item (affected by flow control).
- **SALUInsts**: The average number of scalar ALU instructions executed per work-item (affected by flow control).
- **VFetchInsts**: The average number of vector fetch instructions from the video memory executed per work-item (affected by flow control). Excludes FLAT instructions that fetch from video memory.
- **SFetchInsts**: The average number of scalar fetch instructions from the video memory executed per work-item (affected by flow control).
- **VWriteInsts**: The average number of vector write instructions to the video memory executed per work-item (affected by flow control). Excludes FLAT instructions that write to video memory.
- **FlatVMemInsts**: The average number of FLAT instructions that read from or write to the video memory executed per work item (affected by flow control). Includes FLAT instructions that read from or write to scratch.
- **LDSInsts**: The average number of LDS read or LDS write instructions executed per work item (affected by flow control). Excludes FLAT instructions that read from or write to LDS.
- **FlatLDSInsts**: The average number of FLAT instructions that read or write to LDS executed per work item (affected by flow control).
- **GDSInsts**: The average number of GDS read or GDS write instructions executed per work item (affected by flow control).
- **VALUUtilization**: The percentage of active vector ALU threads in a wave. A lower number can mean either more thread divergence in a wave or that the work-group size is not a multiple of 64. Value range: 0% (bad), 100% (ideal - no thread divergence).
- **VALUBusy**: The percentage of GPUTime vector ALU instructions are processed. Value range: 0% (bad) to 100% (optimal).
- **SALUBusy**: The percentage of GPUTime scalar ALU instructions are processed. Value range: 0% (bad) to 100% (optimal).
- **Mem32Bwrites**: FetchSize: The total kilobytes fetched from the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
• WriteSize: The total kilobytes written to the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
• L2CacheHit: The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache. Value range: 0% (no hit) to 100% (optimal).
• MemUnitBusy: The percentage of GPUTime the memory unit is active. The result includes the stall time (MemUnitStalled). This is measured with all extra fetches and writes and any cache or memory effects taken into account. Value range: 0% to 100% (fetch-bound).
• MemUnitStalled: The percentage of GPUTime the memory unit is stalled. Try reducing the number or size of fetches and writes if possible. Value range: 0% (optimal) to 100% (bad).
• WriteUnitStalled: The percentage of GPUTime the Write unit is stalled. Value range: 0% to 100% (bad).
• ALUStalledByLDS: The percentage of GPUTime ALU units are stalled by the LDS input queue being full or the output queue being not ready. If there are LDS bank conflicts, reduce them. Otherwise, try reducing the number of LDS accesses if possible. Value range: 0% (optimal) to 100% (bad).
• LDSBankConflict: The percentage of GPUTime LDS is stalled by bank conflicts. Value range: 0% (optimal) to 100% (bad).

2.16.14.1 ROC Profiler Library


HW specific low-level performance analysis interface for profiling of GPU compute applications. The profiling includes HW performance counters with complex performance metrics.

GitHub: https://github.com/ROCm-Developer-Tools/rocprofiler

Metrics

• The link to profiler default metrics XML specification.

API specification

• API specification at the GitHub.

To get sources

To clone ROC Profiler from GitHub:

git clone https://github.com/ROCm-Developer-Tools/rocprofiler

The library source tree:

```markdown
* bin
  * rocprof - Profiling tool run script
* doc - Documentation
* inc/rocprofiler.h - Library public API
* src - Library sources
  * core - Library API sources
  * util - Library utils sources
  * xml - XML parser
* test - Library test suite
  * tool - Profiling tool
    * tool.cpp - tool sources
    * metrics.xml - metrics config file
* ctrl - Test control
```
Build

Build environment:

```bash
export CMAKE_PREFIX_PATH=<path to hsa-runtime includes>:<path to hsa-runtime library>
export CMAKE_BUILD_TYPE=<debug|release>  # release by default
export CMAKE_DEBUG_TRACE=1  # to enable debug tracing
```

To Build with the current installed ROCm:

```bash
To build and install to /opt/rocm/rocprofiler
export CMAKE_PREFIX_PATH=/opt/rocm/include/hsa:/opt/rocm
cd ../rocprofiler
mkdir build
cd build
cmake ..
make
make install
```

Internal ‘simple_convolution’ test run script:

```bash
cd ../rocprofiler/build
./run.sh
```

To enable error messages logging to ‘/tmp/rocprofiler_log.txt’:

```bash
export ROCPROFILER_LOG=1
```

To enable verbose tracing:

```bash
export ROCPROFILER_TRACE=1
```

2.17 AMD ROCm ROCTracer

ROTracer library, Runtimes Generic Callback/Activity APIs. The goal of the implementation is to provide a generic
independent from specific runtime profiler to trace API and asynchronous activity.

The API provides functionality for registering the runtimes API callbacks and asynchronous activity records pool
support.

GitHub: https://github.com/ROCm-Developer-Tools/roctracer

**API specification**

- API specification at the GitHub.

**To get sources**

To clone ROC Tracer from GitHub:

```
git clone -b amd-master https://github.com/ROCm-Developer-Tools/roctracer
```

The library source tree:
2.17.1 Overview

AOMP is a scripted build of LLVM and supporting software. It has support for OpenMP target offload on AMD GPUs. Since AOMP is a clang/llvm compiler, it also supports GPU offloading with HIP, CUDA, and OpenCL.

Some sources to support OpenMP target offload on AMD GPUs have not yet been merged into the upstream LLVM trunk. However all sources used by AOMP are available in AOMP repositories. One of those repositories is a mirror of the LLVM monorepo llvm-project with a set of commits applied to a stable LLVM release branch.

The bin directory of this repository contains a README.md and build scripts needed to download, build, and install AOMP from source. In addition to the mirrored LLVM project repository, AOMP uses a number of open-source ROCm components. The build scripts will download, build, and install all components needed for AOMP. However, we recommend that you install the latest release of the debian or rpm package for AOMP described in the install section.
2.17.2 AOMP

Install

Platform Install Options:

- Ubuntu or Debian
- SUSE SLES-15-SP1
- RHEL 7
- Install Without Root
- Build and Install from release source tarball
- Development Source Build and Install

AOMP will install to /usr/lib/aomp. The AOMP environment variable will automatically be set to the install location. This may require a new terminal to be launched to see the change.

On Ubuntu 18.04 LTS (bionic beaver), run these commands:

```
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_Ubuntu1804_0.7-5_amd64.deb
sudo dpkg -i aomp_Ubuntu1804_0.7-5_amd64.deb
```

On Ubuntu 16.04, run these commands:

```
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_Ubuntu1604_0.7-5_amd64.deb
sudo dpkg -i aomp_Ubuntu1604_0.7-5_amd64.deb
```

The AOMP bin directory (which includes the standard clang and llvm binaries) is not intended to be in your PATH for typical operation.

**AMD KFD Driver**

These commands are for supported Debian-based systems and target only the rock_dkms core component. More information can be found [HERE](#).

```
echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"' | sudo tee /etc/udev/rules.d/70-kfd.rules
wget -qO - http://repo.radeon.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
echo 'deb [arch=amd64] http://repo.radeon.com/rocm/apt/debian/ xenial main' | sudo tee /etc/apt/sources.list.d/rocm.list
sudo apt update
sudo apt install rock-dkms
sudo reboot
sudo usermod -a -G video $USER
```

**NVIDIA CUDA Driver**

If you build AOMP with support for nvptx GPUs, you must first install CUDA 10. Note these instructions reference the install for Ubuntu 16.04.

**Download Instructions for CUDA (Ubuntu 16.04)**

Go to https://developer.nvidia.com/cuda-10.0-download-archive For Ubuntu 16.04, select Linux®, x86_64, Ubuntu, 16.04, deb(local) and then click Download. Note you can change these options for your specific distribution type. Navigate to the debian in your Linux® directory and run the following commands:
sudo dpkg -i cuda-repo-ubuntu1604-10-0-local-10.0.130-410.48_1.0-1_amd64.deb
sudo apt-key add /var/cuda-repo-10-0-local-10.0.130-410.48/7fa2af80.pub
sudo apt-get update
sudo apt-get install cuda

Depending on your system the CUDA install could take a very long time.

AOMP will install to /usr/lib/aomp. The AOMP environment variable will automatically be set to the install location. This may require a new terminal to be launched to see the change.

wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_SLES15_SP1-0.7-5.x86_64.rpm
sudo rpm -i aomp_SLES15_SP1-0.7-5.x86_64.rpm

Confirm AOMP environment variable is set:

echo $AOMP

Prerequisites

The ROCm kernel driver is required for AMD GPU support. Also, to control access to the ROCm device, a user group “video” must be created and users need to be added to this group.

AMD KFD DRIVER

Important Note: There is a conflict with the KFD when simultaneously running the GUI on SLES-15-SP1, which leads to unpredictable behavior when offloading to the GPU. We recommend using SLES-15-SP1 in text mode to avoid running both the KFD and GUI at the same time.

SUSE SLES-15-SP1 comes with kfd support installed. To verify this:

sudo dmesg | grep kfd
sudo dmesg | grep amdgpu

Set Group Access

echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"' | sudo tee /etc/udev/rules.d/70-kfd.rules
sudo reboot
sudo usermod -a -G video $USER

NVIDIA CUDA Driver

If you build AOMP with support for nvptx GPUs, you must first install CUDA 10.

Download Instructions for CUDA (SLES15)

Go to https://developer.nvidia.com/cuda-10.0-download-archive For SLES-15, select Linux®, x86_64, SLES, 15.0, rpm(local) and then click Download. Navigate to the rpm in your Linux® directory and run the following commands:

sudo rpm -i cuda-repo-sles15-10-0-local-10.0.130-410.48-1.0-1.x86_64.rpm
sudo zypper refresh
sudo zypper install cuda

If prompted, select the ‘always trust key’ option. Depending on your system the CUDA install could take a very long time.

Important Note: If using a GUI on SLES-15-SP1, such as gnome, the installation of CUDA may cause the GUI to fail to load. This seems to be caused by a symbolic link pointing to nvidia-libglx.so instead of xorg-libglx.so. This can be fixed by updating the symbolic link:
AOMP will install to /usr/lib/aomp. The AOMP environment variable will automatically be set to the install location. This may require a new terminal to be launched to see the change.

The installation may need the following dependency:

```bash
sudo yum install perl-Digest-MD5
```

### Download and Install

```bash
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_REDHAT_7-0.7-5.x86_64.rpm
sudo rpm -i aomp_REDHAT_7-0.7-5.x86_64.rpm
```

If CUDA is not installed the installation may cancel, to bypass this:

```bash
sudo rpm -i --nodeps aomp_REDHAT_7-0.7-5.x86_64.rpm
```

Confirm AOMP environment variable is set:

```bash
echo $AOMP
```

### Prerequisites

The ROCm kernel driver is required for AMD GPU support. Also, to control access to the ROCm device, a user group “video” must be created and users need to be added to this group.

#### AMD KFD Driver

```bash
sudo subscription-manager repos --enable rhel-server-rhscl-7-rpms
sudo subscription-manager repos --enable rhel-7-server-optional-rpms
sudo subscription-manager repos --enable rhel-7-server-extras-rpms
```

Install and setup Devtoolset-7

Devtoolset-7 is recommended, follow instructions 1-3 here: Note that devtoolset-7 is a Software Collections package, and it is not supported by AMD. [https://www.softwarecollections.org/en/scls/rhsc/devtoolset-7/](https://www.softwarecollections.org/en/scls/rhsc/devtoolset-7/)

```
Install dkms tool

```bash
sudo yum install -y epel-release
sudo yum install -y dkms kernel-headers-`uname -r` kernel-devel-`uname -r`
```

Create a /etc/yum.repos.d/rocm.repo file with the following contents:

```bash
[ROCM]
name=ROCM
dbaseurl=http://repo.radeon.com/rocm/yum/rpm
enabled=1
gpgcheck=0
```

Install rock-dkms

```bash
sudo yum install rock-dkms
```

2.17. AMD ROCm ROCTracer
Set Group Access

```bash
echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"' | sudo tee /etc/udev/rules.d/70-kfd.rules
sudo reboot
sudo usermod -a -G video $USER
```

NVIDIA CUDA Driver

To build AOMP with support for nvptx GPUs, you must first install CUDA 10. We recommend CUDA 10.0. CUDA 10.1 will not work until AOMP moves to the trunk development of LLVM 9. The CUDA installation is now optional.

Download Instructions for CUDA (CentOS/RHEL 7)

- Go to https://developer.nvidia.com/cuda-10.0-download-archive
- For SLES-15, select Linux®, x86_64, RHEL or CentOS, 7, rpm(local) and then click Download.
- Navigate to the rpm in your Linux® directory and run the following commands:

```bash
sudo rpm -i cuda-repo-rhel7-10-0-local-10.0.130-410.48-1.0-1.x86_64.rpm
sudo yum clean all
sudo yum install cuda
```

By default, the packages install their content to the release directory `/usr/lib/aomp_0.X-Y` and then a symbolic link is created at `/usr/lib/aomp` to the release directory. This requires root access.

Once installed go to TESTINSTALL for instructions on getting started with AOMP examples.

Debian

To install the debian package without root access into your home directory, you can run these commands. On Ubuntu 18.04 LTS (bionic beaver):

```bash
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_-_Ubuntu1804_0.7-5_amd64.deb
dpkg -x aomp_Ubuntu1804_0.7-5_amd64.deb /tmp/temproot
```

On Ubuntu 16.04:

```bash
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_-_Ubuntu1604_0.7-5_amd64.deb
dpkg -x aomp_Ubuntu1604_0.7-5_amd64.deb /tmp/temproot
```

```bash
mv /tmp/temproot/usr $HOME
export PATH=$PATH:$HOME/usr/lib/aomp/bin
export AOMP=$HOME/usr/lib/aomp
```

The last two commands could be put into your `.bash_profile` file so you can always access the compiler.

RPM

To install the rpm package without root access into your home directory, you can run these commands.

```bash
mkdir /tmp/temproot ; cd /tmp/temproot
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp_-_SLES15_SP1-0.7-5.x86_64.rpm
rpm2cpio aomp_SLES15_SP1-0.7-5.x86_64.rpm | cpio -idmv
mv /tmp/temproot/usr/lib $HOME
export PATH=$PATH:$HOME/rocm/aomp/bin
export AOMP=$HOME/rocm/aomp
```
The last two commands could be put into your .bash_profile file so you can always access the compiler.

The AOMP build and install from the release source tarball can be done manually or with spack. Building from source requires a number of platform dependencies. These dependencies are not yet provided with the spack configuration file. So if you are building from source either manually or building with spack, you must install the prerequisites for the platforms listed below.

Source Build Prerequisites

To build AOMP from source you must: 1. install certain distribution packages, 2. ensure the KFD kernel module is installed and operating, 3. create the Unix video group, and 4. install spack if required.

1. Required Distribution Packages

Debian or Ubuntu Packages

```bash
sudo apt-get install cmake g++-5 g++ pkg-config libpci-dev libnuma-dev libelf-dev
libffi-dev git python libopenmpi-dev gawk
```

SLES-15-SP1 Packages

```bash
sudo zypper install -y git pciutils-devel cmake python-base libffi-devel gcc gcc-c++
libnuma-devel libelf-devel patchutils openmpi2-devel
```

RHEL 7 Packages

Building from source requires a newer gcc. Devtoolset-7 is recommended, follow instructions 1-3 here: Note that devtoolset-7 is a Software Collections package, and it is not supported by AMD.


The build_aomp.sh script will automatically enable devtoolset-7 if found in /opt/rh/devtoolset-7/enable. If you want to build an individual component you will need to manually start devtoolset-7 from the instructions above.

```bash
sudo yum install cmake3 pciutils-devel numactl-devel libffi-devel
```

The build scripts use cmake, so we need to link cmake –> cmake3 in /usr/bin

```bash
sudo ln -s /usr/bin/cmake3 /usr/bin/cmake
```

2. Verify KFD Driver

Please verify you have the proper software installed as AOMP needs certain support to function properly, such as the KFD driver for AMD GPUs.

Debian or Ubuntu Support

```bash
wget -qO - http://repo.radeon.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
echo 'deb [arch=amd64] http://repo.radeon.com/rocm/apt/debian/ xenial main' | sudo
tee /etc/apt/sources.list.d/rocm.list
sudo apt update
sudo apt install rock-dkms
```

SUSE SLES-15-SP1 Support

Important Note: There is a conflict with the KFD when simultaneously running the GUI on SLES-15-SP1, which leads to unpredicatable behavior when offloading to the GPU. We recommend using SLES-15-SP1 in text mode to avoid running both the KFD and GUI at the same time.
SUSE SLES-15-SP1 comes with kfd support installed. To verify this:

```bash
sudo dmesg | grep kfd
sudo dmesg | grep amdgpu
```

### RHEL 7 Support

```bash
sudo subscription-manager repos --enable rhel-server-rhel-7-rpms
sudo subscription-manager repos --enable rhel-7-server-optional-rpms
sudo subscription-manager repos --enable rhel-7-server-extras-rpms
```

### Install dkms tool

```bash
sudo yum install -y epel-release
dsudo yum install -y dkms kernel-headers-`uname -r` kernel-devel-`uname -r`
```

Create a `/etc/yum.repos.d/rocm.repo` file with the following contents:

```ini
[ROCm]
name=ROCm
baseurl=http://repo.radeon.com/rocm/yum/rpm
enabled=1
gpgcheck=0
```

### Install rock-dkms

```bash
sudo yum install rock-dkms
```

3. Create the Unix Video Group

Regardless of Linux distribution, you must create a video group to contain the users authorized to use the GPU.

```bash
echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"' | sudo tee /etc/udev/rules.d/70-kfd.rules
sudo reboot
sudo usermod -a -G video $USER
```

4. Install spack

To use spack to build and install from the release source tarball, you must install spack first. Please refer to these install instructions for instructions on installing spack. Remember, the aomp spack configuration file is currently missing dependencies, so be sure to install the packages listed above before proceeding.

#### Build AOMP manually from release source tarball

To build and install aomp from the release source tarball run these commands:

```bash
wget https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp-0.7-5.tar.gz
tar -xzf aomp-0.7-5.tar.gz
cd aomp
nohup make &
```

Depending on your system, the last command could take a very long time. So it is recommended to use nohup and background the process. The simple Makefile that make will use runs build script “build_aomp.sh” and sets some flags to avoid git checks and applying ROCm patches. Here is that Makefile:
AOMP ?= /usr/local/aomp
AOMP_REPOS = $(shell pwd)
all:
    AOMP=$(AOMP) AOMP_REPOS=$(AOMP_REPOS) AOMP_CHECK_GIT_BRANCH=0 AOMP_APPLY_ROCM_PATCHES=0 $(AOMP_REPOS)/aomp/bin/build_aomp.sh

If you set the environment variable AOMP, the Makefile will install to that directory. Otherwise, the Makefile will install into /usr/local. So you must have authorization to write into /usr/local if you do not set the environment variable AOMP. Let’s assume you set the environment variable AOMP to “$HOME/rocm/aomp” in .bash_profile.
The build_aomp.sh script will install into $HOME/rocm/aomp and create a symbolic link from $HOME/rocm/aomp to $HOME/rocm/aomp_0.7-5. This feature allows multiple versions of AOMP to be installed concurrently. To enable a backlevel version of AOMP, simply set AOMP to $HOME/rocm/aomp_0.7-4.

Build AOMP with spack

Assuming you have installed the prerequisites listed above, use these commands to fetch the source and build aomp.
Currently the aomp configuration is not yet in the spack git hub so you must create the spack package first.

wget https://github.com/ROCm-Developer-Tools/aomp/blob/master/bin/package.py
spack create -n aomp -t makefile --force https://github.com/ROCm-Developer-Tools/aomp/releases/download/rel_0.7-5/aomp-0.7-5.tar.gz
spack edit aomp
spack install aomp

The “spack create” command will download and start an editor of a newly created spack config file. With the exception of the sha256 value, copy the contents of the downloaded package.py file into into the spack configuration file. You may restart this editor with the command “spack edit aomp”

Depending on your system, the “spack install aomp” command could take a very long time. Unless you set the AOMP environment variable, AOMP will be installed in /usr/local/aomp with a symbolic link from /usr/local/aomp to /usr/local/aomp. Be sure you have write access to /usr/local or set AOMP to a location where you have write access.

Build and install from sources is possible. However, the source build for AOMP is complex for several reasons.
• Many repos are required. The clone_aomp.sh script ensures you have all repos and the correct branch.
• Requires that Cuda SDK 10 is installed for NVIDIA GPUs. ROCm does not need to be installed for AOMP.
• It is a bootstrapped build. The built and installed LLVM compiler is used to build library components.
• Additional package dependencies are required that are not required when installing the AOMP package.

Source Build Prerequisites

1. Required Distribution Packages

Debian or Ubuntu Packages

sudo apt-get install cmake g++-5 g++ pkg-config libpci-dev libnuma-dev libelf-dev libffi-dev git python libopenmpi-dev gawk

SLES-15-SP1 Packages

sudo zypper install -y git pciutils-devel cmake python-base libffi-devel gcc gcc-c++ libnuma-devel libelf-devel patchutils openmp12-devel

RHEL 7 Packages

Building from source requires a newer gcc. Devtoolset-7 is recommended, follow instructions 1-3 here: Note that devtoolset-7 is a Software Collections package, and it is not supported by AMD.
The build_aomp.sh script will automatically enable devtoolset-7 if found in /opt/rh/devtoolset-7/enable. If you want to build an individual component you will need to manually start devtoolset-7 from the instructions above.

```
sudo yum install cmake piciutils-devel numactl-devel libffi-devel
```

The build scripts use cmake, so we need to link cmake -> cmake3 in /usr/bin

```
sudo ln -s /usr/bin/cmake3 /usr/bin/cmake
```

2. Verify KFD Driver

Please verify you have the proper software installed as AOMP needs certain support to function properly, such as the KFD driver for AMD GPUs.

**Debian or Ubuntu Support**

These commands are for supported Debian-based systems and target only the rock_dkms core component. More information can be found [HERE](#).

```
wget -qO - http://repo.radeon.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
echo 'deb [arch=amd64] http://repo.radeon.com/rocm/apt/debian/ xenial main' | sudo --tee /etc/apt/sources.list.d/rocm.list
sudo apt update
sudo apt install rock-dkms
```

**SUSE SLES-15-SP1 Support**

**Important Note:** There is a conflict with the KFD when simultaneously running the GUI on SLES-15-SP1, which leads to unpredictable behavior when offloading to the GPU. We recommend using SLES-15-SP1 in text mode to avoid running both the KFD and GUI at the same time.

SUSE SLES-15-SP1 comes with kfd support installed. To verify this:

```
sudo dmesg | grep kfd
sudo dmesg | grep amdgpu
```

**RHEL 7 Support**

```
sudo subscription-manager repos --enable rhel-server-rhscl-7-rpms
sudo subscription-manager repos --enable rhel-7-server-optional-rpms
sudo subscription-manager repos --enable rhel-7-server-extras-rpms
```

**Install dkms tool**

```
sudo yum install -y epel-release
sudo yum install -y dkms kernel-headers-`uname -r` kernel-devel-`uname -r`
```

Create a /etc/yum.repos.d/rocm.repo file with the following contents:

```ini
[ROCm]
name=ROCm
baseurl=http://repo.radeon.com/rocm/yum/rpm
enabled=1
gpgcheck=0
```

**Install rock-dkms**
sudo yum install rock-dkms

3. Create the Unix Video Group

echo 'SUBSYSTEM=="kfd", KERNEL=="kfd", TAG+="uaccess", GROUP="video"' | sudo tee /etc/udev/rules.d/70-kfd.rules
sudo reboot
sudo usermod -a -G video $USER

Clone and Build AOMP

cd $HOME ; mkdir -p git/aomp ; cd git/aomp
git clone https://github.com/rocm-developer-tools/aomp
cd $HOME/git/aomp/aomp/bin

Choose a Build Version (Development or Release) The development version is the next version to be released. It is possible that the development version is broken due to regressions that often occur during development. If instead, you want to build from the sources of a previous release such as 0.7-5 that is possible as well.

For the Development Branch:

git checkout master git pull

For the Release Branch:

git checkout rel_0.7-5 git pull

Clone and Build:

./clone_aomp.sh ./build_aomp.sh

Depending on your system, the last two commands could take a very long time. For more information, please refer to AOMP developers README.

You only need to do the checkout/pull in the AOMP repository. The file “bin/aomp_common_vars” lists the branches of each repository for a particular AOMP release. In the master branch of AOMP, aomp_common_vars lists the development branches. It is a good idea to run clone_aomp.sh twice after you checkout a release to be sure you pulled all the checkouts for a particular release.

For more information on Release Packages, click here

2.17.3 Test

Getting Started

The default install location is /usr/lib/aomp. To run the given examples, for example in /
/usr/lib/aomp/examples/openmp do the following:

Copy the example openmp directory somewhere writable

cd /usr/lib/aomp/examples/
cp -rp openmp /work/tmp/openmp-examples
cd /work/tmp/openmp-examples/vmulsum

Point to the installed AOMP by setting AOMP environment variable

export AOMP=/usr/lib/aomp

Make Instructions
Run ‘make help’ for more details.

View the OpenMP Examples README for more information.

### 2.17.4 AOMP Limitations

See the release notes in github. Here are some limitations.

- Dwarf debugging is turned off for GPUs. `-g` will turn on host level debugging only.
- Some simd constructs fail to vectorize on both host and GPUs.

### 2.17.4.1 ROCmValidationSuite

The ROCm Validation Suite (RVS) is a system administrator’s and cluster manager’s tool for detecting and troubleshooting common problems affecting AMD GPU(s) running in a high-performance computing environment, enabled using the ROCm software stack on a compatible platform.

The RVS is a collection of tests, benchmarks and qualification tools each targeting a specific sub-system of the ROCm platform. All of the tools are implemented in software and share a common command line interface. Each set of tests are implemented in a “module” which is a library encapsulating the functionality specific to the tool. The CLI can specify the directory containing modules to use when searching for libraries to load. Each module may have a set of options that it defines and a configuration file that supports its execution.

### 2.17.5 ROCmValidationSuite Modules

#### GPU Properties – GPUP

The GPU Properties module queries the configuration of a target device and returns the device’s static characteristics. These static values can be used to debug issues such as device support, performance and firmware problems.

#### GPU Monitor – GM module

The GPU monitor tool is capable of running on one, some or all of the GPU(s) installed and will report various information at regular intervals. The module can be configured to halt another RVS modules execution if one of the quantities exceeds a specified boundary value.

#### PCI Express State Monitor – PESM module?

The PCIe State Monitor tool is used to actively monitor the PCIe interconnect between the host platform and the GPU. The module will register a “listener” on a target GPU’s PCIe interconnect, and log a message whenever it detects a state change. The PESM will be able to detect the following state changes:

- PCIe link speed changes
- GPU power state changes

#### ROCm Configuration Qualification Tool - RCQT module

The ROCm Configuration Qualification Tool ensures the platform is capable of running ROCm applications and is configured correctly. It checks the installed versions of the ROCm components and the platform configuration of the system. This includes checking that dependencies, corresponding to the associated operating system and runtime environment, are installed correctly. Other qualification steps include checking:

- The existence of the `/dev/kfd` device
• The /dev/kfd device’s permissions
• The existence of all required users and groups that support ROCm
• That the user mode components are compatible with the drivers, both the KFD and the amdgpu driver.
• The configuration of the runtime linker/loader qualifying that all ROCm libraries are in the correct search path.

**PCI Express Qualification Tool – PEQT module**

The PCIe Qualification Tool consists is used to qualify the PCIe bus on which the GPU is connected. The qualification test will be capable of determining the following characteristics of the PCIe bus interconnect to a GPU:

- Support for Gen 3 atomic completers
- DMA transfer statistics
- PCIe link speed
- PCIe link width

**SBIOS Mapping Qualification Tool – SMQT module**

The GPU SBIOS mapping qualification tool is designed to verify that a platform’s SBIOS has satisfied the BAR mapping requirements for VDI and Radeon Instinct products for ROCm support. Refer to the “ROCm Use of Advanced PCIe Features and Overview of How BAR Memory is Used In ROCm Enabled System” web page for more information about how BAR memory is initialized by VDI and Radeon products.

**P2P Benchmark and Qualification Tool – PBQT module**

The P2P Benchmark and Qualification Tool is designed to provide the list of all GPUs that support P2P and characterize the P2P links between peers. In addition to testing for P2P compatibility, this test will perform a peer-to-peer throughput test between all P2P pairs for performance evaluation. The P2P Benchmark and Qualification Tool will allow users to pick a collection of two or more GPUs on which to run. The user will also be able to select whether or not they want to run the throughput test on each of the pairs.

Please see the web page “ROCm, a New Era in Open GPU Computing” to find out more about the P2P solutions available in a ROCm environment.

**PCI Express Bandwidth Benchmark – PEBB module**

The PCIe Bandwidth Benchmark attempts to saturate the PCIe bus with DMA transfers between system memory and a target GPU card’s memory. The maximum bandwidth obtained is reported to help debug low bandwidth issues. The benchmark should be capable of targeting one, some or all of the GPUs installed in a platform, reporting individual benchmark statistics for each.

**GPU Stress Test - GST module**

The GPU Stress Test runs a Graphics Stress test or SGEMM/DGEMM (Single/Double-precision General Matrix Multiplication) workload on one, some or all GPUs. The GPUs can be of the same or different types. The duration of the benchmark should be configurable, both in terms of time (how long to run) and iterations (how many times to run).

The test should be capable driving the power level equivalent to the rated TDP of the card, or levels below that. The tool must be capable of driving cards at TDP-50% to TDP-100%, in 10% incremental jumps. This should be controllable by the user.

**Input EDPP Test - IET module**

The Input EDPp Test generates EDP peak power on all input rails. This test is used to verify if the system PSU is capable of handling the worst case power spikes of the board. Peak Current at defined period = 1 minute moving average power.

Examples and about config files link.
2.17.6 Prerequisites

Ubuntu:

```
sudo apt-get -y update && sudo apt-get install -y libpci3 libpci-dev doxygen unzip cmake git
```

CentOS:

```
sudo yum install -y cmake3 doxygen pciutils-devel rpm rpm-build git gcc-c++
```

RHEL:

```
sudo yum install -y cmake3 doxygen rpm rpm-build git gcc-c++
wget http://mirror.centos.org/centos/7/os/x86_64/Packages/pciutils-devel-3.5.1-3.el7.x86_64.rpm
sudo rpm -ivh pciutils-devel-3.5.1-3.el7.x86_64.rpm
```

SLES:

```
sudo SUSEConnect -p sle-module-desktop-applications/15.1/x86_64
sudo SUSEConnect --product sle-module-development-tools/15.1/x86_64
sudo zypper install -y cmake doxygen pciutils-devel libpci3 rpm git rpm-build gcc-c++
```

2.17.7 Install ROCm stack, rocblas and rocm_smi64

Install ROCm stack for Ubuntu/CentOS, Refer https://github.com/RadeonOpenCompute/ROCm

Install rocBLAS and rocm_smi64:

Ubuntu:

```
sudo apt-get install rocblas rocm_smi64
```

CentOS & RHEL:

```
sudo yum install rocblas rocm_smi64
```

SUSE:

```
sudo zypper install rocblas rocm_smi64
```

Note: If rocm_smi64 is already installed but “/opt/rocm/rocm_smi” path doesn’t exist. Do below:

Ubuntu: `sudo dpkg -r rocm_smi64` & `sudo apt install rocm_smi64`

CentOS & RHEL: `sudo rpm -e rocm_smi64` & `sudo yum install rocm_smi64`

SUSE: `sudo rpm -e rocm_smi64` & `sudo zypper install rocm_smi64`
2.17.8 Building from Source

This section explains how to get and compile current development stream of RVS.

Clone repository

```
git clone https://github.com/ROCm-Developer-Tools/ROCmValidationSuite.git
```

Configure and build RVS:

```
cd ROCmValidationSuite
```

If OS is Ubuntu and SLES, use cmake

```
cmake ./ -B./build
make -C ./build
```

If OS is CentOS and RHEL, use cmake3

```
cmake3 ./ -B./build
make -C ./build
```

Build package:

```
cd ./build
make package
```

Note: based on your OS, only DEB or RPM package will be built. You may ignore an error for the unrelated configuration

Install package:

```
Ubuntu : sudo dpkg -i rocm-validation-suite*.deb
CentOS & RHEL & SUSE : sudo rpm -i --replacefiles --nodeps rocm-validation-suite*.rpm
```

Running RVS

```
cd ./build/bin
sudo ./rvs -d 3
sudo ./rvsqa.new.sh ; It will run complete rvs test suite
```

2.17.9 Regression

Regression is currently implemented for PQT module only. It comes in the form of a Python script run_regression.py.

The script will first create valid configuration files on $RVS_BUILD/regression folder. It is done by invoking prq_create_conf.py script to generate valid configuration files. If you need different tests, modify the prq_create_conf.py script to generate them.

Then, it will iterate through generated files and invoke RVS to specifying also JSON output and -d 3 logging level.

Finally, it will iterate over generated JSON output files and search for ERROR string. Results are written into $RVS_BUILD/regression/regression_res file.
Results are written into $RVS_BUILD/regression/

**Environment variables**

Before running the run_regression.py you first need to set the following environment variables for location of RVS source tree and build folders (adjust for your particular clone):

```bash
export WB=/work/yourworkfolder
export RVS=$WB/ROCmValidationSuite
export RVS_BUILD=$RVS/../build
```

**Running the script**

Just do:

```bash
cd $RVS/regression
./run_regression.py
```

---

### 2.17.9.1 ROCr Debug Agent

The ROCr Debug Agent is a library that can be loaded by ROCm Platform Runtime to provide the following functionality:

- Print the state of wavefronts that report memory violation or upon executing a s_trap 2 instruction.
- Allows SIGINT (\texttt{ctrl c}) or SIGTERM (\texttt{kill -15}) to print wavefront state of aborted GPU dispatches.
- It is enabled on Vega10 (since ROCm1.9), Vega20 (since ROCm2.0) GPUs.

### 2.17.10 Usage

To use the ROCr Debug Agent set the following environment variable:

```bash
export HSA_TOOLS_LIB=librocr_debug_agent64.so
```

This will use the ROCr Debug Agent library installed at /opt/rocm/lib/librocr_debug_agent64.so by default since the ROCm installation adds /opt/rocm/lib to the system library path. To use a different version set the LD_LIBRARY_PATH, for example:

```bash
export LD_LIBRARY_PATH=/path_to_directory_containing_librocr_debug_agent64.so
```

To display the machine code instructions of wavefronts, together with the source text location, the ROCr Debug Agent uses the \texttt{llvm-objdump} tool. Ensure that a version that supports AMD GCN GPUs is on your $PATH. For example, for ROCm 3.0:

```bash
export PATH=/opt/rocm/opencl/bin/x86_64/:$PATH
```

Execute your application.

If the application encounters a GPU error it will display the wavefront state of the GPU to stdout. Possible error states include:

- The GPU executes a memory instruction that causes a memory violation. This is reported as an XNACK error state.
- Queue error.
- The GPU executes an S_TRAP instruction. The \texttt{__builtin_trap()} language builtin can be used to generate a S_TRAP.
• A SIGINT (ctrl c) or SIGTERM (kill -15) signal is sent to the application while executing GPU code. Enabled by the ROCM_DEBUG_ENABLE_LINUX_SIGNALS environment variable.

For example, a sample print out for GPU memory fault is:

| Memory access fault by GPU agent: AMD gfx900 |
| Node: 1 |
| Address: 0x18DB4xxx (page not present; write access to a read-only page;)

64 wavefront(s) found in XNACK error state @PC: 0x0000001100E01310
printing the first one:

EXEC: 0xFFFFFFFFFFFFFFFF
STATUS: 0x00412460
TRAPSTS: 0x30000000
M0: 0x00001010

s0: 0x000C0000 s1: 0x80000010 s2: 0x10000000 s3: 0x00EA4FAC
s4: 0x17D78400 s5: 0x00000000 s6: 0x01039000 s7: 0x00000000
s8: 0x00000000 s9: 0x00000000 s10: 0x17D78400 s11: 0x04000000
s12: 0x00000000 s13: 0x00000000 s14: 0x00000000 s15: 0x00000000
s16: 0x0103C000 s17: 0x00000000 s18: 0x00000000 s19: 0x00000000
s20: 0x01037060 s21: 0x00000000 s22: 0x00000000 s23: 0x00000011
s24: 0x00004000 s25: 0x00010000 s26: 0x04C00000 s27: 0x00000010
s28: 0xFFFFFFFF s29: 0xFFFFFFFF s30: 0x00000000 s31: 0x00000000

Lane 0x0
v0: 0x00000003 v1: 0x18DB4400 v2: 0x18DB4400 v3: 0x00000000
v4: 0x00000000 v5: 0x00000000 v6: 0x00700000 v7: 0x00800000

Lane 0x1
v0: 0x00000004 v1: 0x18DB4400 v2: 0x18DB4400 v3: 0x00000000
v4: 0x00000000 v5: 0x00000000 v6: 0x00700000 v7: 0x00800000

Lane 0x2
v0: 0x00000005 v1: 0x18DB4400 v2: 0x18DB4400 v3: 0x00000000
v4: 0x00000000 v5: 0x00000000 v6: 0x00700000 v7: 0x00800000

Lane 0x3
v0: 0x00000006 v1: 0x18DB4400 v2: 0x18DB4400 v3: 0x00000000
v4: 0x00000000 v5: 0x00000000 v6: 0x00700000 v7: 0x00800000

Faulty Code Object:
Disassembly of section .text:

```
the_kernel:
; /home/qingchuan/tests/faulty_test/vector_add_kernel.cl:12
; d[100000000] = ga[gid & 31];
    v_mov_b32_e32 v1, v2 // 0000000012F0:
      → 7E020302
    v_mov_b32_e32 v4, v3 // 0000000012F4:
      → 7E080303
    v_add_i32_e32 v1, vcc, s10, v1 // 0000000012F8:
      → 3202020A
    v_mov_b32_e32 v5, s22 // 0000000012FC:
      → 7E0A0216
    v_addc_u32_e32 v4, vcc, v4, v5, vcc // 000000001300:
      → 38080B04
    v_mov_b32_e32 v2, v1 // 000000001304:
      → 7E040301
    v_mov_b32_e32 v3, v4 // 000000001308:
      → 7E060304
    s_waitcnt lgkmcnt(0) // 00000000130C:
      → BF8CC07F
    flat_store_dword v[2:3], v0 // 000000001310:
      → DC700000 00000002
; /home/qingchuan/tests/faulty_test/vector_add_kernel.cl:13
; }
    s_endpgm // 000000001318:
      → BF810000
```

Faulty PC offset: 1310

Aborted (core dumped)

### 2.17.11 Options

By default the wavefront dump is sent to stdout.

To save to a file use:

```
export ROCM_DEBUG_WAVE_STATE_DUMP=file
```

This will create a file called ROCm_Wave_State_Dump in code object directory (see below).

To return to the default stdout use either of the following:

```
export ROCM_DEBUG_WAVE_STATE_DUMP=stdout
unset ROCM_DEBUG_WAVE_STATE_DUMP
```

The following environment variable can be used to enable dumping wavefront states when SIGINT (ctrl c) or SIGTERM (kill -15) is sent to the application:

```
export ROCM_DEBUG_ENABLE_LINUX_SIGNALS=1
```

Either of the following will disable this behavior:
When the ROCr Debug Agent is enabled, each GPU code object loaded by the ROCm Platform Runtime will be saved in a file in the code object directory. By default the code object directory is `/tmp/ROCM_Tmp_PID_XXXX/` where `XXXX` is the application process ID. The code object directory can be specified using the following environment variable:

```
export ROCM_DEBUG_SAVE_CODE_OBJECT=code_object_directory
```

This will use the path `/code_object_directory`.

Loaded code objects will be saved in files named `ROCM_Code_Object_N` where `N` is a unique integer starting at 0 of the order in which the code object was loaded.

If the default code object directory is used, then the saved code object file will be deleted when it is unloaded with the ROCm Platform Runtime, and the complete code object directory will be deleted when the application exits normally. If a code object directory path is specified then neither the saved code objects, nor the code object directory will be deleted.

To return to using the default code object directory use:

```
unset ROCM_DEBUG_SAVE_CODE_OBJECT
```

By default ROCr Debug Agent logging is disabled. It can be enabled to display to `stdout` using:

```
export ROCM_DEBUG_ENABLE_AGENTLOG=stdout
```

Or to a file using:

```
export ROCM_DEBUG_ENABLE_AGENTLOG=<filename>
```

Which will write to the file `<filename>_AgentLog_PID_XXXX.log`.

To disable logging use:

```
unset ROCM_DEBUG_ENABLE_AGENTLOG
```

### 2.17.11.1 HCC: Heterogeneous Compute Compiler

**HCC**: An open source C++ compiler for heterogeneous devices

This repository hosts the HCC compiler implementation project. The goal is to implement a compiler that takes a program that conforms to a parallel programming standard such as HC, C++ 17 ParallelSTL and transforms it into the AMD GCN ISA.

### 2.17.12 Deprecation Notice

AMD is deprecating HCC to put more focus on HIP development and on other languages supporting heterogeneous compute. We will no longer develop any new feature in HCC and we will stop maintaining HCC after its final release, which is planned for June 2019. If your application was developed with the hc C++ API, we would encourage you to transition it to other languages supported by AMD, such as HIP or OpenCL. HIP and hc language share the same compiler technology, so many hc kernel language features (including inline assembly) are also available through the HIP compilation path.

The project is based on LLVM+CLANG. For more information, please visit HCCguide
The ability to write code in assembly is essential to achieving the best performance for a GPU program. In a previous blog we described how to combine several languages in a single program using ROCm and Hsaco. This article explains how to produce Hsaco from assembly code and also takes a closer look at some new features of the GCN architecture. I’d like to thank Ilya Perminov of Luxsoft for co-authoring this blog post. Programs written for GPUs should achieve the highest performance possible. Even carefully written ones, however, won’t always employ 100% of the GPU’s capabilities. Some reasons are the following:

- The program may be written in a high level language that does not expose all of the features available on the hardware.
- The compiler is unable to produce optimal ISA code, either because the compiler needs to ‘play it safe’ while adhering to the semantics of a language or because the compiler itself is generating un-optimized code.

Consider a program that uses one of GCN’s new features (source code is available on GitHub). Recent hardware architecture updates—DPP and DS Permute instructions—enable efficient data sharing between wavefront lanes. To become more familiar with the instruction set, review the GCN ISA Reference Guide. Note: the assembler is currently experimental; some of syntax we describe may change.

### 2.17.14 DS Permute Instructions

Two new instructions, ds_permute_b32 and ds_bpermute_b32, allow VGPR data to move between lanes on the basis of an index from another VGPR. These instructions use LDS hardware to route data between the 64 lanes, but they don’t write to LDS memory. The difference between them is what to index: the source-lane ID or the destination-lane ID. In other words, ds_permute_b32 says “put my lane data in lane i,” and ds_bpermute_b32 says “read data from lane i.” The GCN ISA Reference Guide provides a more formal description. The test kernel is simple: read the initial data and indices from memory into GPRs, do the permutation in the GPRs and write the data back to memory. An analogous OpenCL kernel would have this form:

```c
__kernel void hello_world(__global const uint * in, __global const uint * index, __global uint * out)
{
    size_t i = get_global_id(0);
    out[i] = in[index[i]];
}
```

### 2.17.15 Passing Parameters to a Kernel

Formal HSA arguments are passed to a kernel using a special read-only memory segment called kernarg. Before a wavefront starts, the base address of the kernarg segment is written to an SGPR pair. The memory layout of variables in kernarg must employ the same order as the list of kernel formal arguments, starting at offset 0, with no padding between variables—except to honor the requirements of natural alignment and any align qualifier. The example host program must create the kernarg segment and fill it with the buffer base addresses. The HSA host code might look like the following:

```c
/*
 * This is the host-side representation of the kernel arguments that the simplePermute_kernel expects.
 */
struct simplePermute_args_t {
    uint32_t * in;
    /* (continues on next page) */
```
The host program should also allocate memory for the in, index and out buffers. In the GitHub repository, all the run-time-related stuff is hidden in the Dispatch and Buffer classes, so the sample code looks much cleaner:

```c
// Create Kernarg segment
if (!AllocateKernarg(3 * sizeof(void*))) { return false; }

// Create buffers
Buffer *in, *index, *out;
in = AllocateBuffer(size);
index = AllocateBuffer(size);
out = AllocateBuffer(size);

// Fill Kernarg memory
Kernarg(in); // Add base pointer to "in" buffer
Kernarg(index); // Append base pointer to "index" buffer
Kernarg(out); // Append base pointer to "out" buffer
```

Initial Wavefront and Register State To launch a kernel in real hardware, the run time needs information about the kernel, such as

- The LDS size
- The number of GPRs
- Which registers need initialization before the kernel starts

All this data resides in the amd_kernel_code_t structure. A full description of the structure is available in the AMDGPU-ABI specification. This is what it looks like in source code:

```
.hsa_code_object_version 2,0
.hsa_code_object_isa 8, 0, 3, "AMD", "AMDGPU"

.text
.p2align 8
.amdgp_hsa_kernel hello_world

hello_world:
```
Currently, a programmer must manually set all non-default values to provide the necessary information. Hopefully, this situation will change with new updates that bring automatic register counting and possibly a new syntax to fill that structure. Before the start of every wavefront execution, the GPU sets up the register state on the basis of the enable_sgpr_* and enable_vgpr_* flags. VGPR v0 is always initialized with a work-item ID in the x dimension. Registers v1 and v2 can be initialized with work-item IDs in the y and z dimensions, respectively. Scalar GPRs can be initialized with a work-group ID and work-group count in each dimension, a dispatch ID, and pointers to kernarg, the aql packet, the aql queue, and so on. Again, the AMDGPU-ABI specification contains a full list in in the section on initial register state. For this example, a 64-bit base kernarg address will be stored in the s[0:1] registers (enable_sgpr_kernarg_segment_ptr = 1), and the work-item thread ID will occupy v0 (by default). Below is the scheme showing initial state for our kernel.

![]( ROCm_Tools/initial_state-768x387.png)
The next amd_kernel_code_t fields are obvious: is_ptr64 = 1 says we are in 64-bit mode, and kernarg_segment_byte_size = 24 describes the kernarg segment size. The GPR counting is less straightforward, however. The workitem_vgpr_count holds the number of vector registers that each work item uses, and wavefront_sgpr_count holds the number of scalar registers that a wavefront uses. The code above employs v0–v4, so workitem_vgpr_count = 5. But wavefront_sgpr_count = 8 even though the code only shows s0–s5, since the special registers VCC, FLAT_SCRATCH and XNACK are physically stored as part of the wavefront’s SGPRs in the highest-numbered SGPRs. In this example, FLAT_SCRATCH and XNACK are disabled, so VCC has only two additional registers. In current GCN3 hardware, VGPRs are allocated in groups of 4 registers and SGPRs in groups of 16. Previous generations (GCN1 and GCN2) have a VGPR granularity of 4 registers and an SGPR granularity of 8 registers. The fields compute_pgm_rsrc1_*gprs contain a device-specific number for each register-block type to allocate for a wavefront. As we said previously, future updates may enable automatic counting, but for now you can use following formulas for all three GCN GPU generations:

\[
\text{compute_pgm_rsrc1_vgprs} = \frac{(\text{workitem_vgpr_count}-1)}{4}
\]
\[
\text{compute_pgm_rsrc1_sgprs} = \frac{(\text{wavefront_sgpr_count}-1)}{8}
\]

Now consider the corresponding assembly:

```c
// initial state:
// s[0:1] - kernarg base address
// v0 - workitem id
s_load_dwordx2 s[4:5], s[0:1], 0x10 // load out_ptr into s[4:5] from kernarg
s_load_dwordx4 s[0:3], s[0:1], 0x00 // load in_ptr into s[0:1] and index_ptr into
       --s[2:3] from kernarg
v_lshlrev_b32 v0, 2, v0 // v0 *= 4;
s_waitcnt lgkmcnt(0) // wait for memory reads to finish

// compute address of corresponding element of index buffer
// i.e. v[1:2] = &index[workitem_id]
v_add_u32 v1, vcc, s2, v0
v_mov_b32 v2, s3
v_addc_u32 v2, vcc, v2, 0, vcc

// compute address of corresponding element of in buffer
// i.e. v[3:4] = &in[workitem_id]
v_add_u32 v3, vcc, s0, v0
v_mov_b32 v4, s1
v_addc_u32 v4, vcc, v4, 0, vcc
flat_load_dword v1, v[1:2] // load index[workitem_id] into v1
s_waitcnt vmcnt(0) & lgkmcnt(0) // wait for memory reads to finish

// v1 *= 4; ds_bpermute_b32 uses byte offset and registers are dwords
v_lshlrev_b32 v1, 2, v1

// perform permutation
// temp[thread_id] = v2
// v1 = temp[v1]
// effectively we got v1 = in[index[thread_id]]
ds_bpermute_b32 v1, v1, v2

// compute address of corresponding element of out buffer
```

(continues on next page)
// i.e. v[3:4] = s[out[workitem_id]]
v_add_u32 v3, vcc, s4, v0
v_mov_b32 v2, s5
v_addc_u32 v4, vcc, v2, 0, vcc
s_waitcnt lgkmcnt(0) // wait for permutation to finish

// store final value in out buffer, i.e. out[workitem_id] = v1
flat_store_dword v[3:4], v1
s_endpgm

2.17.17 Compiling GCN ASM Kernel Into Hsaco

The next step is to produce a Hsaco from the ASM source. LLVM has added support for the AMDGCN assembler, so you can use Clang to do all the necessary magic:

```
clang -x assembler -target amdgcn--amdhsa -mcpu=fiji -c -o test.o asm_source.s
clang -target amdgcn--amdhsa test.o -o test.co
```

The first command assembles an object file from the assembly source, and the second one links everything (you could have multiple source files) into a Hsaco. Now, you can load and run kernels from that Hsaco in a program. The GitHub examples use Cmake to automatically compile ASM sources. In a future post we will cover DPP, another GCN cross-lane feature that allows vector instructions to grab operands from a neighboring lane.

2.17.18 Overview

This repository contains the following useful items related to AMDGPU ISA assembler:

- amdphdrs: utility to convert ELF produced by llvm-mc into AMD Code Object (v1)
- examples/asm-kernel: example of AMDGPU kernel code
- examples/gfx8/ds_bpermute: transfer data between lanes in a wavefront with ds_bpermute_b32
- examples/gfx8/dpp_reduce: calculate prefix sum in a wavefront with DPP instructions
- examples/gfx8/s_memrealtime: use s_memrealtime instruction to create a delay
- examples/gfx8/s_memrealtime_inline: inline assembly in OpenCL kernel version of s_memrealtime
- examples/api/assemble: use LLVM API to assemble a kernel
- examples/api/disassemble: use LLVM API to disassemble a stream of instructions
- bin/sp3_to_mc.pl: script to convert some AMD sp3 legacy assembler syntax into LLVM MC
- examples/sp3: examples of sp3 convertible code

At the time of this writing (February 2016), LLVM trunk build and latest ROCR runtime is needed.

LLVM trunk (May or later) now uses llc as linker and produces AMD Code Object (v2).
2.17.19 Building

Top-level CMakeLists.txt is provided to build everything included. The following CMake variables should be set:

- **HSA_DIR** (default /opt/hsa/bin): path to ROCR Runtime
- **LLVM_DIR**: path to LLVM build directory

To build everything, create build directory and run cmake and make:

```
mkdir build
cd build
cmake -DLLVM_DIR=/srv/git/llvm.git/build ..
make
```

Examples that require clang will only be built if clang is built as part of llvm.

2.17.20 Use cases

Assembling to code object with llvm-mc from command line

The following llvm-mc command line produces ELF object `asm.o` from assembly source `asm.s`:

```
llvm-mc -arch=amdgcn -mcpu=fiji -filetype=obj -o asm.o asm.s
```

Assembling to raw instruction stream with llvm-mc from command line

It is possible to extract contents of `.text` section after assembling to code object:

```
llvm-mc -arch=amdgcn -mcpu=fiji -filetype=obj -o asm.o asm.s
objdump -h asm.o | grep .text | awk '{print "dd if='asm.o' of='asm' bs=1 count=${0x"...
|$3 "} skip=${0x "$6 "}"}'} | bash
```

Disassembling code object from command line

The following command line may be used to dump contents of code object:

```
llvm-objdump -disassemble -mcpu=fiji asm.o
```

This includes text disassembly of `.text` section.

Disassembling raw instruction stream from command line

The following command line may be used to disassemble raw instruction stream (without ELF structure):

```
hexdump -v -e '/1 "0x%02X "' asm | llvm-mc -arch=amdgcn -mcpu=fiji -disassemble
```

Here, hexdump is used to display contents of file in hexadecimal (0x.. form) which is then consumed by llvm-mc.
2.17.21 Assembling source into code object using LLVM API

Refer to examples/api/assemble.

2.17.22 Disassembling instruction stream using LLVM API

Refer to examples/api/disassemble.

Using amdphdrs

Note that normally standard lld and Code Object version 2 should be used which is closer to standard ELF format. amdphdrs (now obsolete) is complimentary utility that can be used to produce AMDGPU Code Object version 1. For example, given assembly source in asm.s, the following will assemble it and link using amdphdrs:

```bash
llvm-mc -arch=amdgcn -mcpu=fiji -filetype=obj -o asm.o asm.s
amdphdrs asm.o asm.co
```

2.17.23 Differences between LLVM AMDGPU Assembler and AMD SP3 assembler

Macro support

SP3 supports proprietary set of macros/tools. sp3_to_mc.pl script attempts to translate them into GAS syntax understood by llvm-mc. flat_atomic_cmpswap instruction has 32-bit destination

LLVM AMDGPU:

```
flat Atomic_cmpswap v7, v[9:10], v[7:8]
```

SP3:

```
flat Atomic_cmpswap v[7:8], v[9:10], v[7:8]
```

Atomic instructions that return value should have glc flag explicitly

LLVM AMDGPU:

```
flat Atomic_swap_x2 v[0:1], v[0:1], v[2:3] glc
```

SP3:

```
flat Atomic_swap_x2 v[0:1], v[0:1], v[2:3]
```

2.17.24 References

- LLVM Use Guide for AMDGPU Back-End
- AMD ISA Documents
  - AMD GCN3 Instruction Set Architecture (2016)
  - AMD_Southern_Islands_Instruction_Set_Architecture
2.18 AMD ROCm Debugger

The AMD ROCm Debugger (ROCgdb) is the AMD ROCm source-level debugger for Linux based on the GNU Debugger (GDB). It enables heterogeneous debugging on the AMD ROCm platform of an x86-based host architecture along with AMD GPU architectures and supported by the AMD Debugger API Library (ROCdbgapi). The AMD ROCm Debugger is installed by the rocm-gdb package. The rocm-gdb package is part of the rocm-dev meta-package, which is in the rocm-dkms package.

The current AMD ROCm Debugger (ROCgdb) is an initial prototype that focuses on source line debugging. Note, symbolic variable debugging capabilities are not currently supported. You can use the standard GDB commands for both CPU and GPU code debugging. For more information about ROCgdb, refer to the ROCgdb User Guide, which is installed at:

- /opt/rocm/share/info/gdb.info as a texinfo file
- /opt/rocm/share/doc/gdb/gdb.pdf as a PDF file

The AMD ROCm Debugger User Guide is available as a PDF at:

https://github.com/RadeonOpenCompute/ROCm/blob/master/gdb.pdf

For more information about GNU Debugger (GDB), refer to the GNU Debugger (GDB) web site at:

http://www.gnu.org/software/gdb

2.19 AMD ROCm API Library

The amd-dbgapi library implements an AMD GPU debugger application programming interface (API). It provides the support necessary for a client of the library to control the execution and inspect the state of supported commercially available AMD GPU devices.

The AMD ROCm Debugger API Library (ROCdbgapi) implements an AMD GPU debugger application programming interface (API) that provides the support necessary for a client of the library to control the execution and inspect the state of AMD GPU devices.

The following AMD GPU architectures are supported:

- Vega 10
- Vega 7nm

The AMD ROCm Debugger API Library is installed by the rocm-dbgapi package. The rocm-gdb package is part of the rocm-dev meta-package, which is in the rocm-dkms package.

The AMD ROCm Debugger API Specification is available as a PDF at:


2.20 System Level Debug

2.20.1 ROCm Language & System Level Debug, Flags and Environment Variables

Kernel options to avoid Ethernet port getting renamed every time you change graphics cards
net.ifnames=0 biosdevname=0
2.20.1.1 ROCr

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Invalid Dimension</td>
</tr>
<tr>
<td>4</td>
<td>Invalid Group Memory</td>
</tr>
<tr>
<td>8</td>
<td>Invalid (or Null) Code</td>
</tr>
<tr>
<td>32</td>
<td>Invalid Format</td>
</tr>
<tr>
<td>64</td>
<td>Group is too large</td>
</tr>
<tr>
<td>128</td>
<td>Out of VGPR’s</td>
</tr>
<tr>
<td>0x80000000</td>
<td>Debug Trap</td>
</tr>
</tbody>
</table>

2.20.1.2 Command to dump firmware version and get Linux Kernel version

- sudo cat /sys/kernel/debug/dri/1/amdgpu_firmware_info
- uname -a

2.20.1.3 Debug

Debug messages when developing/debugging base ROCm driver. You could enable the printing from libhsakmt.so by setting an environment variable, HSAKMT_DEBUG_LEVEL. Available debug levels are 3~7. The higher level you set, the more messages will print.

- export HSAKMT_DEBUG_LEVEL=3 : only pr_err() will print.
- export HSAKMT_DEBUG_LEVEL=4 : pr_err() and pr_warn() will print.
- export HSAKMT_DEBUG_LEVEL=5 : We currently don’t implement “notice”. Setting to 5 is same as setting to 4.
- export HSAKMT_DEBUG_LEVEL=6 : pr_err(), pr_warn(), and pr_info will print.
- export HSAKMT_DEBUG_LEVEL=7 : Everything including pr_debug will print.

2.20.1.4 ROCr level env variable for debug

- HSA_ENABLE_SDMA=0
- HSA_ENABLE_INTERRUPT=0
- HSA_SVM_GUARD_PAGES=0
- HSA_DISABLE_CACHE=1
### 2.20.1.5 Turn Off Page Retry on GFX9/Vega devices

- `sudo -s`
- `echo 1 > /sys/module/amdkfd/parameters/noretry`

### 2.20.1.6 HCC Debug Environment Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCC_PRINT_ENV=1</td>
<td>will print usage and current values for the HCC and HIP env variables.</td>
</tr>
<tr>
<td>HCC_PRINT_ENV = 1</td>
<td>Print values of HCC environment variables</td>
</tr>
<tr>
<td>HCC_SERIALIZE_KERNEL= 0</td>
<td>0x1=pre-serialize before each kernel launch, 0x2=post-serialize after each kernel launch, 0x3=both</td>
</tr>
<tr>
<td>HCC_SERIALIZE_COPY= 0</td>
<td>0x1=pre-serialize before each data copy, 0x2=post-serialize after each data copy, 0x3=both</td>
</tr>
<tr>
<td>HCC_DB = 0</td>
<td>Enable HCC trace debug</td>
</tr>
<tr>
<td>HCC_OPT_FLUSH = 1</td>
<td>Perform system-scope acquire/release only at CPU sync boundaries (rather than after each kernel)</td>
</tr>
<tr>
<td>HCC_MAX_QUEUES= 20</td>
<td>Set max number of HSA queues this process will use. acceleration_views will share allotted queues and steal from each other as necessary</td>
</tr>
<tr>
<td>HCC_UNPINNED_COPY_MODE = 2</td>
<td>Select algorithm for unpinned copies. 0=ChooseBest(see thresholds), 1=PinInPlace, 2=StagingBuffer, 3=Memcpy</td>
</tr>
<tr>
<td>HCC_CHECK_COPY = 0</td>
<td>Check dst == src after each copy operation. Only works on large-bar systems.</td>
</tr>
<tr>
<td>HCC_H2D_STAGING_THRESHOLD = 64</td>
<td>Min size (in KB) to use staging buffer algorithm for H2D copy if ChooseBest algorithm selected</td>
</tr>
<tr>
<td>HCC_H2D_PININPLACE_THRESHOLD = 4096</td>
<td>Min size (in KB) to use pin-in-place algorithm for H2D copy if ChooseBest algorithm selected</td>
</tr>
<tr>
<td>HCC_D2H_PININPLACE_THRESHOLD = 1024</td>
<td>Min size (in KB) to use pin-in-place for D2H copy if ChooseBest algorithm selected</td>
</tr>
<tr>
<td>HCC_PROFILE = 0</td>
<td>Enable HCC kernel and data profiling. 1=summary, 2=trace</td>
</tr>
<tr>
<td>HCC_PROFILE_VERBOSE = 31</td>
<td>Bitmark to control profile verbosity and format. 0x1=default, 0x2=show begin/end, 0x4=show barrier</td>
</tr>
</tbody>
</table>

### 2.20. System Level Debug

605
## 2.20.1.7 HIP Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIP_PRINT_ENV=1</td>
<td>Print HIP environment variables.</td>
</tr>
<tr>
<td>HIP_LAUNCH_BLOCKING=0</td>
<td>Make HIP kernel launches ‘host-synchronous’, so they block until any kernel launches. Alias: CUDA_LAUNCH_BLOCKING</td>
</tr>
<tr>
<td>HIP_LAUNCH_BLOCKING_KERNELS=</td>
<td>Comma-separated list of kernel names to make host-synchronous, so they block until completed.</td>
</tr>
<tr>
<td>HIP_API_BLOCKING= 0</td>
<td>Make HIP APIs ‘host-synchronous’, so they block until completed. Impacts hipMemcpyAsync, hipMemsetAsync</td>
</tr>
<tr>
<td>HIP_HIDDEN_FREE_MEM= 256</td>
<td>Amount of memory to hide from the free memory reported by hipMemGetInfo, specified in MB. Impacts hipMemGetInfo</td>
</tr>
<tr>
<td>HIP_DB = 0</td>
<td>Print debug info. Bitmask (HIP_DB=0xff) or flags separated by ‘+’ (HIP_DB=api+sync+mem+copy)</td>
</tr>
<tr>
<td>HIP_TRACE_API=0</td>
<td>Trace each HIP API call. Print function name and return code to stderr as program executes.</td>
</tr>
<tr>
<td>HIP_TRACE_API_COLOR= green</td>
<td>Color to use for HIP_API. None/Red/Green/Yellow/Blue/Magenta/Cyan/White</td>
</tr>
<tr>
<td>HIP_PROFILE_API = 0</td>
<td>Add HIP API markers to ATP file generated with CodeXL. 0x1=short API name, 0x2=full API name including args</td>
</tr>
<tr>
<td>HIP_DB_START_API =</td>
<td>Comma-separated list of tid.api_seq_num for when to start debug and profiling.</td>
</tr>
<tr>
<td>HIP_DB_STOP_API =</td>
<td>Comma-separated list of tid.api_seq_num for when to stop debug and profiling.</td>
</tr>
<tr>
<td>HIP_VISIBLE_DEVICES = 0</td>
<td>Only devices whose index is present in the sequence are visible to HIP applications and they are enumerated in the order of sequence</td>
</tr>
<tr>
<td>HIP_WAIT_MODE = 0</td>
<td>Force synchronization mode. 1= force yield, 2=force spin, 0=defaults specified in application</td>
</tr>
<tr>
<td>HIP_FORCE_P2P_HOST = 0</td>
<td>Force use of host/staging copy for peer-to-peer copies.1=always use copies, 2=always return false for hipDeviceCanAccessPeer</td>
</tr>
<tr>
<td>HIP_FORCE_SYNC_COPY = 0</td>
<td>Force all copies (even hipMemcpyAsync) to use sync copies</td>
</tr>
<tr>
<td>HIP_FAIL_SOC = 0</td>
<td>Fault on Sub-Optimal-Copy, rather than use a slower but functional implementation.Bit 0x1=Fail on async copy with unpinned memory. Bit 0x2=Fail peer copy rather than use staging buffer copy</td>
</tr>
</tbody>
</table>
2.20.1.8 OpenCL Debug Flags

- AMD_OCL_WAIT_COMMAND=1 (0 = OFF, 1 = On)

2.20.1.9 PCIe-Debug

Refer here for PCIe-Debug

More information here on how to debug and profile HIP applications

- HIP-Debugging
- HIP-Profiling

2.21 GCN ISA Manuals

2.21.1 GCN ISA Manual for Hawaii pdf

2.21.2 GCN ISA Manual for Fiji and Polaris pdf

2.21.3 Vega

- testdocbook

2.21.4 Inline GCN ISA Assembly Guide

The ability to write code in assembly is essential to achieving the best performance for a GPU program. In a previous blog we described how to combine several languages in a single program using ROCm and Hsaco. This article explains how to produce Hsaco from assembly code and also takes a closer look at some new features of the GCN architecture. I’d like to thank Ilya Perminov of Luxsoft for co-authoring this blog post. Programs written for GPUs should achieve the highest performance possible. Even carefully written ones, however, won’t always employ 100% of the GPU’s capabilities. Some reasons are the following:

- The program may be written in a high level language that does not expose all of the features available on the hardware.
- The compiler is unable to produce optimal ISA code, either because the compiler needs to ‘play it safe’ while adhering to the semantics of a language or because the compiler itself is generating un-optimized code.

Consider a program that uses one of GCN’s new features (source code is available on GitHub). Recent hardware architecture updates—DPP and DS Permute instructions—enable efficient data sharing between wavefront lanes. To become more familiar with the instruction set, review the GCN ISA Reference Guide. Note: the assembler is currently experimental; some of syntax we describe may change.
2.21.4.2 DS Permute Instructions

Two new instructions, ds_permute_b32 and ds_bpermute_b32, allow VGPR data to move between lanes on the basis of an index from another VGPR. These instructions use LDS hardware to route data between the 64 lanes, but they don’t write to LDS memory. The difference between them is what to index: the source-lane ID or the destination-lane ID. In other words, ds_permute_b32 says “put my lane data in lane i,” and ds_bpermute_b32 says “read data from lane i.” The GCN ISA Reference Guide provides a more formal description. The test kernel is simple: read the initial data and indices from memory into GPRs, do the permutation in the GPRs and write the data back to memory. An analogous OpenCL kernel would have this form:

```c
__kernel void hello_world(__global const uint * in, __global const uint * index, __global uint * out)
{
    size_t i = get_global_id(0);
    out[i] = in[index[i]];
}
```

2.21.4.3 Passing Parameters to a Kernel

Formal HSA arguments are passed to a kernel using a special read-only memory segment called kernarg. Before a wavefront starts, the base address of the kernarg segment is written to an SGPR pair. The memory layout of variables in kernarg must employ the same order as the list of kernel formal arguments, starting at offset 0, with no padding between variables—except to honor the requirements of natural alignment and any align qualifier. The example host program must create the kernarg segment and fill it with the buffer base addresses. The HSA host code might look like the following:

```c
/*
 * This is the host-side representation of the kernel arguments that the simplePermute_kernel expects.
 */
struct simplePermute_args_t {
    uint32_t * in;
    uint32_t * index;
    uint32_t * out;
};
/*
 * Allocate the kernel-argument buffer from the correct region.
 */
hsa_status_t status;
simplePermute_args_t * args = NULL;
status = hsa_memory_allocate(kernarg_region, sizeof(simplePermute_args_t), (void**)&args);
assert(HSA_STATUS_SUCCESS == status);
aql->kernarg_address = args;
/*
 * Write the args directly to the kernargs buffer;
 * the code assumes that memory is already allocated for the buffers that in_ptr, index_ptr and out_ptr point to
 */
args->in = in_ptr;
args->index = index_ptr;
args->out = out_ptr;
```

The host program should also allocate memory for the in, index and out buffers. In the GitHub repository, all the run-time-related stuff is hidden in the Dispatch and Buffer classes, so the sample code looks much cleaner:
// Create Kernarg segment
if (!AllocateKernarg(3 * sizeof(void*))) { return false; }

// Create buffers
Buffer *in, *index, *out;
in = AllocateBuffer(size);
index = AllocateBuffer(size);
out = AllocateBuffer(size);

// Fill Kernarg memory
Kernarg(in); // Add base pointer to "in" buffer
Kernarg(index); // Append base pointer to "index" buffer
Kernarg(out); // Append base pointer to "out" buffer

Initial Wavefront and Register State
To launch a kernel in real hardware, the run time needs information about the
kernel, such as

• The LDS size
• The number of GPRs
• Which registers need initialization before the kernel starts

All this data resides in the amd_kernel_code_t structure. A full description of the structure is available in
the AMDGPU-ABI specification. This is what it looks like in source code:

```
.hsa_code_object_version 2,0
.hsa_code_object_isa 8, 0, 3, "AMD", "AMDGPU"

.text
.p2align 8
.amdgpu_hsa_kernel hello_world

hello_world:

.enable_sgpr_kernarg_segment_ptr = 1
.is_ptr64 = 1
.computepgm_rsrc1_vgprs = 1
.computepgm_rsrc1_sgprs = 0
.computepgm_rsrc2_user_sgpr = 2
.kernarg_segment_byte_size = 24
.wavefront_sgpr_count = 8
.workitem_vgpr_count = 5

.s_load_dwordx2 s[4:5], s[0:1], 0x10
.s_load_dwordx4 s[0:3], s[0:1], 0x00
.v_lshlrev_b32 v0, 2, v0
.s_waitcnt lgkmcnt(0)
.v_add_u32 v1, vcc, s2, v0
.v_mov_b32 v2, s3
.v_addc_u32 v2, vcc, v2, 0, vcc
.v_add_u32 v3, vcc, s0, v0
.v_mov_b32 v4, s1
.v_addc_u32 v4, vcc, v4, 0, vcc
.flat_load_dword v1, v[1:2]
.flat_load_dword v2, v[3:4]
.s_waitcnt vmcnt(0) & lgkmcnt(0)
```

(continues on next page)
Currently, a programmer must manually set all non-default values to provide the necessary information. Hopefully, this situation will change with new updates that bring automatic register counting and possibly a new syntax to fill that structure. Before the start of every wavefront execution, the GPU sets up the register state on the basis of the enable_sgpr_* and enable_vgpr_* flags. VGPR v0 is always initialized with a work-item ID in the x dimension. Registers v1 and v2 can be initialized with work-item IDs in the y and z dimensions, respectively. Scalar GPRs can be initialized with a work-group ID and work-group count in each dimension, a dispatch ID, and pointers to kernarg, the aql packet, the aql queue, and so on. Again, the AMDGPU-ABI specification contains a full list in in the section on initial register state. For this example, a 64-bit base kernarg address will be stored in the s[0:1] registers (enable_sgpr_kernarg_segment_ptr = 1), and the work-item thread ID will occupy v0 (by default). Below is the scheme showing initial state for our kernel.

2.21.4.4 The GPR Counting

The next amd_kernel_code_t fields are obvious: is_ptr64 = 1 says we are in 64-bit mode, and kernarg_segment_byte_size = 24 describes the kernarg segment size. The GPR counting is less straightforward, however. The workitem_vgpr_count holds the number of vector registers that each work item uses, and wavefront_sgpr_count holds the number of scalar registers that a wavefront uses. The code above employs v0–v4, so workitem_vgpr_count = 5. But wavefront_sgpr_count = 8 even though the code only shows s0–s5, since the special registers VCC, FLAT_SCRATCH and XNACK are physically stored as part of the wavefront’s SGPRs in the highest-numbered SGPRs. In this example, FLAT_SCRATCH and XNACK are disabled, so VCC has only two additional registers. In current GCN3 hardware, VGPRs are allocated in groups of 4 registers and SGPRs in groups of 16. Previous generations (GCN1 and GCN2) have a VGPR granularity of 4 registers and an SGPR granularity of 8 registers. The fields compute_pgm_rsrc1_*gprs contain a device-specific number for each register-block type to
allocate for a wavefront. As we said previously, future updates may enable automatic counting, but for now you can use following formulas for all three GCN GPU generations:

```c
compute_pgm_rsrc1_vgprs = (workitem_vgpr_count-1)/4
compute_pgm_rsrc1_sgprs = (wavefront_sgpr_count-1)/8
```

Now consider the corresponding assembly:

```assembly
// initial state:
// s[0:1] - kernarg base address
// v0 - workitem id
s_load_dwordx2 s[4:5], s[0:1], 0x10 // load out_ptr into s[4:5] from kernarg
s_load_dwordx4 s[0:3], s[0:1], 0x00 // load in_ptr into s[0:1] and index_ptr into...

v_lshlrev_b32 v0, 2, v0 // v0 *= 4;
s_waitcnt lgkmcnt(0) // wait for memory reads to finish

// compute address of corresponding element of index buffer
// i.e. v[1:2] = &index[workitem_id]
v_add_u32 v1, vcc, s2, v0
v_mov_b32 v2, s3
v_addc_u32 v2, vcc, v2, 0, vcc

v_add_u32 v3, vcc, s0, v0
v_mov_b32 v4, s1
v_addc_u32 v4, vcc, v4, 0, vcc

flat_load_dword v1, v[1:2] // load index[workitem_id] into v1
s_waitcnt vmcnt(0) & lgkmcnt(0) // wait for permutation to finish

// v1 *= 4; ds_bpermute_b32 uses byte offset and registers are dwords
v_lshlrev_b32 v1, 2, v1

// perform permutation
// temp[thread_id] = v2
// v1 = temp[v1]
// effectively we got v1 = in[index[thread_id]]
ds_bpermute_b32 v1, v1, v2

// compute address of corresponding element of out buffer
// i.e. v[3:4] = &out[workitem_id]
v_add_u32 v3, vcc, s4, v0
v_mov_b32 v4, s5
v_addc_u32 v4, vcc, v4, 0, vcc
s_waitcnt lgkmcnt(0) // wait for permutation to finish

// store final value in out buffer, i.e. out[workitem_id] = v1
flat_store_dword v[3:4], v1

s_endpgm
```
2.21.4.5 Compiling GCN ASM Kernel Into Hsaco

The next step is to produce a Hsaco from the ASM source. LLVM has added support for the AMDGCN assembler, so you can use Clang to do all the necessary magic:

```
clang -x assembler -target amdgcn--amdhsa -mcpu=fiji -c -o test.o asm_source.s
clang -target amdgcn--amdhsa test.o -o test.co
```

The first command assembles an object file from the assembly source, and the second one links everything (you could have multiple source files) into a Hsaco. Now, you can load and run kernels from that Hsaco in a program. The GitHub examples use Cmake to automatically compile ASM sources. In a future post we will cover DPP, another GCN cross-lane feature that allows vector instructions to grab operands from a neighboring lane.

2.22 Remote Device Programming

2.22.1 ROCmRDMA

Peer-to-Peer bridge driver for PeerDirect - Deprecated Repo

This is now included as part of the ROCK Kernel Driver ROCmRDMA is the solution designed to allow third-party kernel drivers to utilize DMA access to the GPU memory. It allows direct path for data exchange (peer-to-peer) using the standard features of PCI Express.

Currently ROCmRDMA provides the following benefits:

- Direct access to ROCm memory for 3rd party PCIe devices
- Support for PeerDirect(c) interface to offloads the CPU when dealing with ROCm memory for RDMA network stacks;

2.22.1.1 Restrictions and limitations

To fully utilize ROCmRDMA the number of limitation could apply impacting either performance or functionality in the whole:

- It is recommended that devices utilizing ROCmRDMA share the same upstream PCI Express root complex. Such limitation depends on PCIe chipset manufactures and outside of GPU controls;
- To provide peer-to-peer DMA access all GPU local memory must be exposed via PCI memory BARs (so called large-BAR configuration);
- It is recommended to have IOMMU support disabled or configured in pass-through mode due to limitation in Linux kernel to support local PCIe device memory for any form transition others then 1:1 mapping.
2.22.1.2 ROCmRDMA interface specification

The implementation of ROCmRDMA interface could be found in [amd_rdma.h] file.

2.22.1.3 Data structures

```c
/*
 * Structure describing information needed to P2P access from another device
 * to specific location of GPU memory
 *
 * Structure providing function pointers to support rdma/p2p requirements.
 * to specific location of GPU memory
 */

struct amd_p2p_info {
    uint64_t va;  /* Specify user virt. address */
    uint64_t size; /* Specify total size of allocation */
    struct pid *pid; /* Specify process pid to which virtual address belongs */
    struct sg_table *pages; /* Specify DMA/Bus addresses */
    void *priv; /* Pointer set by AMD kernel driver */
};

/*
 * Structure providing function pointers to support rdma/p2p requirements.
 * to specific location of GPU memory
 */

struct amd_rdma_interface {
    int (*get_pages)(uint64_t address, uint64_t length, struct pid *pid,
        struct amd_p2p_info **amd_p2p_data,
        void (*free_callback)(void *client_priv),
        void *client_priv);
    int (*put_pages)(struct amd_p2p_info **amd_p2p_data);
    int (*is_gpu_address)(uint64_t address, struct pid *pid);
    int (*get_page_size)(uint64_t address, uint64_t length, struct pid *pid,
        unsigned long *page_size);
};
```
/**
 * amdkfd_query_rdma_interface - Return interface (function pointers table) for rdma interface
 * 
 * \param interface - OUT: Pointer to interface
 * \return 0 if operation was successful.
 */
int amdkfd_query_rdma_interface(const struct amd_rdma_interface **rdma);

/**
 * amdkfd_query_rdma_interface - Return interface (function pointers table) for rdma interface
 * \param interface - OUT: Pointer to interface
 * \return 0 if operation was successful.
 */
int amdkfd_query_rdma_interface(const struct amd_rdma_interface **rdma);

/**
 * This function makes the pages underlying a range of GPU virtual memory accessible for DMA operations from another PCIe device
 * \param address - The start address in the Unified Virtual Address space in the specified process
 * \param length - The length of requested mapping
 * \param pid - Pointer to structure pid to which address belongs. Could be NULL for current process address space.
 * \param p2p_data - On return: Pointer to structure describing underlying pages/locations
 * \param free_callback - Pointer to callback which will be called when access to such memory must be stopped immediately: Memory was freed, GECC events, etc.
 * \param client_priv - Pointer to be passed as parameter on 'free_callback;
 * \return 0 if operation was successful
 */
int get_pages(uint64_t address, uint64_t length, struct pid *pid,
              struct amd_p2p_info **amd_p2p_data,
              void (*free_callback)(void *client_priv),
              void *client_priv);
2.22.2 UCX

What is UCX?

Unified Communication X (UCX) is a communication library for building Message Passing (MPI), PGAS/OpenSHMEM libraries and RPC/data-centric applications. UCX utilizes high-speed networks for inter-node and shared memory mechanisms for intra-node communication. For more information, visit http://openucx.github.io/ucx/

How to install UCX with ROCm?

See How to install UCX and OpenMPI

2.22.3 MPI

OpenMPI and OpenSHMEM installation

1. Get latest-and-greatest OpenMPI version:

   $ git clone https://github.com/open-mpi/ompi.git

2. Autogen:

   $ cd ompi
   $ ./autogen.pl

3. Configure with UCX

   $ mkdir build
   $ cd build
   $ ./configure --prefix=/your_install_path/ --with-ucx=/path_to_ucx_installation

4. Build:

   $ make
   $ make install
Running Open MPI with UCX

Example of the command line (for InfiniBand RC + shared memory):

```
$ mpirun -np 2 -mca pml ucx -x UCX_NET_DEVICES=mlx5_0:1 -x UCX_TLS=rc,sm ./app
```

Open MPI runtime optimizations for UCX

- By default OpenMPI enables build-in transports (BTLs), which may result in additional software overheads in the OpenMPI progress function. In order to workaround this issue you may try to disable certain BTLs.

```
$ mpirun -np 2 -mca pml ucx --mca btl ^vader,tcp,openib -x UCX_NET_DEVICES=mlx5_0:1 -x UCX_TLS=rc,sm ./app
```

- OpenMPI version https://github.com/open-mpi/ompi/commit/066370202dcad8e302f2baf8921e9ef0f1f7dfc leverages more efficient timer mechanism and therefore reduces software overheads in OpenMPI progress.

MPI and OpenSHMEM release versions tested with UCX master

1. UCX current tarball: https://github.com/openucx/ucx/archive/master.zip
2. The table of MPI and OpenSHMEM distributions that are tested with the HEAD of UCX master

<table>
<thead>
<tr>
<th>MPI/OpenSHMEM</th>
<th>project</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMPI/OSHMEM</td>
<td>2.1.0</td>
</tr>
<tr>
<td>MPICH</td>
<td>Latest</td>
</tr>
</tbody>
</table>

### 2.22.4 IPC

#### 2.22.4.1 Introduction

#### 2.22.4.1.1 IPC API

New datatypes

```c
hsa_amd_ipc_memory_handle_t

/** IPC memory handle to be passed from one process to another */
typedef struct hsa_amd_ipc_memory_handle_s {
  uint64_t handle;
} hsa_amd_ipc_memory_handle_t;

hsa_amd_ipc_signal_handle_t

/** IPC signal handle to be passed from one process to another */
typedef struct hsa_amd_ipc_signal_handle_s {
  uint64_t handle;
} hsa_amd_ipc_signal_handle_t;
```

Memory sharing API

Allows sharing of HSA allocated memory between different processes.

```c
hsa_amd_ipc_get_memory_handle
```

The purpose of this API is to get / export an IPC handle for an existing allocation from pool.
hsa_status_t HSA_API

hsa_amd_ipc_get_memory_handle(void *ptr, hsa_amd_ipc_memory_handle_t *ipc_handle);
where:
   IN: ptr - Pointer to memory previously allocated via hsa_amd_memory_pool_allocate() call
   OUT: ipc_handle - Unique IPC handle to be used in IPC.
       Application must pass this handle to another process.

hsa_amd_ipc_close_memory_handle
Close IPC memory handle previously received via “hsa_amd_ipc_get_memory_handle()” call.

hsa_status_t HSA_API

hsa_amd_ipc_close_memory_handle(hsa_amd_ipc_memory_handle_t ipc_handle);
where:
   IN: ipc_handle - IPC Handle to close

hsa_amd_ipc_open_memory_handle
Open / import an IPC memory handle exported from another process and return address to be used in the current
process.

hsa_status_t HSA_API

hsa_amd_ipc_open_memory_handle(hsa_amd_ipc_memory_handle_t ipc_handle, void **ptr);
where:
   IN: ipc_handle - IPC Handle
   OUT: ptr- Address which could be used in the given process for access to the memory

Client should call hsa_amd_memory_pool_free() when access to this resource is not needed any more.

Signal sharing API

Allows sharing of HSA signals between different processes.

hsa_amd_ipc_get_signal_handle
The purpose of this API is to get / export an IPC handle for an existing signal.

hsa_status_t HSA_API

hsa_amd_ipc_get_signal_handle(hsa_signal_t signal, hsa_amd_ipc_signal_handle_t *ipc_handle);
where:
IN: signal - Signal handle created as the result of hsa_signal_create() call.
OUT: ipc_handle - Unique IPC handle to be used in IPC.
   Application must pass this handle to another process.

hsa_amd_ipc_close_signal_handle
Close IPC signal handle previously received via “hsa_amd_ipc_get_signal_handle()” call .

hsa_status_t HSA_API

hsa_amd_ipc_close_signal_handle(hsa_amd_ipc_signal_handle_t ipc_handle);
where:
   IN: ipc_handle - IPC Handle to close

hsa_amd_ipc_open_signal_handle
Open / import an IPC signal handle exported from another process and return address to be used in the current
process.

hsa_status_t HSA_API

hsa_amd_ipc_open_signal_handle(hsa_amd_ipc_signal_handle_t ipc_handle, hsa_signal_t &signal);
where:
   IN: ipc_handle - IPC Handle
   OUT: signal - Signal handle to be used in the current process

   Client should call hsa_signal_destroy() when access to this resource is not needed any more.

Query API

Query memory information

Allows query information about memory resource based on address. It is partially overlapped with the following
requirement Memory info interface so it may be possible to merge those two interfaces.

typedef enum hsa_amd_address_info_s {
    /* Return uint32_t / boolean if address was allocated via HSA stack */
    HSA_AMD_ADDRESS_HSA_ALLOCATED = 0x1,

    /* Return agent where such memory was allocated */
    HSA_AMD_ADDRESS_AGENT = 0x2,

    /* Return pool from which this address was allocated */
    HSA_AMD_ADDRESS_POOL = 0x3,

    /* Return size of allocation */
}
HSA_AMD_ADDRESS_ALLOC_SIZE = 0x4

{ hsa_amd_address_info_t;

hsa_status_t HSA_API

hsa_amd_get_address_info(void ptr, hsa_amd_address_info_t attribute, void value);

where:

ptr - Address information about which to query
attribute - Attribute to query

2.23 Tutorial

• caffe How use Caffe on ROCm
• Vector-Add example using the HIP Programming Language
• mininbody This sample demonstrates the use of the HIP API for a mini n-body problem.
• GCN-asm-tutorial Assembly Sample The Art of AMDGCN Assembly: How to Bend the Machine to Your Will. This tutorial demonstrates GCN assembly with ROCm application development.
• Optimizing-Dispatches ROCm With Rapid Harmony: Optimizing HSA Dispatch: This tutorial shows how to optimize HSA dispatch performance for ROCm application development.
• rocnloc ROCm With Harmony: Combining OpenCL Kernels, HCC and HSA in a Single Program. This tutorial demonstrates how to compile OpenCL kernels using the CL offline compiler (CLOC) and integrate them with HCC C++ compiled ROCm applications.
• The AMD GCN Architecture - A Crash Course, by Layla Mah
• AMD GCN Architecture White paper
• ROCm-MultiGPU

2.24 ROCm

**host, host cpu**: Executes the HIP runtime API and is capable of initiating kernel launches to one or more devices.

**default device**: Each host thread maintains a default device. Most HIP runtime APIs (including memory allocation, copy commands, kernel launches) do not use an explicit device argument but instead implicitly use the default device. The default device can be set with hipSetDevice.

**active host thread**: The thread which is running the HIP APIs

**completion_future**: becomes ready, “Completes”.

**HIP-Clang**: Heterogeneous AMDGPU Compiler, with its capability to compile HIP programs on AMD platform ([https://github.com/RadeonOpenCompute/llvm-project](https://github.com/RadeonOpenCompute/llvm-project)).

**ROCclr**: a virtual device interface that compute runtimes interact with different backends such as ROCr on Linux or PAL on Windows.

**The ROCclr** ([https://github.com/ROCm-Developer-Tools/ROCclr](https://github.com/ROCm-Developer-Tools/ROCclr)) is an abstraction layer allowing runtimes to work on both OS effort.
**hipify tools** - tools to convert CUDA(R) code to portable C++ code

**hipconfig** - tool to report various configuration properties of the target platform.

**nvcc** - nvcc compiler, do not capitalize.

**ROCr ROCm runtime** The HSA runtime is a thin, user-mode API that exposes the necessary interfaces to access and interact with graphics hardware driven by the AMDGPU driver set and the ROCK kernel driver. Together they enable programmers to directly harness the power of AMD discrete graphics devices by allowing host applications to launch compute kernels directly to the graphics hardware.

**HCC (Heterogeneous Compute Compiler)**: HCC is an Open Source, Optimizing C++ Compiler for Heterogeneous Compute. It supports heterogeneous offload to AMD APUs and discrete GPUs via HSA enabled runtimes and drivers. It is based on Clang, the LLVM Compiler Infrastructure and the ‘libc++’ C++ standard library. The goal is to implement a compiler that takes a program that conforms to a parallel programming standard such as C++ AMP, HC, C++ 17 ParallelSTL, or OpenMP, and transforms it into the AMD GCN ISA.

In the AMD ROCM v3.5 release, the HCC compiler is deprecated and HIP-Clang compiler is introduced for compiling HIP programs
(https://github.com/RadeonOpenCompute/hcc)

**Accelerator Modes Supported:**

- HC C++ API
- HIP
- C++AMP
- C++ Parallel STL
- OpenMP

**HIP (Heterogeneous Interface for Portability)**: Heterogeneous Interface for Portability is a C++ runtime API and kernel language that allows developers to create portable applications that can run on AMD and other GPU’s. It provides a C-style API and a C++ kernel language. The first big feature available in the HIP is porting apps that use the CUDA Driver API.

**OpenCL**: Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators.

OpenCL provides a standard interface for parallel computing using task- and data-based parallelism. The programming language that is used to write compute kernels is called OpenCL C and is based on C99,[16] but adapted to fit the device model in OpenCL. OpenCL consists of a set of headers and a shared object that is loaded at runtime. As of 2016 OpenCL runs on Graphics processing units, CPUs with SIMD instructions, FPGAs, Movidius Myriad 2, Adapteva epiphany and DSPs.

**PCle Platform Atomics**: PCI Express (PCIe) was developed as the next generation I/O system interconnect after PCI, designed to enable advanced performance and features in connected devices while remaining compatible with the PCI software environment. Today, atomic transactions are supported for synchronization without using an interrupt mechanism. In emerging applications where math co-processing, visualization and content processing are required, enhanced synchronization would enable higher performance.

**Queue**: A Queue is a runtime-allocated resource that contains a packet buffer and is associated with a packet processor. The packet processor tracks which packets in the buffer have already been processed. When it has been informed by the application that a new packet has been enqueued, the packet processor is able to process it because the packet format is standard and the packet contents are self-contained – they include all the necessary information to run a command. A queue has an associated set of high-level operations defined in “HSA Runtime Specification” (API functions in host code) and “HSA Programmer Reference Manual Specification” (kernel code).
**HSA (Heterogeneous System Architecture)**: HSA provides a unified view of fundamental computing elements. HSA allows a programmer to write applications that seamlessly integrate CPUs (called latency compute units) with GPUs (called throughput compute units), while benefiting from the best attributes of each. HSA creates an improved processor design that exposes the benefits and capabilities of mainstream programmable compute elements, working together seamlessly. HSA is all about delivering new, improved user experiences through advances in computing architectures that deliver improvements across all four key vectors: improved power efficiency; improved performance; improved programmability; and broad portability across computing devices. For more on HSA.

**AQL Architectured Queuing Language**: The Architected Queuing Language (AQL) is a standard binary interface used to describe commands such as a kernel dispatch. An AQL packet is a user-mode buffer with a specific format that encodes one command. AQL allows agents to build and enqueue their own command packets, enabling fast, low-power dispatch. AQL also provides support for kernel agent queue submissions: the kernel agent kernel can write commands in AQL format.
Symbols

__STDC_WANT_IEC_60559_TYPES_EXT__ (C macro), 236

G
get_rocblas_status_for_hip_status (C++ function), 241

H
hipStream_t (C++ type), 236

M
MAT_BUFF_MAX_BYTES (C++ member), 241
MATRIX_DIM_X (C++ member), 241
MATRIX_DIM_Y (C++ member), 241

N
NB_X (C++ member), 241

O
open_log_stream (C++ function), 240

R
rocalution::_rocalution_sync (C++ function), 463
rocalution::AIChebyshev (C++ class), 513
rocalution::AIChebyshev::Set (C++ function), 513
rocalution::allocate_host (C++ function), 460
rocalution::AS (C++ class), 515
rocalution::AS::Set (C++ function), 515
rocalution::BaseAMG (C++ class), 509
rocalution::BaseAMG::BuildHierarchy (C++ function), 509
rocalution::BaseAMG::BuildSmoothers (C++ function), 510
rocalution::BaseAMG::ClearLocal (C++ function), 509
rocalution::BaseAMG::GetNumLevels (C++ function), 510
rocalution::BaseAMG::SetCoarsestLevel (C++ function), 510
rocalution::BaseAMG::SetDefaultSmootherFormat (C++ function), 510
rocalution::BaseAMG::SetManualSmoothers (C++ function), 510
rocalution::BaseAMG::SetManualSolver (C++ function), 510
rocalution::BaseAMG::SetOperatorFormat (C++ function), 510
rocalution::BaseMultiGrid (C++ class), 508
rocalution::BaseMultiGrid::InitLevels (C++ function), 508
rocalution::BaseMultiGrid::SetCycle (C++ function), 508
rocalution::BaseMultiGrid::SetHostLevels (C++ function), 508
rocalution::BaseMultiGrid::SetKcycleFull (C++ function), 508
rocalution::BaseMultiGrid::SetOperatorHierarchy (C++ function), 508
rocalution::BaseMultiGrid::SetProlongOperator (C++ function), 508
rocalution::BaseMultiGrid::SetRestrictOperator (C++ function), 508
rocalution::BaseMultiGrid::SetScaling (C++ function), 508
rocalution::BaseMultiGrid::SetSmoothing (C++ function), 508
rocalution::BaseMultiGrid::SetSmoothersPostIter (C++ function), 508
rocalution::BaseMultiGrid::SetSmoothersPreIter (C++ function), 508
rocalution::BaseMultiGrid::SetSolver (C++ function), 508
rocalution::BaseRocalution (C++ class), 463
rocalution::BaseRocalution::Clear (C++ function), 464
rocalution::BaseRocalution::Info (C++ function), 463
rocalution::BaseRocalution::MoveToAccelerator (C++ function), 463
rocalution::BaseRocalution::MoveToAcceleratorAsync (C++ function), 463
rocalution::BaseRocalution::MoveToHost (C++ function), 463
rocalution::BaseRocalution::MoveToHostAsync (C++ function), 463
rocalution::BaseRocalution::Sync (C++ function), 463
rocalution::BiCGStab (C++ class), 505
rocalution::BiCGStab1 (C++ class), 505
rocalution::BiCGStab1::SetOrder (C++ function), 505
rocalution::BlockJacobi (C++ class), 516
rocalution::BlockJacobi::Set (C++ function), 516
rocalution::BlockPreconditioner (C++ class), 516
rocalution::BlockPreconditioner::Set (C++ function), 516
rocalution::BlockPreconditioner::SetDiagonalSolver (C++ function), 516
rocalution::BlockPreconditioner::SetExternalLastMatrix (C++ function), 516
rocalution::BlockPreconditioner::SetLSolver (C++ function), 516
rocalution::BlockPreconditioner::SetPermutation (C++ function), 516
rocalution::CG (C++ class), 506
rocalution::Chebyshev (C++ class), 505
rocalution::Chebyshev::Set (C++ function), 505
rocalution::CR (C++ class), 506
rocalution::DiagJacobiSaddlePointPrecond (C++ class), 521
rocalution::DiagJacobiSaddlePointPrecond::Set (C++ function), 522
rocalution::DirectLinearSolver (C++ class), 512
rocalution::disable_accelerator_rocalution (C++ function), 462
rocalution::FCG (C++ class), 506
rocalution::FGMRES (C++ class), 507
rocalution::FGMRES::SetBasisSize (C++ function), 507
rocalution::FixedPoint (C++ class), 504
rocalution::FixedPoint::SetRelaxation (C++ function), 504
rocalution::free_host (C++ function), 460
rocalution::FSAI (C++ class), 513
rocalution::FSAI::SetPrecondMatrixFormat (C++ function), 514
rocalution::GlobalMatrix (C++ class), 493
rocalution::GlobalMatrix::AllocateCOO (C++ function), 493
rocalution::GlobalMatrix::AllocateCSR (C++ function), 493
rocalution::GlobalMatrix::Check (C++ function), 493
rocalution::GlobalMatrix::CloneFrom (C++ function), 494
rocalution::GlobalMatrix::CoarsenOperator (C++ function), 495
rocalution::GlobalMatrix::ConvertTo (C++ function), 494
rocalution::GlobalMatrix::ConvertToBCSR (C++ function), 494
rocalution::GlobalMatrix::ConvertToCOO (C++ function), 494
rocalution::GlobalMatrix::ConvertToCSR (C++ function), 494
rocalution::GlobalMatrix::ConvertToDENSE (C++ function), 494
rocalution::GlobalMatrix::ConvertToELL (C++ function), 494
rocalution::GlobalMatrix::ConvertToHYB (C++ function), 494
rocalution::GlobalMatrix::ConvertToMCSR (C++ function), 494
rocalution::GlobalMatrix::CopyFrom (C++ function), 494
rocalution::GlobalMatrix::ExtractInverseDiagonal (C++ function), 495
rocalution::GlobalMatrix::FurtherPairwiseAggregation (C++ function), 495
rocalution::GlobalMatrix::InitialPairwiseAggregation (C++ function), 495
rocalution::GlobalMatrix::LeaveDataPtrCOO (C++ function), 494
rocalution::GlobalMatrix::LeaveDataPtrCSR (C++ function), 494
rocalution::GlobalMatrix::LeaveGhostDataPtrCOO (C++ function), 494
rocalution::GlobalMatrix::LeaveGhostDataPtrCSR (C++ function), 494
rocalution::GlobalMatrix::LeaveLocalDataPtrCOO (C++ function), 494
rocalution::GlobalMatrix::LeaveLocalDataPtrCSR (C++ function), 494
rocalution::GlobalMatrix::ReadFileCSR (C++ function), 495
rocalution::GlobalMatrix::ReadFileMTX (C++ function), 495
rocalution::GlobalMatrix::Scale (C++ function), 495
rocalution::GlobalMatrix::SetDataPtrCOO (C++ function), 493
rocalution::GlobalMatrix::SetDataPtrCSR (C++ function), 493
rocalution::GlobalMatrix::SetPrecondMatrixFormat (C++ function), 493
rocalution::GlobalMatrix::SetDataPtrCOO (C++ function), 493
rocalution::GlobalMatrix::SetDataPtrCSR (C++ function), 493
rocalution::GlobalMatrix::SetDataPtrCSR (C++ function), 493
rocalution::GlobalMatrix::SetGhostDataPtrCOO (C++ function), 494
rocalution::GlobalMatrix::SetDataPtrCSR (C++ function), 493
rocalution::GlobalMatrix::SetGhostDataPtrCSR (C++ function), 493
rocalution::GlobalMatrix::SetLocalDataPtrCOO (C++ function), 493
rocalution::GlobalMatrix::SetLocalDataPtrCSR (C++ function), 493
rocalution::GlobalMatrix::SetParallelManager (C++ function), 493
rocalution::GlobalMatrix::Sort (C++ function), 495
rocalution::GlobalMatrix::WriteFileCSR (C++ function), 495
rocalution::GlobalMatrix::WriteFileMTX (C++ function), 495
rocalution::GlobalPairwiseAMG (C++ class), 511
rocalution::GlobalPairwiseAMG::SetBeta (C++ function), 512
rocalution::GlobalPairwiseAMG::SetCoarseningFactor (C++ function), 512
rocalution::GlobalPairwiseAMG::SetOrdering (C++ function), 512
rocalution::GlobalVector (C++ class), 498
rocalution::GlobalVector::Allocate (C++ function), 498
rocalution::GlobalVector::LeaveDataPtr (C++ function), 499
rocalution::GlobalVector::Prolongation (C++ function), 499
rocalution::GlobalVector::Restriction (C++ function), 499
rocalution::GlobalVector::SetDataPtr (C++ function), 499
rocalution::GlobalVector::SetParallelManager (C++ function), 498
rocalution::GMRES (C++ class), 506
rocalution::GMRES::SetBasisSize (C++ function), 507
rocalution::GS (C++ class), 517
rocalution::IC (C++ class), 518
rocalution::IDR (C++ class), 507
rocalution::IDR::SetRandomSeed (C++ function), 507
rocalution::IDR::SetShadowSpace (C++ function), 507
rocalution::ILU (C++ class), 517
rocalution::ILUT (C++ class), 518
rocalution::info_rocalution (C++ function), 462
rocalution::LocalMatrix::AMGAggregate
    ((C++ function), 491)
rocalution::LocalMatrix::AMGAggregation
    ((C++ function), 491)
rocalution::LocalMatrix::AMGConnect
    ((C++ function), 491)
rocalution::LocalMatrix::AMGSmoothedAggregation
    ((C++ function), 491)
rocalution::LocalMatrix::Check
    ((C++ function), 473)
rocalution::LocalMatrix::CloneFrom
    ((C++ function), 487)
rocalution::LocalMatrix::CMK
    ((C++ function), 482)
rocalution::LocalMatrix::CoarsenOperator
    ((C++ function), 492)
rocalution::LocalMatrix::Compress
    ((C++ function), 490)
rocalution::LocalMatrix::ConnectivityOrder
    ((C++ function), 483)
rocalution::LocalMatrix::ConvertTo
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToBCSR
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToCOO
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToCSR
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToDENSE
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToELL
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToHYB
    ((C++ function), 489)
rocalution::LocalMatrix::ConvertToMCSR
    ((C++ function), 489)
rocalution::LocalMatrix::CopyFrom
    ((C++ function), 487)
rocalution::LocalMatrix::CopyFromAsync
    ((C++ function), 487)
rocalution::LocalMatrix::CopyFromCOO
    ((C++ function), 488)
rocalution::LocalMatrix::CopyFromCSR
    ((C++ function), 488)
rocalution::LocalMatrix::CopyFromHostCSR
    ((C++ function), 488)
rocalution::LocalMatrix::CopyToCOO
    ((C++ function), 488)
rocalution::LocalMatrix::CopyToCSR
    ((C++ function), 488)
rocalution::LocalMatrix::CopyToHostCSR
    ((C++ function), 488)
rocalution::LocalMatrix::CopyToMCSR
    ((C++ function), 488)
rocalution::LocalMatrix::DiagonalMatrixMult
    ((C++ function), 490)
rocalution::LocalMatrix::DiagonalMatrixMultL
    ((C++ function), 491)
rocalution::LocalMatrix::DiagonalMatrixMultR
    ((C++ function), 490)
rocalution::LocalMatrix::ExtractColumnVector
    ((C++ function), 490)
rocalution::LocalMatrix::ExtractDiagonal
    ((C++ function), 482)
rocalution::LocalMatrix::ExtractInverseDiagonal
    ((C++ function), 482)
rocalution::LocalMatrix::ExtractL
    ((C++ function), 482)
rocalution::LocalMatrix::ExtractRowVector
    ((C++ function), 490)
rocalution::LocalMatrix::ExtractSubMatrices
    ((C++ function), 482)
rocalution::LocalMatrix::ExtractSubMatrix
    ((C++ function), 481)
rocalution::LocalMatrix::ExtractU
    ((C++ function), 482)
rocalution::LocalMatrix::FSAI
    ((C++ function), 491)
rocalution::LocalMatrix::Gershgorin
    ((C++ function), 490)
rocalution::LocalMatrix::GetFormat
    ((C++ function), 473)
rocalution::LocalMatrix::Householder
    ((C++ function), 485)
rocalution::LocalMatrix::ICFactorize
    ((C++ function), 484)
rocalution::LocalMatrix::ILU0Factorize
    ((C++ function), 484)
rocalution::LocalMatrix::ILUpFactorize
    ((C++ function), 484)
rocalution::LocalMatrix::ILUTFactorize
    ((C++ function), 484)
rocalution::LocalMatrix::Invert
    ((C++ function), 485)
rocalution::LocalMatrix::Key
    ((C++ function), 490)
rocalution::LocalMatrix::LAnalyse
    ((C++ function), 485)
rocalution::LocalMatrix::LAnalyseClear
    ((C++ function), 485)
rocalution::LocalMatrix::LeaveDataPtrCOO
    ((C++ function), 478)
rocalution::LocalMatrix::LeaveDataPtrCSR
    ((C++ function), 479)
rocalution::LocalMatrix::LeaveDataPtrDENSE
    ((C++ function), 481)
rocalution::LocalMatrix::LeaveDataPtrDIA
    ((C++ function), 480)
rocalution::LocalMatrix::LeaveDataPtrELL
    ((C++ function), 477)
rocalution::LocalMatrix::LeaveDataPtrMCSR  (C++ function), 479
rocalution::LocalMatrix::LLAnalyse  (C++ function), 481
rocalution::LocalMatrix::LLAnalyseClear  (C++ function), 481
rocalution::LocalMatrix::LSolve  (C++ function), 481
rocalution::LocalMatrix::LUAnalyse  (C++ function), 481
rocalution::LocalMatrix::LUAnalyseClear  (C++ function), 481
rocalution::LocalMatrix::LUFactorize  (C++ function), 481
rocalution::LocalMatrix::LUSolve  (C++ function), 481
rocalution::LocalMatrix::MatrixAdd  (C++ function), 481
rocalution::LocalMatrix::MatrixMult  (C++ function), 481
rocalution::LocalMatrix::MaximalIndependentSet  (C++ function), 481
rocalution::LocalMatrix::MultiColoring  (C++ function), 481
rocalution::LocalMatrix::Permute  (C++ function), 481
rocalution::LocalMatrix::PermuteBackward  (C++ function), 481
rocalution::LocalMatrix::QRDecompose  (C++ function), 481
rocalution::LocalMatrix::QRSolve  (C++ function), 481
rocalution::LocalMatrix::RCMK  (C++ function), 481
rocalution::LocalMatrix::ReadFileCSR  (C++ function), 481
rocalution::LocalMatrix::ReadFileMTX  (C++ function), 481
rocalution::LocalMatrix::ReplaceColumnVector  (C++ function), 481
rocalution::LocalMatrix::ReplaceRowVector  (C++ function), 481
rocalution::LocalMatrix::RugeStueben  (C++ function), 481
rocalution::LocalMatrix::Scale  (C++ function), 481
rocalution::LocalMatrix::ScaleDiagonal  (C++ function), 481
rocalution::LocalMatrix::ScaleOffDiagonal  (C++ function), 481
rocalution::LocalMatrix::SetDataPtrCOO  (C++ function), 481
rocalution::LocalMatrix::SetDataPtrCSR  (C++ function), 481
rocalution::LocalMatrix::SetDataPtrDENSE  (C++ function), 481
rocalution::LocalMatrix::SetDataPtrDIA  (C++ function), 481
rocalution::LocalMatrix::SetDataPtrELL  (C++ function), 481
rocalution::LocalMatrix::SetDataPtrMCSR  (C++ function), 481
rocalution::LocalMatrix::Sort  (C++ function), 481
rocalution::LocalMatrix::SPAI  (C++ function), 481
rocalution::LocalMatrix::SymbolicPower  (C++ function), 481
rocalution::LocalMatrix::Transpose  (C++ function), 481
rocalution::LocalMatrix::UAnalyse  (C++ function), 481
rocalution::LocalMatrix::UAnalyseClear  (C++ function), 481
rocalution::LocalMatrix::UpdateValuesCSR  (C++ function), 481
rocalution::LocalMatrix::ZeroBlockPermutation  (C++ function), 481
rocalution::LocalMatrix::Zeroes  (C++ function), 481
rocalution::LocalStencil::GetNDim  (C++ function), 481
rocalution::LocalStencil::SetGrid  (C++ function), 481
rocalution::LocalVector::Allocate  (C++ function), 481
rocalution::LocalVector::CopyFromData  (C++ function), 481
rocalution::LocalVector::CopyFromPermute  (C++ function), 481
rocalution::LocalVector::CopyFromPermuteBackward  (C++ function), 481
rocalution::LocalVector::ExtractCoarseBoundary  (C++ function), 481
rocalution::LocalVector::ExtractCoarseMapping  (C++ function), 481
rocalution::LocalVector::GetContinuousValues  (C++ function), 481
rocalution::LocalVector::GetIndexValues (C++ function), 498
rocalution::LocalVector::LeaveDataPtr (C++ function), 496
rocalution::LocalVector::Permute (C++ function), 497
rocalution::LocalVector::PermuteBackward (C++ function), 497
rocalution::LocalVector::Prolongation (C++ function), 498
rocalution::LocalVector::Restriction (C++ function), 497
rocalution::LocalVector::SetContinuousValues (C++ function), 498
rocalution::LocalVector::SetDataPtr (C++ function), 496
rocalution::LocalVector::SetIndexArray (C++ function), 498
rocalution::LocalVector::SetIndexValues (C++ function), 498
rocalution::LU (C++ class), 512
rocalution::MixedPrecisionDC (C++ class), 504
rocalution::MixedPrecisionDC::Set (C++ function), 505
rocalution::MultiColored (C++ class), 519
rocalution::MultiColored::SetDecomposition (C++ function), 519
rocalution::MultiColored::SetPrecondMatrixFormat (C++ function), 519
rocalution::MultiColoredGS (C++ class), 520
rocalution::MultiColoredILU (C++ class), 520
rocalution::MultiColoredSGS (C++ class), 519
rocalution::MultiColoredSGS::SetRelaxation (C++ function), 520
rocalution::MultiElimination (C++ class), 520
rocalution::MultiElimination::GetLevel (C++ function), 521
rocalution::MultiElimination::GetSizeDiagBlock (C++ function), 521
rocalution::MultiElimination::Set (C++ function), 521
rocalution::MultiElimination::SetPrecondMatrixFormat (C++ function), 521
rocalution::MultiGrid (C++ class), 509
rocalution::Operator (C++ class), 464
rocalution::Operator::GetGhostM (C++ function), 464
rocalution::Operator::GetGhostN (C++ function), 464
rocalution::Operator::GetGhostNnz (C++ function), 464
rocalution::Operator::GetLocalM (C++ function), 464
rocalution::Operator::GetLocalN (C++ function), 464
rocalution::Operator::GetLocalNnz (C++ function), 464
rocalution::Operator::GetM (C++ function), 464
rocalution::Operator::GetN (C++ function), 464
rocalution::ParallelManager (C++ class), 499
rocalution::ParallelManager::Clear (C++ function), 499
rocalution::ParallelManager::GetGlobalSize (C++ function), 499
rocalution::ParallelManager::GetLocalSize (C++ function), 499
rocalution::ParallelManager::GetNumProcs (C++ function), 500
rocalution::ParallelManager::GetNumReceivers (C++ function), 499
rocalution::ParallelManager::GetNumSenders (C++ function), 500
rocalution::ParallelManager::GlobalToLocal (C++ function), 500
rocalution::ParallelManager::LocalToGlobal (C++ function), 500
rocalution::ParallelManager::ReadFileASCII (C++ function), 500
rocalution::ParallelManager::SetBoundaryIndex (C++ function), 500
rocalution::ParallelManager::SetGlobalSize (C++ function), 500
rocalution::ParallelManager::SetLocalSize (C++ function), 500
rocalution::ParallelManager::SetMPICommunicator (C++ function), 499
rocalution::ParallelManager::SetReceivers (C++ function), 500
rocalution::ParallelManager::SetSenders (C++ function), 500
rocalution::ParallelManager::Status (C++ function), 500
(C++ function), 466
rocblas::Vector::WriteFileBinary
(C++ function), 467
rocblas::Vector::Zeros (C++ function), 466
rocblas (C++ type), 159
rocblas::reinit_logs (C++ function), 159
rocblas_add_stream (C++ function), 155, 160
rocblas_bfdot (C++ function), 89, 167
rocblas_bfdot_batched (C++ function), 89, 167
rocblas_bfdot_batched (C++ function), 90, 168
rocblas_bfdot_strided_batched (C++ function), 92, 170
rocblas_caxpy (C++ function), 95, 173
rocblas_caxpy_batched (C++ function), 173
rocblas_caxpy_strided_batched (C++ function), 174
rocblas_ccopy (C++ function), 87, 165
rocblas_ccopy_batched (C++ function), 87, 165
rocblas_ccopy_strided_batched (C++ function), 88, 166
rocblas_cdotc (C++ function), 89, 167
rocblas_cdotc_batched (C++ function), 90, 169
rocblas_cdotc_strided_batched (C++ function), 92, 170
rocblas_cdotu (C++ function), 89, 167
rocblas_cdotu_batched (C++ function), 90, 168
rocblas_cdotu_strided_batched (C++ function), 92, 170
rocblas_cgemm (C++ function), 140, 218
rocblas_cgemm_batched (C++ function), 141, 219
rocblas_cgemm_strided_batched (C++ function), 143, 221
rocblas_cgemv (C++ function), 113, 192
rocblas_cgemv_batched (C++ function), 113, 192
rocblas_cgemv_strided_batched (C++ function), 117, 196
rocblas_chemv (C++ function), 118, 197
rocblas_chemv_batched (C++ function), 113, 192
rocblas_chemv_strided_batched (C++ function), 114, 193
rocblas_create_handle (C++ function), 155, 160, 240
rocblas_crot (C++ function), 103, 182
rocblas_crot_batched (C++ function), 104, 183
rocblas_crot_strided_batched (C++ function), 105, 184
rocblas_crotg (C++ function), 106, 185
rocblas_crotg_batched (C++ function), 107, 186
rocblas_crotg_strided_batched (C++ function), 108, 187
rocblas_cscal (C++ function), 84, 163
rocblas_cscal_batched (C++ function), 85, 163
rocblas_cscal_strided_batched (C++ function), 86, 164
rocblas_csrot (C++ function), 103, 182
rocblas_csrot_batched (C++ function), 104, 183
rocblas_csrot_strided_batched (C++ function), 105, 184
rocblas_csscal (C++ function), 84, 163
rocblas_csscal_batched (C++ function), 85, 163
rocblas_csscal_strided_batched (C++ function), 86, 164
rocblas_cswap (C++ function), 93, 171
rocblas_cswap_batched (C++ function), 93, 171
rocblas_cswap_strided_batched (C++ function), 94, 172
rocblas_ctbmv (C++ function), 123, 201
rocblas_ctbmv_batched (C++ function), 124, 202
rocblas_ctbmv_strided_batched (C++ function), 125, 204
rocblas_ctrmv (C++ function), 119, 198
rocblas_ctrmv_batched (C++ function), 120, 199
rocblas_ctrmv_strided_batched (C++ function), 121, 200
rocblas_dasum (C++ function), 95, 175
rocblas_dasum_batched (C++ function), 96, 175
rocblas_dasum_strided_batched (C++ function), 97, 176
rocblas_datatype (C++ enum), 82, 237
rocblas_datatype::rocblas_datatype_bf16_c
(C++ enumerator), 83, 238
rocblas_datatype::rocblas_datatype_bf16_r
(C++ enumerator), 83, 238
rocblas_datatype::rocblas_datatype_f16_c
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_f16_r
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_f32_c
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_f32_r
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_f64_c
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_f64_r
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_i32_c
(C++ enumerator), 83, 238
rocblas_datatype::rocblas_datatype_i32_r
(C++ enumerator), 83, 238
rocblas_datatype::rocblas_datatype_i8_c
(C++ enumerator), 83, 238
rocblas_datatype::rocblas_datatype_i8_r
(C++ enumerator), 82, 238
rocblas_datatype::rocblas_datatype_u32_c
(C++ enumerator), 83, 238
rocblas_datatype::rocblas_datatype_u32_r
(C++ enumerator), 83, 238
rocblas_zscal (C++ function), 84, 163
rocblas_zscal_batched (C++ function), 85, 163
rocblas_zscal_strided_batched (C++ function), 86, 164
rocblas_zswap (C++ function), 93, 171
rocblas_zswap_batched (C++ function), 93, 171
rocblas_zswap_strided_batched (C++ function), 94, 172
rocblas_ztbmv (C++ function), 123, 201
rocblas_ztbmv_batched (C++ function), 124, 202
rocblas_ztbmv_strided_batched (C++ function), 125, 204
rocblas_ztromv (C++ function), 119, 198
rocblas_ztromv_batched (C++ function), 120, 199
rocblas_ztromv_strided_batched (C++ function), 121, 200
rocfft_array_type (C++ enum), 256
rocfft_array_type::rocfft_array_type_complex_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_real (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_hermitian_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_real (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_planar (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_real (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_interleaved (C++ enumerator), 256
rocfft_array_type::rocfft_array_type_complex_planar (C++ enumerator), 256
rocfft_status::rocfft_status_invalid_array_type (C++ enumerator), 255
rocfft_status::rocfft_status_invalid_dimensions (C++ enumerator), 255
rocfft_status::rocfft_status_invalid_distance (C++ enumerator), 255
rocfft_status::rocfft_status_invalid_offset (C++ enumerator), 255
rocfft_status::rocfft_status_invalid_strides (C++ enumerator), 255
rocfft_status::rocfft_status_success (C++ enumerator), 255
rocfft_plan (C++ function), 251
rocfft_plan_destroy (C++ function), 252
rocfft_plan_get_print (C++ function), 252
rocfft_plan_get_work_buffer_size (C++ function), 252
rocfft_precision (C++ enum), 256
rocfft_precision::rocfft_precision_double (C++ enumerator), 256
rocfft_precision::rocfft_precision_single (C++ enumerator), 256
rocfft_result_placement (C++ enum), 256
rocfft_result_placement::rocfft_placement_notinplace (C++ enumerator), 256
rocfft_result_placement::rocfft_placement_inplace (C++ enumerator), 256
rocfft_setup (C++ function), 251
rocfft_status (C++ enum), 255
rocfft_status::rocfft_status_failure (C++ enumerator), 255
rocfft_transform_type::rocfft_transform_type_real_forward (C++ function), 431
rocfft_transform_type::rocfft_transform_type_complex_forward (C++ function), 432
rocfft_transform_type::rocfft_transform_type_real_forward (C++ function), 433
rocfft_transform_type::rocfft_transform_type_complex_forward (C++ function), 434
rocfft_transform_type::rocfft_transform_type_complex_forward (C++ function), 435
rocfft_transform_type::rocfft_transform_type_real_forward (C++ function), 436
rocfft_transform_type::rocfft_transform_type_complex_forward (C++ function), 450

Index
rocsparse_dcsrilu0_buffer_size (C++ function), 358
rocsparse_dcsrmv (C++ function), 328
rocsparse_dcsrmv_analysis (C++ function), 297
rocsparse_dcsrsv_analysis (C++ function), 294
rocsparse_dcsrsv_buffer_size (C++ function), 312
rocsparse_dcsrsv_solve (C++ function), 319
rocsparse_ddoti (C++ function), 279
rocsparse_dell2csr (C++ function), 395
rocsparse_dellmv (C++ function), 304
rocsparse_destroy_handle (C++ function), 269
rocsparse_destroy_hyb_mat (C++ function), 276
rocsparse_destroy_mat_descr (C++ function), 272
rocsparse_destroy_mat_info (C++ function), 277
rocsparse_dgthr (C++ function), 283
rocsparse_dgthrz (C++ function), 285
rocsparse_dhybmv (C++ function), 309
rocsparse_diag_type (C++ enum), 266
rocsparse_diag_type (C++ enumerator), 266
rocsparse_droti (C++ function), 287
rocsparse_dsctr (C++ function), 288
rocsparse_ell2csr_nnz (C++ function), 394
rocsparse_fill_mode (C++ enum), 265
rocsparse_fill_mode (C++ enumerator), 265
rocsparse_fill_mode::rocsparse_fill_mode_lower (C++ function), 265
rocsparse_fill_mode::rocsparse_fill_mode_upper (C++ function), 275
rocsparse_fill_mode::rocsparse_fill_mode_triangular (C++ function), 275
rocsparse_get_git_rev (C++ function), 272
rocsparse_get_mat_diag_type (C++ function), 275
rocsparse_get_mat_fill_mode (C++ function), 275
rocsparse_get_mat_index_base (C++ function), 273
rocsparse_get_mat_type (C++ function), 274
rocsparse_get_pointer_mode (C++ function), 271
rocsparse_get_stream (C++ function), 270
rocsparse_get_version (C++ function), 271
rocsparse_handle (C++ type), 263
rocsparse_hyb_mat (C++ type), 264
rocsparse_hyb_partition (C++ type), 264
rocsparse_hyb_partition (C++ enum), 264
rocsparse_hyb_partition (C++ enumerator), 264
rocsparse_hyb_partition::rocsparse_hyb_partition_max (C++ function), 364
rocsparse_hyb_partition::rocsparse_hyb_partition_user (C++ function), 364
rocsparse_index_base (C++ enum), 265
rocsparse_index_base::rocsparse_index_base_one (C++ function), 265
rocsparse_index_base::rocsparse_index_base_zero (C++ function), 265
rocsparse_layer_mode (C++ enum), 267
rocsparse_layer_mode::rocsparse_layer_mode_log_trace (C++ function), 267
rocsparse_layer_mode::rocsparse_layer_mode_log_bench (C++ function), 267
rocsparse_layer_mode::rocsparse_layer_mode_none (C++ function), 267
rocsparse_mat_descr (C++ type), 263
rocsparse_mat_info (C++ type), 264
rocsparse_matrix_type (C++ enum), 265
rocsparse_matrix_type::rocsparse_matrix_type_general (C++ function), 265
rocsparse_matrix_type::rocsparse_matrix_type_infinite (C++ function), 265
rocsparse_matrix_type::rocsparse_matrix_type_upper (C++ function), 265
rocsparse_matrix_type::rocsparse_matrix_type_triangular (C++ function), 265
rocsparse_matrix_type::rocsparse_matrix_type_lower (C++ function), 265
rocsparse_operation (C++ enum), 266
rocsparse_operation::rocsparse_operation_conjugate_transpose (C++ function), 266
rocsparse_operation::rocsparse_operation_none (C++ function), 266
rocsparse_operation::rocsparse_operation_transpose (C++ function), 266
rocsparse_pointer_mode (C++ type), 266
rocsparse_pointer_mode (C++ function), 266
rocsparse_pointer_mode (C++ enumerator), 266
rocsparse_pointer_mode::rocsparse_pointer_mode_device (C++ function), 266
rocsparse_pointer_mode::rocsparse_pointer_mode_host (C++ function), 266
rocsparse_saxpyi (C++ function), 277
rocsparse_scoommv (C++ function), 290
rocsparse_scscrmv (C++ function), 328
rocsparse_scsrmv (C++ function), 329
rocsparse_scsrilu0 (C++ function), 365
rocsparse_scsrilu0_analysis (C++ function), 361
rocsparse_scsrilu0_buffer_size (C++ function), 358
rocsparse_scsrmv (C++ function), 297

638

Index
rocsparse_scsrmv_analysis (C++ function), 294
rocsparse_scsrsrv_analysis (C++ function), 316
rocsparse_scsrsrv_buffer_size (C++ function), 312
rocsparse_scsrsrv_solve (C++ function), 319
rocsparse_sell2csr (C++ function), 395
rocsparse_sellmv (C++ function), 304
rocsparse_set_mat_diag_type (C++ function), 275
rocsparse_set_mat_fill_mode (C++ function), 274
rocsparse_set_mat_index_base (C++ function), 271
rocsparse_set_mat_type (C++ function), 274
rocsparse_set_pointer_mode (C++ function), 273
rocsparse_set_mat_type (C++ function), 274
rocsparse_set_stream (C++ function), 271
rocsparse_sghtr (C++ function), 283
rocsparse_sghthr (C++ function), 285
rocsparse_shybmv (C++ function), 309
rocsparse_solve_policy (C++ enum), 267
rocsparse_solve_policy::rocsparse_solve_policy_auto (C++ enumerator), 267
rocsparse_status (C++ enum), 267
rocsparse_status::rocsparse_status_arch_mismatch (C++ enumerator), 268
rocsparse_status::rocsparse_status_internal_error (C++ enumerator), 268
rocsparse_status::rocsparse_status_invalid_handle (C++ enumerator), 267
rocsparse_status::rocsparse_status_invalid_pointer (C++ enumerator), 268
rocsparse_status::rocsparse_status_invalid_size (C++ enumerator), 268
rocsparse_status::rocsparse_status_invalid_value (C++ enumerator), 268
rocsparse_status::rocsparse_status_memory_error (C++ enumerator), 268
rocsparse_status::rocsparse_status_not_implemented (C++ enumerator), 268
rocsparse_status::rocsparse_status_success (C++ enumerator), 267
rocsparse_status::rocsparse_status_zero_pivot (C++ enumerator), 268
rocsparse_zaxpyi (C++ function), 279
rocsparse_zcoomv (C++ function), 293
rocsparse_zcsr2csc (C++ function), 386
rocsparse_zcsr2ell (C++ function), 392
rocsparse_zcsr2hyb (C++ function), 403
rocsparse_zcsrgemm (C++ function), 353
rocsparse_zcsrgemm_buffer_size (C++ function), 340
rocsparse_zcsrilu0 (C++ function), 373
rocsparse_zcsrilu0_analysis (C++ function), 364
rocsparse_zcsrilu0_buffer_size (C++ function), 360
rocsparse_zcsrmv (C++ function), 302
rocsparse_zcsrmv_analysis (C++ function), 296
rocsparse_zcsrsrv_analysis (C++ function), 318
rocsparse_zcsrsrv_buffer_size (C++ function), 314
rocsparse_zcsrsrv_solve (C++ function), 324
rocsparse_zdotci (C++ function), 282
rocsparse_zdoti (C++ function), 281
rocsparse_zell2csr (C++ function), 398
rocsparse_zellmv (C++ function), 307
rocsparse_zgthr (C++ function), 284
rocsparse_zgthrz (C++ function), 286
rocsparse_zhybmv (C++ function), 311
rocsparse_zsctr (C++ function), 289