ReadTheDocs-Breathe Documentation

Release 1.0.0

Thomas Edvalson

Sep 06, 2019

CONTENTS

1 Going to 11: Amping Up the Programming-Language Run-Time Foundation 3

2 Solid Compilation Foundation and Language Support 5

2.1 Quick Start Guide . . . 5
2.1.1 Current Release Notes . . . 5
2.1.2 Installation Guide . . . 5
2.1.3 Programming Guide . . . 6
2.1.4 ROCm GPU Tunning Guides . . . 6
2.1.5 GCN ISA Manuals . . . 7
2.1.6 ROCm API References . . . 7
2.1.7 ROCm Tools . . . 8
2.1.8 ROCm Libraries . . . 9
2.1.9 ROCm Compiler SDK . . . 10
2.1.10 ROCm System Management . . . 11
2.1.11 ROCm Virtualization & Containers . . . 11
2.1.12 Remote Device Programming . . . 11
2.1.13 Deep Learning on ROCm . . . 12
2.1.14 System Level Debug . . . 12
2.1.15 Tutorial . . . 12
2.1.16 ROCm Glossary . . . 12
2.2 Current Release Notes . . . 12
2.2.1 Hotfix release ROCm 2.7.1 . . . 12
2.2.1.1 Defect fixed in ROCm 2.7.1 . . . 12
2.2.1.2 Upgrading from ROCm 2.7 to 2.7.1 . . . 13
2.2.2 New features and enhancements in ROCm 2.7 . . . 13
2.2.2.1 [rocFFT] Real FFT Functional . . . 13
2.2.2.2 rocRand Enhancements and Optimizations . . . 13
2.2.2.3 RAS . . . 13
2.2.2.4 ROCm-SMI enhancements . . . 13
2.2.3 New features and enhancements in ROCm 2.6 . . . 13
2.2.3.1 ROCmInfo enhancements . . . 13
2.2.3.2 [Thrust] Functional Support on Vega20 . . . 14
2.2.3.3 MIGraphX v0.3 . . . 14
2.2.3.4 MIOpen 2.0 . . . 14
2.2.3.5 Bloat16 software support in rocBLAS/Tensile . . . 14
2.2.3.6 AMD Infinity Fabric™ Link enablement . . . 15
2.2.3.7 ROCm-smi features and bug fixes . . . 15
2.2.3.8 ROCm-smi-lib enhancements . . . 15
2.2.3.9 RCCL2 Enablement . . . 15
2.2.3.10 rocFFT enhancements . . . 15

2.2.4 New features and enhancements in ROCm 2.5 . . . 15
2.2.4.1 UCX 1.6 support . . . 15
2.2.4.2 BFloat16 GEMM in rocBLAS/Tensile . . . 15
2.2.4.3 ROCm-SMI enhancements . . . 16
2.2.4.4 [PyTorch] multi-GPU functional support (CPU aggregation/Data Parallel) . . . 16
2.2.4.5 rocSparse optimization on Radeon Instinct MI50 and MI60 . . . 16
2.2.4.6 [Thrust] Preview . . . 16
2.2.4.7 Support overlapping kernel execution in same HIP stream . . . 16
2.2.4.8 AMD Infinity Fabric™ Link enablement . . . 16
2.2.5 New features and enhancements in ROCm 2.4 . . . 16
2.2.5.1 TensorFlow 2.0 support . . . 16
2.2.5.2 AMD Infinity Fabric™ Link enablement . . . 16
2.2.6 New features and enhancements in ROCm 2.3 . . . 17
2.2.6.1 Mem usage per GPU . . . 17
2.2.6.2 MIVisionX, v1.1 - ONNX . . . 17
2.2.6.3 MIGraphX, v0.2 . . . 17
2.2.6.4 MIOpen, v1.8 - 3d convolutions and int8 . . . 17
2.2.6.5 Caffe2 - mGPU support . . . 17
2.2.6.6 rocTracer library, ROCm tracing API for collecting runtimes API and asynchronous GPU activity traces . . . 17
2.2.6.7 BLAS - Int8 GEMM performance, Int8 functional and performance . . . 17
2.2.6.8 Prioritized L1/L2/L3 BLAS (functional) . . . 17
2.2.6.9 BLAS - tensile optimization . . . 18
2.2.6.10 MIOpen Int8 support . . . 18
2.2.7 New features and enhancements in ROCm 2.2 . . . 18
2.2.7.1 rocSparse Optimization on Vega20 . . . 18
2.2.7.2 DGEMM and DTRSM Optimization . . . 18
2.2.7.3 Caffe2 . . . 18
2.2.8 New features and enhancements in ROCm 2.1 . . . 18
2.2.8.1 RocTracer v1.0 preview release – ‘rocprof’ HSA runtime tracing and statistics support - . . . 18
2.2.8.2 Improvements to ROCM-SMI tool - . . . 18
2.2.8.3 DGEMM Optimizations - . . . 18
2.2.9 New features and enhancements in ROCm 2.0 . . . 18
2.2.9.1 Adds support for RHEL 7.6 / CentOS 7.6 and Ubuntu 18.04.1 . . . 19
2.2.9.2 Adds support for Vega 7nm, Polaris 12 GPUs . . . 19
2.2.9.3 Introduces MIVisionX . . . 19
2.2.9.4 Improvements to ROCm Libraries . . . 19
2.2.9.5 MIOpen . . . 19
2.2.9.6 Tensorflow multi-gpu and Tensorflow FP16 support for Vega 7nm . . . 19
2.2.9.7 PyTorch/Caffe2 with Vega 7nm Support . . . 19
2.2.9.8 Improvements to ROCProfiler tool . . . 19
2.2.9.9 Support for hipStreamCreateWithPriority . . . 19
2.2.9.10 OpenCL 2.0 support . . . 20
2.2.9.11 Improved Virtual Addressing (48 bit VA) management for Vega 10 and later GPUs . 20
2.2.9.12 Kubernetes support . . . 20
2.2.9.13 Removed features . . . 20
2.2.10 New features and enhancements in ROCm 1.9.2 . . . 20
2.2.10.1 RDMA(MPI) support on Vega 7nm . . . 20
2.2.10.2 Improvements to HCC . . . 20
2.2.10.3 Improvements to ROCProfiler tool . . . 20
2.2.10.4 Critical bug fixes . . . 20
2.2.11 New features and enhancements in ROCm 1.9.1 . . . 20
2.2.11.1 Added DPM support to Vega 7nm . . . 20

2.2.11.2 Fix for ‘ROCm profiling’ “Version mismatch between HSA runtime and libhsa-runtime-tools64.so.1” error 21
2.2.12 New features and enhancements in ROCm 1.9.0 21
2.2.12.1 Preview for Vega 7nm 21
2.2.12.2 System Management Interface 21
2.2.12.3 Improvements to HIP/HCC 21
2.2.12.4 Preview for rocprof Profiling Tool 21
2.2.12.5 Preview for rocr Debug Agent rocr_debug_agent 21
2.2.12.6 New distribution support 21
2.2.12.7 ROCm 1.9 is ABI compatible with KFD in upstream Linux kernels. 22
2.3 ROCm Installation Guide 22
2.3.1 Are You Ready to ROCK? 22
2.3.2 Current ROCm Version: 2.7.1 22
2.3.3 The latest ROCm platform - ROCm 2.7 22
2.3.4 Hardware Support 23
2.3.4.1 Supported GPUs 24
2.3.4.2 Supported CPUs 24
2.3.4.3 Not supported or limited support under ROCm 25
2.3.4.3.1 Limited support 25
2.3.4.3.2 Not supported 26
2.3.5 Supported Operating Systems - New operating systems available 26
2.3.5.1 ROCm support in upstream Linux kernels 26
2.3.6 Installing from AMD ROCm repositories 27
2.3.6.1 ROCm Binary Package Structure 27
2.3.6.2 Ubuntu Support - installing from a Debian repository 29
2.3.6.2.1 First make sure your system is up to date 29
2.3.6.2.2 Add the ROCm apt repository 29
2.3.6.2.3 Install 30
2.3.6.2.4 Next set your permissions 30
2.3.6.2.5 Test basic ROCm installation 30
2.3.6.2.6 Performing an OpenCL-only Installation of ROCm 30
2.3.6.2.7 How to uninstall from Ubuntu 16.04 or Ubuntu 18.04 31
2.3.6.2.8 Installing development packages for cross compilation 31
2.3.6.2.9 Using Debian-based ROCm with upstream kernel drivers 31
2.3.6.3 CentOS/RHEL 7 (7.6) Support 31
2.3.6.3.1 Preparing RHEL 7 (7.6) for installation 31
2.3.6.3.2 Install and setup Devtoolset-7 32
2.3.6.3.3 Prepare CentOS/RHEL (7.6) for DKMS Install 32
2.3.6.3.4 Installing ROCm on the system 32
2.3.6.3.5 Set up permissions 32
2.3.6.3.6 Test basic ROCm installation 33
2.3.6.3.7 Performing an OpenCL-only Installation of ROCm 33
2.3.6.3.8 Compiling applications using HCC, HIP, and other ROCm software 33
2.3.6.3.9 How to uninstall ROCm from CentOS/RHEL 7.6 33
2.3.6.3.10 Installing development packages for cross compilation 34
2.3.6.3.11 Using ROCm with upstream kernel drivers 34
2.3.7 Known issues / workarounds 34
2.3.7.1 rocprofiler –hiptrace and –hsatrace fails to load roctracer library 34
2.3.7.2 rocFFT unit tests - memory access fault 34
2.3.8 Closed source components 34
2.3.9 Getting ROCm source code 35
2.3.9.1 Installing repo 35
2.3.9.2 Downloading the ROCm source code 35
2.3.9.3 Building the ROCm source code 35

2.3.10 Deprecation Notice . . . 35
2.3.10.1 HCC . . . 35
2.3.10.2 hipThrust . . . 36
2.3.11 Final notes . . . 36
2.4 Programming Guide . . . 36
2.4.1 ROCm Languages . . . 36
2.4.1.1 ROCm, Lingua Franca, C++, OpenCL and Python . . . 36
2.4.1.2 What is the Heterogeneous Compute (HC) API? . . . 36
2.4.1.3 When to Use HC . . . 37
2.4.1.4 HIP: Heterogeneous-Computing Interface for Portability . . . 37
2.4.1.5 When to Use HIP . . . 37
2.4.1.6 OpenCL™: Open Compute Language . . . 37
2.4.1.7 When to Use OpenCL . . . 37
2.4.1.8 Anaconda Python With Numba . . . 37
2.4.1.9 Numba . . . 37
2.4.1.10 When to Use Anaconda . . . 38
2.4.1.11 Wrap-Up . . . 38
2.4.1.12 Table Comparing Syntax for Different Compute APIs . . . 40
2.4.1.13 Notes . . . 41
2.4.2 HCC Programming Guide . . . 41
2.4.2.1 HCC: Heterogeneous Compute Compiler . . . 41
2.4.2.2 Accelerator Modes Supported . . . 41
2.4.2.3 Platform Requirements . . . 42
2.4.2.4 Compiler Backends . . . 42
2.4.2.5 Installation . . . 42
2.4.2.6 Ubuntu . . . 42
2.4.2.7 Download HCC . . . 42
2.4.2.8 Build HCC from source . . . 43
2.4.2.9 Use HCC . . . 44
2.4.2.10 Multiple ISA . . . 44
2.4.2.11 CodeXL Activity Logger . . . 45
2.4.3 HC Best Practices . . . 45
2.4.3.1 HCC built-in macros . . . 46
2.4.3.2 HC-specific features . . . 46
2.4.3.3 Differences between HC API and C++ AMP . . . 47
2.4.3.4 HCC Profile Mode . . . 48
2.4.3.4.1 Kernel Commands . . . 49
2.4.3.4.2 Memory Copy Commands . . . 49
2.4.3.4.3 Barrier Commands . . . 50
2.4.3.4.4 Overhead . . . 51
2.4.3.4.5 Additional Details and tips . . . 51
2.4.3.5 API documentation . . . 51
2.4.4 HIP Programing Guide . . . 51
2.4.5 HIP Best Practices . . . 52
2.4.6 OpenCL Programing Guide . . . 52
2.4.7 OpenCL Best Practices . . . 52
2.5 ROCm GPU Tuning Guides . . . 52
2.5.1 GFX7 Tuning Guide . . . 52
2.5.2 GFX8 Tuning Guide . . . 52
2.5.3 Vega Tuning Guide . . . 52
2.6 GCN ISA Manuals . . . 52
2.6.1 GCN 1.1 . . . 52
2.6.2 GCN 2.0 . . . 53
2.6.3 Vega . . . 53

2.6.4 Inline GCN ISA Assembly Guide . . . 53
2.6.4.1 The Art of AMDGCN Assembly: How to Bend the Machine to Your Will . . . 53
2.6.4.2 DS Permute Instructions . . . 53
2.6.4.3 Passing Parameters to a Kernel . . . 53
2.6.4.4 The GPR Counting . . . 56
2.6.4.5 Compiling GCN ASM Kernel Into Hsaco . . . 57
2.7 ROCm API References . . . 57
2.7.1 ROCr System Runtime API . . . 57
2.7.2 HCC Language Runtime API . . . 57
2.7.3 HIP Language Runtime API . . . 57
2.7.4 HIP Math API . . . 57
2.7.5 Math Library API's . . . 58
2.7.6 Deep Learning API's . . . 58
2.8 ROCm Tools . . . 58
2.8.1 HCC . . . 58
2.8.1.1 Download HCC . . . 58
2.8.1.2 Build HCC from source . . . 58
2.8.1.3 Use HCC . . . 59
2.8.1.4 Multiple ISA . . . 59
2.8.1.5 CodeXL Activity Logger . . . 60
2.8.1.6 HCC with ThinLTO Linking . . . 60
2.8.2 GCN Assembler and Disassembler . . . 60
2.8.3 GCN Assembler Tools . . . 64
2.8.4 ROC Profiler . . . 66
2.8.4.1 Download . . . 67
2.8.4.2 Build . . . 67
2.8.4.3 Profiling Tool 'rocprof' Usage . . . 68
2.8.5 ROCr Debug Agent . . . 69
2.8.6 ROCm-GDB . . . 72
2.8.7 Radeon Compute Profiler . . . 73
2.8.8 ROC Tracer . . . 73
2.8.8.1 Documentation . . . 73
2.8.8.2 To build and run test . . . 73
2.8.9 CodeXL . . . 74
2.8.10 GPUPerfAPI . . . 74
2.8.11 ROCm Binary Utilities . . . 74
2.8.12 MIVisionX . . . 74
2.8.12.1 AMD OpenVX (amd_openvx) . . . 75
2.8.12.2 AMD OpenVX Extensions . . . 75
2.8.12.2.1 Features . . . 75
2.8.12.2.2 Pre-requisites: . . . 75
2.8.12.2.3 Build Instructions . . . 76
2.8.12.2.4 Build using Visual Studio Professional 2013 on 64-bit Windows 10/8.1/7 . 76
2.8.12.2.5 Build using CMake . . . 76
2.8.12.2.6 Radeon Loom Stitching Library (vx_loomsl) . . . 76
2.8.12.2.7 OpenVX Neural Network Extension Library (vx_nn) . . . 78
2.8.12.2.8 AMD Module for OpenCV-interop from OpenVX (vx_opencv) . . . 79
2.8.12.2.8.1 Build Instructions . . . 80
2.8.12.3 Applications . . . 81
2.8.12.3.1 Cloud Inference Application (cloud_inference) . . . 81
2.8.12.3.2 Convert Neural Net models into AMD NNIR and OpenVX Code . . . 81
2.8.12.3.3 Currently supported . . . 83
2.8.12.4 Samples . . . 84
2.8.12.4.1 GDF - Graph Description Format . . . 84

2.8.12.5 MIVisionX Toolkit 85
2.8.12.6 Utilities 85
2.8.12.7 Prerequisites 85
2.8.12.7.1 Windows 86
2.8.12.7.2 Linux 86
2.8.12.7.3 Pre-requisites setup script - MIVisionX-setup.py 86
2.8.12.7.4 Prerequisites for running the scripts 86
2.8.12.8 Build MIVisionX 87
2.8.12.9 Build & Install MIVisionX 87
2.8.12.10 Prerequisites 87
2.8.12.10.1 Using MIVisionX-setup.py and CMake on Linux (Ubuntu 16.04/18.04 or CentOS 7.5/7.6) with ROCm 88
2.8.12.10.2 Build using CMake on Linux (Ubuntu 16.04 64-bit or CentOS 7.5 / 7.6) with ROCm 88
2.8.12.10.3 Build annInferenceApp using Qt Creator 89
2.8.12.10.4 Build Radeon LOOM using Visual Studio Professional 2013 on 64-bit Windows 10/8.1/7 89
2.8.12.10.5 Verify the Installation 89
2.8.12.10.6 Linux 89
2.8.12.11 Docker 90
2.8.12.11.1 MIVisionX Docker 90
2.8.12.11.2 Docker Workflow Sample on Ubuntu 16.04 90
2.8.12.11.3 Workflow 90
2.8.12.11.4 Supported Neural Net Layers 91
2.8.12.12 Release Notes 92
2.8.12.12.1 Known issues 92
2.8.12.12.2 Tested configurations 92
2.9 ROCm Libraries 92
2.9.1 rocFFT 92
2.9.1.1 API design 93
2.9.1.2 Installing pre-built packages 93
2.9.1.3 Quickstart rocFFT build 93
2.9.1.4 Manual build (all supported platforms) 93
2.9.1.5 Example 93
2.9.1.6 API 94
2.9.1.6.1 Types 94
2.9.1.6.2 Library Setup and Cleanup 95
2.9.1.6.3 Plan 95
2.9.1.6.4 Plan description 96
2.9.1.6.5 Execution 97
2.9.1.6.6 Execution info 98
2.9.1.6.7 Enumerations 99
2.9.2 rocBLAS 100
2.9.2.1 Prerequisites 100
2.9.2.2 Installing pre-built packages 100
2.9.2.3 Quickstart rocBLAS build 100
2.9.2.4 Manual build (all supported platforms) 101
2.9.2.5 rocBLAS interface examples 101
2.9.2.6 GEMV API 101
2.9.2.7 Batched and strided GEMM API 101
2.9.2.8 Asynchronous API 102
2.9.2.9 API 102
2.9.2.9.1 Types 102
2.9.2.9.1.1 Definitions 102

2.9.2.9.1.2 rocblas_int 102
2.9.2.9.1.3 rocblas_long 102
2.9.2.9.1.4 rocblas_float_complex 102
2.9.2.9.1.5 rocblas_double_complex 102
2.9.2.9.1.6 rocblas_half 102
2.9.2.9.1.7 rocblas_half_complex 102
2.9.2.9.1.8 rocblas_handle 102
2.9.2.9.1.9 Enums 103
2.9.2.9.1.10 rocblas_operation 103
2.9.2.9.1.11 rocblas_fill 103
2.9.2.9.1.12 rocblas_diagonal 103
2.9.2.9.1.13 rocblas_side 103
2.9.2.9.1.14 rocblas_status 104
2.9.2.9.1.15 rocblas_datatype 104
2.9.2.9.1.16 rocblas_pointer_mode 105
2.9.2.9.1.17 rocblas_layer_mode 105
2.9.2.9.1.18 rocblas_gemm_algo 105
2.9.2.9.2 Functions 105
2.9.2.9.2.1 Level 1 BLAS 105
2.9.2.9.2.2 rocblas_<type>scal() 105
2.9.2.9.2.3 rocblas_<type>copy() 106
2.9.2.9.2.4 rocblas_<type>dot() 106
2.9.2.9.2.5 rocblas_<type>swap() 107
2.9.2.9.2.6 rocblas_<type>axpy() 107
2.9.2.9.2.7 rocblas_<type>asum() 108
2.9.2.9.2.8 rocblas_<type>nrm2() 108
2.9.2.9.2.9 rocblas_i<type>amax() 109
2.9.2.9.2.10 rocblas_i<type>amin() 109
2.9.2.9.2.11 Level 2 BLAS 109
2.9.2.9.2.12 rocblas_<type>gemv() 109
2.9.2.9.2.13 rocblas_<type>trsv() 110
2.9.2.9.2.14 rocblas_<type>ger() 111
2.9.2.9.2.15 rocblas_<type>syr() 112
2.9.2.9.2.16 Level 3 BLAS 113
2.9.2.9.2.17 rocblas_<type>trtri_batched() 113
2.9.2.9.2.18 rocblas_<type>trsm() 113
2.9.2.9.2.19 rocblas_<type>gemm() 114
2.9.2.9.2.20 rocblas_<type>gemm_strided_batched() 115
2.9.2.9.2.21 rocblas_<type>gemm_kernel_name() 117
2.9.2.9.2.22 rocblas_<type>geam() 117
2.9.2.9.2.23 BLAS Extensions 118
2.9.2.9.2.24 rocblas_gemm_ex() 118
2.9.2.9.2.25 rocblas_gemm_strided_batched_ex() 118
2.9.2.9.2.26 Build Information 120
2.9.2.9.2.27 rocblas_get_version_string() 120
2.9.2.9.2.28 Auxiliary 121
2.9.2.9.2.29 rocblas_pointer_to_mode() 121
2.9.2.9.2.30 rocblas_create_handle() 121
2.9.2.9.2.31 rocblas_destroy_handle() 121
2.9.2.9.2.32 rocblas_add_stream() 121
2.9.2.9.2.33 rocblas_set_stream() 121
2.9.2.9.2.34 rocblas_get_stream() 121
2.9.2.9.2.35 rocblas_set_pointer_mode() 122
2.9.2.9.2.36 rocblas_get_pointer_mode() 122

2.9.2.9.2.37 rocblas_set_vector() 122
2.9.2.9.2.38 rocblas_get_vector() 122
2.9.2.9.2.39 rocblas_set_matrix() 122
2.9.2.9.2.40 rocblas_get_matrix() 122
2.9.2.10 All API 122
2.9.3 hipBLAS 144
2.9.3.1 Introduction 144
2.9.3.1.1 Installing pre-built packages 144
2.9.3.1.2 Quickstart hipBLAS build 144
2.9.3.2 Build 145
2.9.3.2.1 Dependencies For Building Library 145
2.9.3.2.2 Build Library Using Script (Ubuntu only) 145
2.9.3.2.3 Build Library Using Individual Commands 145
2.9.3.2.4 Build Library + Tests + Benchmarks + Samples Using Individual Commands 145
2.9.3.2.5 Common build problems 146
2.9.3.3 Running 147
2.9.3.3.1 Notice 147
2.9.3.3.2 hipBLAS interface examples 147
2.9.3.3.3 GEMV API 147
2.9.3.3.4 Batched and strided GEMM API 148
2.9.4 Tensile 148
2.9.4.1 Introduction 148
2.9.4.1.1 Quick Example (Ubuntu): 148
2.9.4.2 Benchmark Config example 149
2.9.4.2.1 Example Benchmark config.yaml as input file to Tensile 149
2.9.4.2.2 Structure of config.yaml 150
2.9.4.2.3 Global Parameters 150
2.9.4.2.4 Problem Type Parameters 151
2.9.4.2.5 Solution / Kernel Parameters 152
2.9.4.2.6 Defaults 152
2.9.4.3 Benchmark Protocol 152
2.9.4.3.1 Old Benchmark Architecture was Intractable 152
2.9.4.3.2 Incremental Benchmark is Faster 152
2.9.4.3.3 Phases of Benchmark 152
2.9.4.3.4 Initial Solution Parameters 153
2.9.4.3.5 Problem Sizes 153
2.9.4.3.6 Benchmark Common Parameters 154
2.9.4.3.7 Fork Parameters 154
2.9.4.3.8 Benchmark Fork Parameters 154
2.9.4.3.9 Join Parameters 155
2.9.4.3.10 Benchmark Join Parameters 155
2.9.4.3.11 Benchmark Final Parameters 155
2.9.4.4 Contributing 155
2.9.4.5 Dependencies 155
2.9.4.5.1 CMake 155
2.9.4.5.2 Python 155
2.9.4.5.3 Compilers 156
2.9.4.6 Installation 156
2.9.4.7 Kernel Parameters 156
2.9.4.7.1 Solution / Kernel Parameters 156
2.9.4.7.2 Kernel Parameters Affect Performance 157
2.9.4.7.3 How N-Dimensional Tensor Contractions Are Mapped to Finite-Dimensional GPU Kernels 158
2.9.4.7.4 Special Dimensions: D0, D1 and DU 158

2.9.4.7.5 GPU Kernel Dimension 158
2.9.4.8 Languages 158
2.9.4.8.1 Tensile Benchmarking is Python 158
2.9.4.8.2 Tensile Library 158
2.9.4.8.3 Device Languages 158
2.9.4.9 Library Logic 159
2.9.4.10 Problem Nomenclature 159
2.9.4.10.1 Example Problems 159
2.9.4.10.2 Nomenclature 159
2.9.4.10.3 Limitations 160
2.9.4.11 Tensile.lib 160
2.9.4.12 Versioning 160
2.9.5 rocThrust 161
2.9.5.1 Introduction 161
2.9.5.2 Requirements 161
2.9.5.3 Hardware 161
2.9.5.4 Build And Install 161
2.9.5.5 Using rocThrust In A Project 162
2.9.5.6 Running Unit Tests 162
2.9.5.7 Documentation 162
2.9.5.8 Support 162
2.9.6 hipCUB 162
2.9.6.1 Requirements 163
2.9.6.2 Build And Install 163
2.9.6.3 Using hipCUB In A Project 164
2.9.6.4 Running Unit Tests 164
2.9.6.5 Documentation 164
2.9.6.6 Support 164
2.9.6.7 Contributions and License 165
2.9.7 ROCm SMI library 165
2.9.7.1 Important note about Versioning and Backward Compatibility 165
2.9.7.2 Building ROCm SMI 165
2.9.7.2.1 Additional Required software for building 165
2.9.7.3 Building the Documentation 165
2.9.7.4 Building the Tests 166
2.9.7.5 Usage Basics 166
2.9.7.5.1 Device Indices 166
2.9.7.5.2 Hello ROCm SMI 166
2.9.8 RCCL 167
2.9.8.1 Introduction 167
2.9.8.2 Requirements 167
2.9.8.3 Quickstart RCCL Build 167
2.9.8.4 Manual build 168
2.9.8.4.1 To build the library : 168
2.9.8.5 To build the RCCL package and install package : 168
2.9.8.6 Tests 168
2.9.8.7 Library and API Documentation 169
2.9.8.8 Copyright 169
2.9.9 rocALUTION 169
2.9.9.1 Introduction 169
2.9.9.2 Overview 169
2.9.9.3 Building and Installing 170
2.9.9.4 Installing from AMD ROCm repositories 170
2.9.9.5 Building rocALUTION from Open-Source repository 170

2.9.9.6 Download rocALUTION . . . 170
2.9.9.7 Using *install.sh* to build dependencies + library . . . 170
2.9.9.8 Using *install.sh* to build dependencies + library + client . . . 171
2.9.9.9 Using individual commands to build rocALUTION . . . 171
2.9.9.10 Common build problems . . . 172
2.9.9.11 Simple Test . . . 173
2.9.9.12 API . . . 173
2.9.9.12.1 Host Utility Functions . . . 173
2.9.9.12.2 Backend Manager . . . 174
2.9.9.12.3 Base Rocalution . . . 176
2.9.9.12.4 Operator . . . 178
2.9.9.12.5 Vector . . . 179
2.9.9.12.6 Local Matrix . . . 185
2.9.9.12.7 Local Stencil . . . 203
2.9.9.12.8 Global Matrix . . . 204
2.9.9.12.9 Local Vector . . . 206
2.9.9.12.10 Global Vector . . . 209
2.9.9.12.11 Parallel Manager . . . 210
2.9.9.12.12 Solvers . . . 211
2.9.9.12.13 Preconditioners . . . 224
2.9.10 rocSPARSE . . . 232
2.9.10.1 Introduction . . . 232
2.9.10.2 Device and Stream Management . . . 232
2.9.10.2.1 Asynchronous Execution . . . 232
2.9.10.2.2 HIP Device Management . . . 232
2.9.10.2.3 HIP Stream Management . . . 233
2.9.10.2.4 Multiple Streams and Multiple Devices . . . 233
2.9.10.3 Building and Installing . . . 233
2.9.10.3.1 Installing from AMD ROCm repositories . . . 233
2.9.10.3.2 Building rocSPARSE from Open-Source repository . . . 233
2.9.10.3.3 Using *install.sh* to build dependencies + library . . . 234
2.9.10.3.4 Using *install.sh* to build dependencies + library + client . . . 234
2.9.10.3.5 Using individual commands to build rocSPARSE . . . 234
2.9.10.3.6 Common build problems . . . 235
2.9.10.4 Unit tests . . . 235
2.9.10.5 Benchmarks . . . 236
2.9.10.6 Storage Formats . . . 236
2.9.10.6.1 COO storage format . . . 236
2.9.10.6.2 CSR storage format . . . 236
2.9.10.6.3 ELL storage format . . . 237
2.9.10.7 Types . . . 238
2.9.10.7.1 rocsparse_handle . . . 238
2.9.10.7.2 rocsparse_mat_descr . . . 238
2.9.10.7.3 rocsparse_mat_info . . . 238
2.9.10.7.4 rocsparse_hyb_mat . . . 238
2.9.10.7.5 rocsparse_action . . . 238
2.9.10.7.6 rocsparse_hyb_partition . . . 239
2.9.10.7.7 rocsparse_index_base . . . 239
2.9.10.7.8 rocsparse_matrix_type . . . 239
2.9.10.7.9 rocsparse_fill_mode . . . 240
2.9.10.7.10 rocsparse_diag_type . . . 240
2.9.10.7.11 rocsparse_operation . . . 240
2.9.10.7.12 rocsparse_pointer_mode . . . 241
2.9.10.7.13 rocsparse_analysis_policy . . . 241

2.9.10.7.14 rocsparse_solve_policy . . . 241
2.9.10.7.15 rocsparse_layer_mode . . . 241
2.9.10.7.16 rocsparse_status . . . 242
2.9.10.8 Logging . . . 242
2.9.10.9 Sparse Auxiliary Functions . . . 243
2.9.10.9.1 rocsparse_create_handle() . . . 243
2.9.10.9.2 rocsparse_destroy_handle() . . . 243
2.9.10.9.3 rocsparse_set_stream() . . . 244
2.9.10.9.4 rocsparse_get_stream() . . . 244
2.9.10.9.5 rocsparse_set_pointer_mode() . . . 245
2.9.10.9.6 rocsparse_get_pointer_mode() . . . 245
2.9.10.9.7 rocsparse_get_version() . . . 245
2.9.10.9.8 rocsparse_get_git_rev() . . . 246
2.9.10.9.9 rocsparse_create_mat_descr() . . . 246
2.9.10.9.10 rocsparse_destroy_mat_descr() . . . 246
2.9.10.9.11 rocsparse_copy_mat_descr() . . . 247
2.9.10.9.12 rocsparse_set_mat_index_base() . . . 247
2.9.10.9.13 rocsparse_get_mat_index_base() . . . 247
2.9.10.9.14 rocsparse_set_mat_type() . . . 248
2.9.10.9.15 rocsparse_get_mat_type() . . . 248
2.9.10.9.16 rocsparse_set_mat_fill_mode() . . . 248
2.9.10.9.17 rocsparse_get_mat_fill_mode() . . . 249
2.9.10.9.18 rocsparse_set_mat_diag_type() . . . 249
2.9.10.9.19 rocsparse_get_mat_diag_type() . . . 249
2.9.10.9.20 rocsparse_create_hyb_mat() . . . 249
2.9.10.9.21 rocsparse_destroy_hyb_mat() . . . 250
2.9.10.9.22 rocsparse_create_mat_info() . . . 250
2.9.10.9.23 rocsparse_destroy_mat_info() . . . 250
2.9.10.10 Sparse Level 1 Functions . . . 251
2.9.10.10.1 rocsparse_axpyi() . . . 251
2.9.10.10.2 rocsparse_doti() . . . 252
2.9.10.10.3 rocsparse_gthr() . . . 252
2.9.10.10.4 rocsparse_gthrz() . . . 253
2.9.10.10.5 rocsparse_roti() . . . 254
2.9.10.10.6 rocsparse_sctr() . . . 255
2.9.10.11 Sparse Level 2 Functions . . . 256
2.9.10.11.1 rocsparse_coomv() . . . 256
2.9.10.11.2 rocsparse_csrmv_analysis() . . . 257
2.9.10.11.3 rocsparse_csrmv() . . . 258
2.9.10.11.4 rocsparse_csrmv_analysis_clear() . . . 261
2.9.10.11.5 rocsparse_ellmv() . . . 261
2.9.10.11.6 rocsparse_hybmv() . . . 263
2.9.10.11.7 rocsparse_csrsv_zero_pivot() . . . 264
2.9.10.11.8 rocsparse_csrsv_buffer_size() . . . 264
2.9.10.11.9 rocsparse_csrsv_analysis() . . . 265
2.9.10.11.10rocsparse_csrsv_solve() . . . 267
2.9.10.11.11rocsparse_csrsv_clear() . . . 269
2.9.10.12 Sparse Level 3 Functions . . . 270
2.9.10.12.1 rocsparse_csrmm() . . . 270
2.9.10.13 Preconditioner Functions . . . 273
2.9.10.13.1 rocsparse_csrilu0_zero_pivot() . . . 273
2.9.10.13.2 rocsparse_csrilu0_buffer_size() . . . 274
2.9.10.13.3 rocsparse_csrilu0_analysis() . . . 275
2.9.10.13.4 rocsparse_csrilu0() . . . 276

2.9.10.13.5 rocsparse_csrilu0_clear() 280
2.9.10.14 Sparse Conversion Functions 281
2.9.10.14.1 rocsparse_csr2coo() 281
2.9.10.14.2 rocsparse_coo2csr() 282
2.9.10.14.3 rocsparse_csr2csc_buffer_size() 284
2.9.10.14.4 rocsparse_csr2csc() 284
2.9.10.14.5 rocsparse_csr2ell_width() 287
2.9.10.14.6 rocsparse_csr2ell() 287
2.9.10.14.7 rocsparse_ell2csr_nnz() 289
2.9.10.14.8 rocsparse_ell2csr() 290
2.9.10.14.9 rocsparse_csr2hyb() 291
2.9.10.14.10rocsparse_create_identity_permutation() 292
2.9.10.14.11rocsparse_csrsort_buffer_size() 293
2.9.10.14.12rocsparse_csrsort() 294
2.9.10.14.13rocsparse_coosort_buffer_size() 295
2.9.10.14.14rocsparse_coosort_by_row() 296
2.9.10.14.15rocsparse_coosort_by_column() 298
2.9.11 Deprecated Libraries 300
2.9.11.1 hCRNG 300
2.9.11.2 hipeigen 300
2.9.11.3 clFFT 300
2.9.11.4 clBLAS 300
2.9.11.5 clSPARSE 300
2.9.11.6 clRNG 300
2.9.11.7 hcFFT 301
2.10 ROCm Compiler SDK 301
2.10.1 GCN Native ISA LLVM Code Generator 301
2.10.2 ROCm Code Object Format 301
2.10.3 ROCm Device Library 301
2.10.3.1 OVERVIEW 301
2.10.3.2 BUILDING 301
2.10.3.3 USING BITCODE LIBRARIES 302
2.10.3.4 TESTING 303
2.10.4 ROCr Runtime 303
2.10.4.1 HSA Runtime API and runtime for ROCm 303
2.10.4.2 Source code 304
2.10.4.3 Binaries for Ubuntu & Fedora and installation instructions 304
2.10.4.4 Infrastructure 304
2.10.4.5 Sample 305
2.10.4.6 Known issues 305
2.11 ROCm System Management 305
2.11.1 ROCm-SMI 305
2.11.1.1 Programing ROCm-SMI 310
2.11.2 SYSFS Interface 310
2.11.2.1 Naming and data format standards for sysfs files 310
2.11.2.1.1 Global attributes 311
2.11.2.1.2 Voltages 313
2.11.2.1.3 Fans 316
2.11.2.1.4 PWM 319
2.11.2.1.5 Temperatures 322
2.11.2.1.6 Currents 325
2.11.2.1.7 Power 328
2.11.2.1.8 Energy 329
2.11.2.1.9 Humidity 329

2.11.2.1.10 Alarms . . . 329
2.11.2.1.11 Intrusion detection . . . 332
2.11.2.1.11.1 sysfs attribute writes interpretation . . . 332
2.11.2.1.12 Performance . . . 333
2.11.3 KFD Topology . . . 333
2.11.3.1 HSA Agent Information . . . 333
2.11.3.2 Node Information . . . 334
2.11.3.3 Memory . . . 334
2.11.3.4 Cache . . . 334
2.11.3.5 IO-LINKS . . . 334
2.11.3.6 How to use topology information . . . 334
2.12 ROCm Virtualization & Containers . . . 336
2.12.1 PCIe Passthrough on KVM . . . 336
2.12.1.1 Ubuntu 16.04 . . . 336
2.12.1.2 Fedora 27 or CentOS 7 (1708) . . . 337
2.12.2 ROCm-Docker . . . 338
2.12.2.1 Docker Hub . . . 338
2.12.2.2 ROCm-docker set up guide . . . 338
2.12.2.3 Details . . . 339
2.12.2.4 Building images . . . 339
2.12.2.5 Docker compose . . . 340
2.13 Remote Device Programming . . . 341
2.13.1 ROCmRDMA . . . 341
2.13.1.1 Restrictions and limitations . . . 341
2.13.1.2 ROCmRDMA interface specification . . . 341
2.13.1.3 Data structures . . . 341
2.13.1.4 The function to query ROCmRDMA interface . . . 342
2.13.1.5 The function to query ROCmRDMA interface . . . 342
2.13.1.6 ROCmRDMA interface functions description . . . 343
2.13.2 UCX . . . 344
2.13.2.1 Introduction . . . 344
2.13.2.2 UCX Quick start . . . 344
2.13.2.3 UCX API usage examples . . . 344
2.13.2.4 Running UCX . . . 344
2.13.2.4.1 UCX internal performance tests . . . 344
2.13.2.4.2 OpenMPI and OpenSHMEM with UCX . . . 346
2.13.2.5 Interface to ROCm . . . 347
2.13.2.6 Documentation . . . 347
2.13.2.6.1 High Level Design . . . 347
2.13.2.6.2 Infrastructure and Tools . . . 348
2.13.2.7 FAQ . . . 350
2.13.3 MPI . . . 351
2.13.4 IPC . . . 352
2.13.4.1 Introduction . . . 352
2.14 Deep Learning on ROCm . . . 355
2.14.1 TensorFlow . . . 355
2.14.1.1 ROCm Tensorflow v2.0 Release . . . 355
2.14.1.2 Tensorflow Installation . . . 355
2.14.1.3 Tensorflow More Resources . . . 355
2.14.2 MIOpen . . . 356
2.14.2.1 ROCm MIOpen v2.0 Release . . . 356
2.14.2.2 Porting from cuDNN to MIOpen . . . 356
2.14.2.3 The ROCm 2.7 has prebuilt packages for MIOpen . . . 356
2.14.3 PyTorch . . . 357

2.14.3.1 Building PyTorch for ROCm 357
2.14.3.2 Recommended:Install using published PyTorch ROCm docker image: 357
2.14.3.3 Option 2: Install using PyTorch upstream docker file 358
2.14.3.4 Option 3: Install using minimal ROCm docker file 359
2.14.3.5 Try PyTorch examples 360
2.14.4 Caffe2 360
2.14.4.1 Building Caffe2 for ROCm 360
2.14.4.2 Option 1: Docker image with Caffe2 installed: 361
2.14.4.3 Option 2: Install using Caffe2 ROCm docker image: 361
2.14.4.4 Test the Caffe2 Installation 361
2.14.4.5 Run benchmarks 362
2.14.4.6 Running example scripts 362
2.14.4.7 Building own docker images 362
2.14.5 Deep Learning Framework support for ROCm 363
2.14.6 Tutorials 363
2.15 System Level Debug 363
2.15.1 ROCm Language & System Level Debug, Flags and Environment Variables 363
2.15.1.1 ROCr Error Code 363
2.15.1.2 Command to dump firmware version and get Linux Kernel version 364
2.15.1.3 Debug Flags 364
2.15.1.4 ROCr level env variable for debug 364
2.15.1.5 Turn Off Page Retry on GFX9/Vega devices 364
2.15.1.6 HCC Debug Enviroment Varibles 365
2.15.1.7 HIP Environment Varibles 367
2.15.1.8 OpenCL Debug Flags 368
2.15.1.9 PCIe-Debug 368
2.16 Tutorial 368
2.17 ROCm Glossary 368

Index **371**

We are excited to present ROCm, the first open-source HPC/Hyperscale-class platform for GPU computing that's also programming-language independent. We are bringing the UNIX philosophy of choice, minimalism and modular software development to GPU computing. The new ROCm foundation lets you choose or even develop tools and a language run time for your application.

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

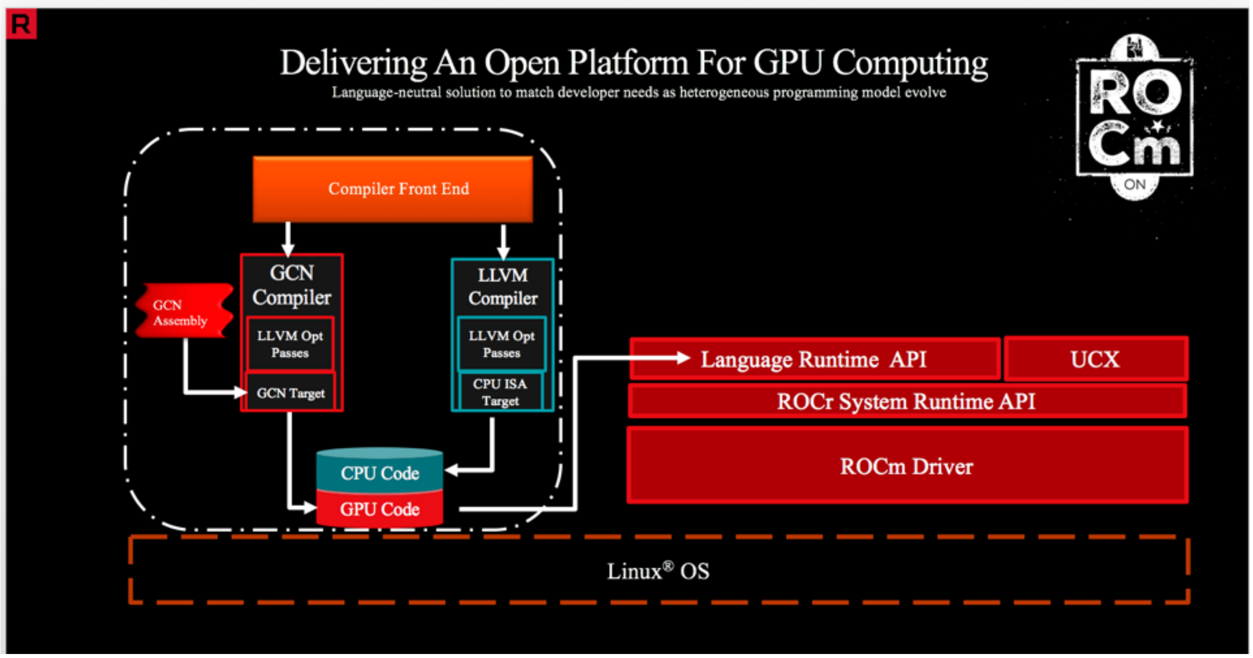
ROCm has a rich system run time with the critical features that large-scale application, compiler and language-run-time development requirements.

CHAPTER

ONE

GOING TO 11: AMPING UP THE PROGRAMMING-LANGUAGE RUN-TIME FOUNDATION

The 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, the Khronos Group's OpenCL, and Continuum's Anaconda Python.



Delivering An Open Platform For GPU Computing

Language-neutral solution to match developer needs as heterogeneous programming model evolve

Compiler Front End

GCN Compiler

LLVM Opt Passes

GCN Target

GCN Assembly

LLVM Compiler

LLVM Opt Passes

CPU ISA Target

CPU Code

GPU Code

Language Runtime API

UCX

ROCr System Runtime API

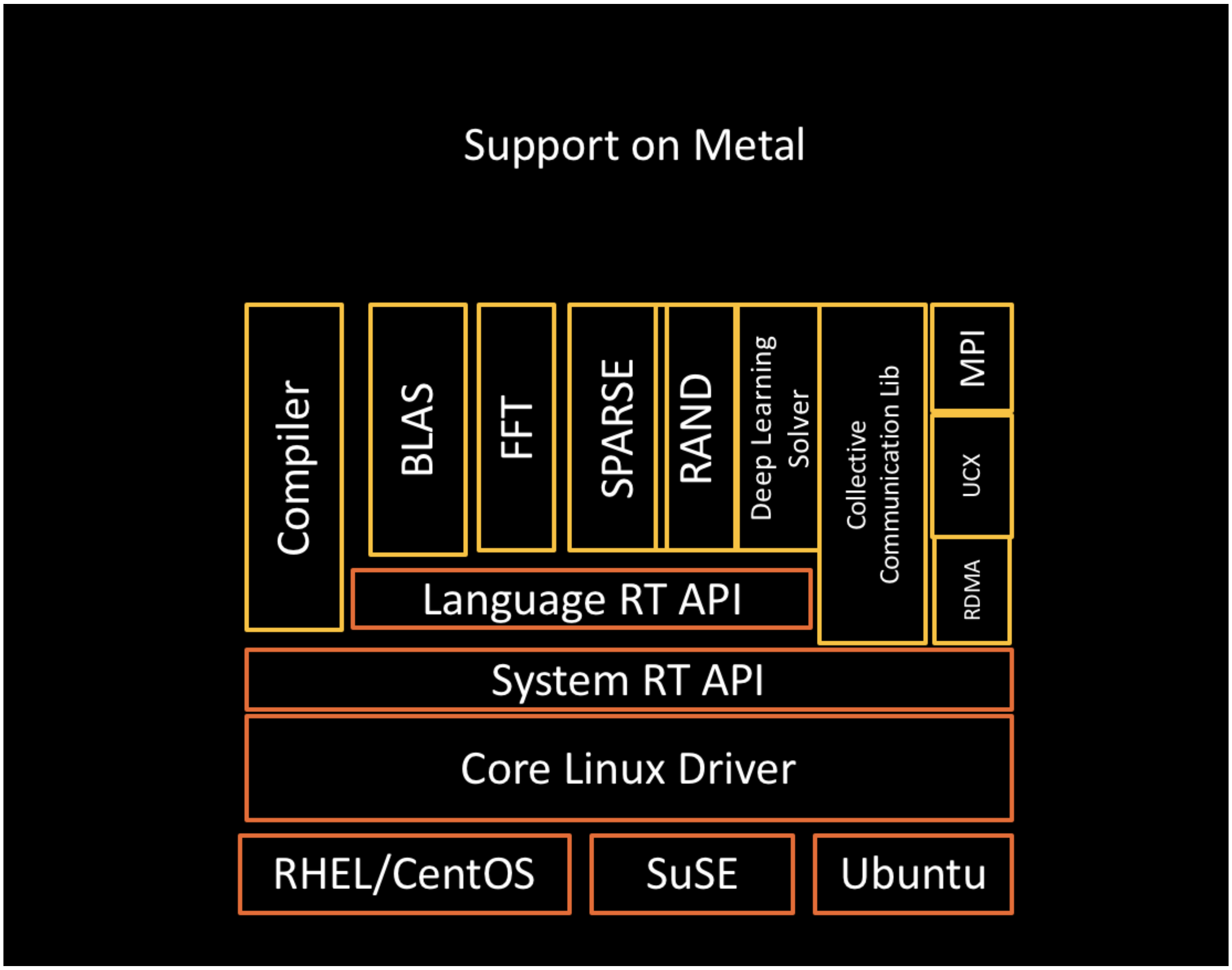
ROCm Driver

Linux® OS

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



CHAPTER

TWO

SOLID COMPILATION FOUNDATION AND LANGUAGE SUPPORT

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

The frontiers of what you can accomplish with ROCm are vast and uncharted. We look forward to working with you to improve the platform so you can use it more effectively in your own projects. Our efforts have opened the door to unique heterogeneous-computing applications that explore what this growing technology can do.

2.1 Quick Start Guide

2.1.1 Current Release Notes

Release Notes
The Release Notes for the ROCm Latest version.

2.1.2 Installation Guide

Installing from AMD ROCm repositories
This guide discusses how to install and check for correct operation of ROCm using AMD ROCm Repository.

Installing from a Debian repository
This guide discusses how to install and check for correct operation of ROCm using Debian repository on Ubuntu.

Installing from an yum repository
This guide describes how to install and check for correct operation of ROCm using yum on RHEL and CentOS 7.5.

Getting ROCm source code
This guide discusses how to modify the open source code base and rebuild the components of ROCm latest version.

Installing ROCk-Kernel only
This guide discusses how to install ROCm Kernel into the system.

FAQ on Installation

This section provides answers for various frequently asked questions related to installation steps and issues faced during installation.

2.1.3 Programming Guide

This guide provides a detailed discussion of the ROCm programming model and programming interface. It describes the hardware implementation and provides guidance on how to achieve maximum performance.

The appendices include a list of all ROCm-enabled devices, detailed description of all extensions to the C language, listings of supported mathematical functions, C++ features supported in host and device code, technical specifications of various devices, and concludes by introducing the low-level driver API.

ROCm Languages

This guide provides information on different ROCm languages. ROCm stack offers multiple programming-language choices found in this section.

HC Programing Guide

This guide provides a detailed discussion on The Heterogeneous Compute programming installation requirements, methods to install on various platforms and how to build it from source

HC Best Practices

This section deals with detailed working with HCC, build the program, Build-in Macros, HCC Profiler mode and API Documentaion.

HIP Programing Guide

This guide provides a detailed discussion of The HIP programming, installation requirements, methods to install on various platfroms and how to build it from source

HIP Best Practices

This section Provides details regarding various concepts of HIP Porting, Debugging, Bugs, FAQ and other aspects of the HIP.

OpenCL Programing Guide

This guide provides a detailed discussion of The OpenCL Architecture, AMD Implementation, Profiling, and other aspects of Opencl.

OpenCL Best Practices

This section provides information on Performance and optimization for various device types such as GCN devices.

2.1.4 ROCm GPU Tunning Guides

GFX7 Tuning Guide

– In-Progress

GFX8 Tuning Guide
– In-Progress

Vega Tuning Guide
– In-Progress

2.1.5 GCN ISA Manuals

GCN 1.1
This Section gives information on ISA Manual for Hawaii (Sea Islands Series Instruction Set Architecture)

GCN 2.0
This Section gives information on ISA Manual for Fiji and Polaris (AMD Accelerated Parallel Processing technology)

Vega
This section provides "Vega" Instruction Set Architecture, Program Organization, Mode register and more details.

Inline GCN ISA Assembly Guide
This section covers various concepts of AMDGCN Assembly, DS Permute Instructions, Parameters to a Kernel, GPR Counting.

2.1.6 ROCm API References

Here API References are listed out for users

ROCr System Runtime API
ROCr System Runtime API Details are listed here

HCC Language Runtime API
HCC Language Runtime APIDetails are listed here

HIP Language Runtime API
HIP Language Runtime API Details are listed here

HIP Math API
Here HIP Math API are listed with sample working classes

Thrust API Documentation
Here you can find all the details on installation, working of Thrust Library and Thrust API List

Math Library API's
HIP MAth API with hcRNG, clBLAS, clSPARSE API's.

Deep Learning API's
Here MIOpen API and MIOpenGEMM API are listed.

2.1.7 ROCm Tools

HCC
Complete description of Heterogeneous Compute Compiler has been listed and documented.

GCN Assembler and Disassembler
This Section provides details regarding GCN in-detail.

GCN Assembler Tools
In this Section, information related to AMDGPU ISA assembler is documented.

ROCm-GDB
Complete Documentaion of ROCm-GDB tool is provided here. Installtion, Build steps and working of Debugger and API related to it has been documented here.

ROCm-Profiler
This section gives details on Radeon Compute Profiler- performance analysis tool,and we have details on how to clone and use it.

ROCm-Tracer
This section gives Details on ROCm Tracer, which provides a generic independent from specific runtime profiler to trace API and asyncronous activity. Here we have details on library source tree, steps to build and run test.

CodeXL
This section provides details on CodeXL, a comprehensive tool suite. The Documentaion of Installation and builds and other details related to Codexl is given.

GPUperfAPI
This section provides details on GPU Performance API. The content related to how to clone, system requiments and source code directory layout can be found.

ROCm Binary Utilities
– In-progess

2.1.8 ROCm Libraries

rocFFT

This section provides details on rocFFT,it is a AMD's software library compiled with the CUDA compiler using HIP tools for running on Nvidia GPU devices.

rocBLAS

This section provides details on rocBLAS, it is a library for BLAS on ROCm.rocBLAS is implemented in the HIP programming language and optimized for AMD's latest discrete GPUs.

hipBLAS

This section provides details on hipBLAS, it is a BLAS marshalling library, with multiple supported backends. hipBLAS exports an interface that does not require the client to change. Currently,it supports rocblas and cuBLAS as backends.

hcRNG

This section provides details on hcRNG. It is a software library ,where uniform random number generators targeting the AMD heterogeneous hardware via HCC compiler runtime is implemented..

hipeigen

This section provides details on Eigen.It is a C++ template library which provides linear algebra for matrices, vectors, numerical solvers, and related algorithms.

clFFT

This section provides details on clFFT.It is a software library which contains FFT functions written in OpenCL,and clFFt also supports running on CPU devices to facilitate debugging and heterogeneous programming.

clBLAS

This section provides details on clBLAS. It makes easier for developers to utilize the inherent performance and power efficiency benefits of heterogeneous computing.

clSPARSE

This section provides details on clSPARSE, it is an OpenCL library which implements Sparse linear algebra routines.

clRNG

This section provides details on clRNG,This is a library for uniform random number generation in OpenCL.

hcFFT

This section provides details on hcFFT, it hosts the HCC based FFT Library and targets GPU acceleration of FFT routines on AMD devices.

Tensile

This section provides details on Tensile. It is a tool for creating a benchmark-driven backend library for GEMMs,N-dimensional tensor contractions and multiplies two multi-dimensional objects together on a GPU.

rocALUTION

This section provides details on rocALUTION. It 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. It can be seen as middle-ware between different parallel backends and application specific packages.

rocSPARSE

This section provides details on rocSPARSE.It 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.

rocThrust

This section provides details on rocThrust. It is a parallel algorithmn library.

hipCUB This section provides details on hipCUB.

It is a thin wrapper library on top of rocPRIM or CUB. It enables developers to port the project using CUB library to the HIP layer and to

run them on AMD hardware.

ROCm SMI Library This section provides details on 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 aplications.

RCCL This section provides details on ROCm Communications Collectives Library. It is a stand alone library of standard collective communication routines for GPUS, implememting all-reduce, all gather, reduce, broadcast, and reduce scatter.

2.1.9 ROCm Compiler SDK

GCN Native ISA LLVM Code Generator

This section provide complete description on LLVM such as introduction, Code Object, Code conventions, Source languages, etc.,

ROCm Code Object Format

This section describes about application binary interface (ABI) provided by the AMD, implementation of the HSA runtime. It also provides details on Kernel, AMD Queue and Signals.

ROCm Device Library

Documentation on instruction related to ROCm Device Library overview,Building and Testing related information with respect to Device Library is provided.

ROCr Runtime

This section refers the user-mode API interfaces and libraries necessary for host applications to launch compute kernels to available HSA ROCm kernel agents. we can find installation details and Infrastructure details related to ROCr.

2.1.10 ROCm System Management

ROCm-SMI

ROCm System Management Interface a complete guide to use and work with rocm-smi tool.

SYSFS Interface

This section provides information on sysfs file structure with details related to file structure related to system are captured in sysfs.

KFD Topology

KFD Kernel Topology is the system file structure which describes about AMD GPU related information such as nodes, Memory, Cache and IO-links.

2.1.11 ROCm Virtualization & Containers

PCIe Passthrough on KVM

Here PCIe Passthrough on KVM is described. A KVM-based instructions assume a headless host with an input/output memory management unit (IOMMU) to pass peripheral devices such as a GPU to guest virtual machines.more information can be found on the same here.

ROCm-Docker

A framework for building the software layers defined in the Radeon Open Compute Platform into portable docker images. Detailed Information related to ROCm-Docker can be found.

2.1.12 Remote Device Programming

ROCnRDMA

ROCmRDMA is the solution designed to allow third-party kernel drivers to utilize DMA access to the GPU memory. Complete indoemation related to ROCmRDMA is Documented here.

UCX

This section gives information related to UCX, How to install, Running UCX and much more

MPI

This section gives information related to MPI.

IPC

This section gives information related to IPC.

2.1.13 Deep Learning on ROCm

This section provides details on ROCm Deep Learning concepts.

Porting from cuDNN to MIOpen
The porting guide highlights the key differences between the current cuDNN and MIOpen APIs.

Deep Learning Framework support for ROCm
This section provides detailed chart of Frameworks supported by ROCm and repository details.

Tutorials
Here Tutorials on different DeepLearning Frameworks are documented.

2.1.14 System Level Debug

ROCm Language & System Level Debug, Flags and Environment Variables
Here in this section we have details regardinf various system related debugs and commands for isssues faced while using ROCm.

2.1.15 Tutorial

This section Provide details related to few Concepts of HIP and other sections.

2.1.16 ROCm Glossary

ROCm Glossary gives highlight concept and their main concept of how they work.

2.2 Current Release Notes

2.2.1 Hotfix release ROCm 2.7.1

This release is a hotfix release for ROCm release 2.7.1, and addresses the defect mentioned below. The features and enhancements as mentioned in ROCm 2.7 remain relevant to ROCm release 2.7.1 as well.

2.2.1.1 Defect fixed in ROCm 2.7.1

rocprofiler –hiptrace and –hsatrace fails to load roctracer library

In ROCm 2.7.1, 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 ex:

```
/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

2.2.1.2 Upgrading from ROCm 2.7 to 2.7.1

To upgrade, please remove 2.7 completely as specified here or here, and install 2.7.1 as per instructions here.

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

2.2.2 New features and enhancements in ROCm 2.7

2.2.2.1 [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.

2.2.2.2 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

2.2.2.3 RAS

Added support for RAS on Radeon Instinct MI50, including:

- Memory error detection
- Memory error detection counter

2.2.2.4 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.

2.2.3 New features and enhancements in ROCm 2.6

2.2.3.1 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

2.2.3.2 [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 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.

2.2.3.3 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

2.2.3.4 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

2.2.3.5 Bloat16 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.

2.2.3.6 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.

2.2.3.7 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

2.2.3.8 ROCm-smi-lib enhancements

- Documentation updates
- Improvements to *name_get functions

2.2.3.9 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.

2.2.3.10 rocFFT enhancements

- Added: Debian package with FFT test, benchmark, and sample programs
- Improved: hipFFT interfaces
- Improved: rocFFT CPU reference code, plan generation code and logging code

Features and enhancements introduced in previous versions of ROCm can be found in version_history.md

2.2.4 New features and enhancements in ROCm 2.5

2.2.4.1 UCX 1.6 support

Support for UCX version 1.6 has been added.

2.2.4.2 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

2.2.4.3 ROCm-SMI enhancements

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

2.2.4.4 [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 06c8aa7a3bbd91cda2fd6255ec82aad21fa1c0d5 commit or later.

2.2.4.5 rocSparse optimization on Radeon Instinct MI50 and MI60

This release includes performance optimizations for csrsv routines in the rocSparse library.

2.2.4.6 [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 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.

2.2.4.7 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.

2.2.4.8 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.

Features and enhancements introduced in previous versions of ROCm can be found in version_history.md

2.2.5 New features and enhancements in ROCm 2.4

2.2.5.1 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

2.2.5.2 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.

2.2.6 New features and enhancements in ROCm 2.3

2.2.6.1 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

2.2.6.2 MIVisionX, v1.1 - ONNX

ONNX parser changes to adjust to new file formats

2.2.6.3 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

2.2.6.4 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

2.2.6.5 Caffe2 - mGPU support

Multi-gpu support is enabled for Caffe2.

2.2.6.6 rocTracer library, ROCm tracing API for collecting runtimes API and asynchronous GPU activity traces

HIP/HCC domains support is introduced in rocTracer library.

2.2.6.7 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.

2.2.6.8 Prioritized L1/L2/L3 BLAS (functional)

Functional implementation of BLAS L1/L2/L3 functions

2.2.6.9 BLAS - tensile optimization

Improvements and optimizations with tensile

2.2.6.10 MIOpen Int8 support

Support for int8

2.2.7 New features and enhancements in ROCm 2.2

2.2.7.1 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.

2.2.7.2 DGEMM and DTRSM Optimization

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

2.2.7.3 Caffe2

Added support for multi-GPU training

2.2.8 New features and enhancements in ROCm 2.1

2.2.8.1 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

2.2.8.2 Improvements to ROCM-SMI tool -

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

2.2.8.3 DGEMM Optimizations -

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

2.2.9 New features and enhancements in ROCm 2.0

Features and enhancements introduced in previous versions of ROCm can be found in version_history.md

2.2.9.1 Adds support for RHEL 7.6 / CentOS 7.6 and Ubuntu 18.04.1

2.2.9.2 Adds support for Vega 7nm, Polaris 12 GPUs

2.2.9.3 Introduces MIVisionX

A comprehensive computer vision and machine intelligence libraries, utilities and applications bundled into a single toolkit.

2.2.9.4 Improvements to ROCm Libraries

- rocSPARSE & hipSPARSE
- rocBLAS with improved DGEMM efficiency on Vega 7nm

2.2.9.5 MIOpen

- This release contains general bug fixes and an updated performance database
- Group convolutions backwards weights performance has been improved
- RNNs now support fp16

2.2.9.6 Tensorflow multi-gpu and Tensorflow FP16 support for Vega 7nm

- TensorFlow v1.12 is enabled with fp16 support

2.2.9.7 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

2.2.9.8 Improvements to ROCProfiler tool

- Support for Vega 7nm

2.2.9.9 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.

2.2.9.10 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-openclc.pdf

2.2.9.11 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

2.2.9.12 Kubernetes support

2.2.9.13 Removed features

- HCC: removed support for C++AMP

2.2.10 New features and enhancements in ROCm 1.9.2

2.2.10.1 RDMA(MPI) support on Vega 7nm

- Support ROCnRDMA based on Mellanox InfiniBand.

2.2.10.2 Improvements to HCC

- Improved link time optimization.

2.2.10.3 Improvements to ROCProfiler tool

- General bug fixes and implemented versioning APIs.

2.2.10.4 Critical bug fixes

2.2.11 New features and enhancements in ROCm 1.9.1

2.2.11.1 Added DPM support to Vega 7nm

Dynamic Power Management feature is enabled on Vega 7nm.

2.2.11.2 Fix for 'ROCm profiling' "Version mismatch between HSA runtime and libhsa-runtime-tools64.so.1" error

2.2.12 New features and enhancements in ROCm 1.9.0

2.2.12.1 Preview for Vega 7nm

- Enables developer preview support for Vega 7nm

2.2.12.2 System Management Interface

- Adds support for the ROCm SMI (System Management Interface) library, which provides monitoring and management capabilities for AMD GPUs.

2.2.12.3 Improvements to HIP/HCC

- Support for gfx906
- 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

2.2.12.4 Preview for rocprof Profiling Tool

Developer preview (alpha) of profiling tool 'rpl_run.sh', cmd-line front-end for rocProfiler, 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 location is: /opt/rocm/bin/rpl_run.sh

2.2.12.5 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

2.2.12.6 New distribution support

- Binary package support for Ubuntu 18.04

2.2.12.7 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
```

2.3 ROCm Installation Guide

2.3.1 Are You Ready to ROCK?

The ROCm Platform brings a rich foundation to advanced computing by seamlessly integrating the CPU and GPU with the goal of solving real-world problems. This software enables the high-performance operation of AMD GPUs for computation oriented tasks in the Linux operating system.

2.3.2 Current ROCm Version: 2.7.1

2.3.3 The latest ROCm platform - ROCm 2.7

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 Development Tools
  - HCC compiler
  - HIP
  - ROCm Device Libraries
  - ROCm OpenCL, which is created from the following components:
    - ROCm OpenCL Runtime
    - ROCm OpenCL Driver

        - The ROCm OpenCL compiler, which is created from the following components:
            - ROCm LLVM OCL
            - ROCm LLVM HCC
            - ROCm Clang
            - ROCm lld OCL
            - ROCm lld HCC
            - ROCm Device Libraries
    - ROCM Clang-OCL Kernel Compiler
    - **'Asynchronous Task and Memory Interface'_**
    - ROCr Debug Agent
    - ROCm Code Object Manager
    - ROC Profiler
    - ROC Tracer
    - Radeon Compute Profiler
    - Example Applications:
        - HCC Examples
        - HIP Examples
- ROCm Libraries
    - rocBLAS
    - hipBLAS
    - rocFFT
    - rocRAND
    - rocSPARSE
    - hipSPARSE
    - rocALUTION
    - MIOpenGEMM
    - MIOpen
    - rocThrust
    - ROCm SMI Lib
    - RCCL
    - MIVisionX
    - hipCUB

2.3.4 Hardware Support

ROCm is focused on using AMD GPUs to accelerate computational tasks such as machine learning, engineering workloads, and scientific computing. In order to focus our development efforts on these domains of interest, ROCm supports a targeted set of hardware configurations which are detailed further in this section.

2.3.4.1 Supported GPUs

Because the ROCm Platform has a focus on particular computational domains, we offer official support for a selection of AMD GPUs that are designed to offer good performance and price in these domains.

ROCm officially supports AMD GPUs that use following chips:

- GFX8 GPUs
  - "Fiji" chips, such as on the AMD Radeon R9 Fury X and Radeon Instinct MI8
  - "Polaris 10" chips, such as on the AMD Radeon RX 580 and Radeon Instinct MI6
  - "Polaris 11" chips, such as on the AMD Radeon RX 570 and Radeon Pro WX 4100
  - "Polaris 12" chips, such as on the AMD Radeon RX 550 and Radeon RX 540
- GFX9 GPUs
  - "Vega 10" chips, such as on the AMD Radeon RX Vega 64 and Radeon Instinct MI25
  - "Vega 7nm" chips, such as on the Radeon Instinct MI50, Radeon Instinct MI60 or AMD Radeon VII

ROCm is a collection of software ranging from drivers and runtimes to libraries and developer tools. Some of this software may work with more GPUs than the "officially supported" list above, though AMD does not make any official claims of support for these devices on the ROCm software platform. The following list of GPUs are enabled in the ROCm software, though full support is not guaranteed:

- **GFX8 GPUs**
  - "Polaris 11" chips, such as on the AMD Radeon RX 570 and Radeon Pro WX 4100
  - "Polaris 12" chips, such as on the AMD Radeon RX 550 and Radeon RX 540
- GFX7 GPUs
  - "Hawaii" chips, such as the AMD Radeon R9 390X and FirePro W9100

As described in the next section, GFX8 GPUs require PCI Express 3.0 (PCIe 3.0) with support for PCIe atomics. This requires both CPU and motherboard support. GFX9 GPUs, by default, also require PCIe 3.0 with support for PCIe atomics, but they can operate in most cases without this capability.

The integrated GPUs in AMD APUs are not officially supported targets for ROCm. As descried below , "Carrizo", "Bristol Ridge", and "Raven Ridge" APUs are enabled in our upstream drivers and the ROCm OpenCL runtime. However, they are not enabled in our HCC or HIP runtimes, and may not work due to motherboard or OEM hardware limitations. As such, they are not yet officially supported targets for ROCm.

For a more detailed list of hardware support, please see the following documentation.

2.3.4.2 Supported CPUs

As described above, GFX8 GPUs require PCIe 3.0 with PCIe atomics in order to run ROCm. In particular, the CPU and every active PCIe point between the CPU and GPU require support for PCIe 3.0 and PCIe atomics. The CPU root must indicate PCIe AtomicOp Completion capabilities and any intermediate switch must indicate PCIe AtomicOp Routing capabilities.

Current CPUs which support PCIe Gen3 + PCIe Atomics are:

- AMD Ryzen CPUs;
- The CPUs in AMD Ryzen APUs;
- AMD Ryzen Threadripper CPUs
- AMD EPYC CPUs;

- Intel Xeon E7 v3 or newer CPUs;
- Intel Xeon E5 v3 or newer CPUs;
- Intel Xeon E3 v3 or newer CPUs;
- Intel Core i7 v4, Core i5 v4, Core i3 v4 or newer CPUs (i.e. Haswell family or newer).
- Some Ivy Bridge-E systems

Beginning with ROCm 1.8, GFX9 GPUs (such as Vega 10) no longer require PCIe atomics. We have similarly opened up more options for number of PCIe lanes. GFX9 GPUs can now be run on CPUs without PCIe atomics and on older PCIe generations, such as PCIe 2.0. This is not supported on GPUs below GFX9, e.g. GFX8 cards in the Fiji and Polaris families.

If you are using any PCIe switches in your system, please note that PCIe Atomics are only supported on some switches, such as Broadcom PLX. When you install your GPUs, make sure you install them in a PCIe 3.0 x16, x8, x4, or x1 slot attached either directly to the CPU's Root I/O controller or via a PCIe switch directly attached to the CPU's Root I/O controller.

In our experience, many issues stem from trying to use consumer motherboards which provide physical x16 connectors that are electrically connected as e.g. PCIe 2.0 x4, PCIe slots connected via the Southbridge PCIe I/O controller, or PCIe slots connected through a PCIe switch that does not support PCIe atomics.

If you attempt to run ROCm on a system without proper PCIe atomic support, you may see an error in the kernel log (`dmesg`):

```
kfd: skipped device 1002:7300, PCI rejects atomics
```

Experimental support for our Hawaii (GFX7) GPUs (Radeon R9 290, R9 390, FirePro W9100, S9150, S9170) does not require or take advantage of PCIe Atomics. However, we still recommend that you use a CPU from the list provided above for compatibility purposes.

2.3.4.3 Not supported or limited support under ROCm

2.3.4.3.1 Limited support

- ROCm 2.7.x should support PCIe 2.0 enabled CPUs such as the AMD Opteron, Phenom, Phenom II, Athlon, Athlon X2, Athlon II and older Intel Xeon and Intel Core Architecture and Pentium CPUs. However, we have done very limited testing on these configurations, since our test farm has been catering to CPUs listed above. This is where we need community support. *If you find problems on such setups, please report these issues*.
- Thunderbolt 1, 2, and 3 enabled breakout boxes should now be able to work with ROCm. Thunderbolt 1 and 2 are PCIe 2.0 based, and thus are only supported with GPUs that do not require PCIe 3.0 atomics (e.g. Vega 10). However, we have done no testing on this configuration and would need community support due to limited access to this type of equipment.
- AMD "Carrizo" and "Bristol Ridge" APUs are enabled to run OpenCL, but do not yet support HCC, HIP, or our libraries built on top of these compilers and runtimes.
  - As of ROCm 2.1, "Carrizo" and "Bristol Ridge" require the use of upstream kernel drivers.
  - In addition, various "Carrizo" and "Bristol Ridge" platforms may not work due to OEM and ODM choices when it comes to key configurations parameters such as inclusion of the required CRAT tables and IOMMU configuration parameters in the system BIOS.
  - Before purchasing such a system for ROCm, please verify that the BIOS provides an option for enabling IOMMUv2 and that the system BIOS properly exposes the correct CRAT table. Inquire with your vendor about the latter.

- AMD “Raven Ridge” APUs are enabled to run OpenCL, but do not yet support HCC, HIP, or our libraries built on top of these compilers and runtimes.
  - As of ROCm 2.1, “Raven Ridge” requires the use of upstream kernel drivers.
  - In addition, various “Raven Ridge” platforms may not work due to OEM and ODM choices when it comes to key configurations parameters such as inclusion of the required CRAT tables and IOMMU configuration parameters in the system BIOS.
  - Before purchasing such a system for ROCm, please verify that the BIOS provides an option for enabling IOMMUv2 and that the system BIOS properly exposes the correct CRAT table. Inquire with your vendor about the latter.

2.3.4.3.2 Not supported

- “Tonga”, “Iceland”, “Vega M”, and “Vega 12” GPUs are not supported in ROCm 2.7.x
- We do not support GFX8-class GPUs (Fiji, Polaris, etc.) on CPUs that do not have PCIe 3.0 with PCIe atomics.
  - As such, we do not support AMD Carrizo and Kaveri APUs as hosts for such GPUs.
  - Thunderbolt 1 and 2 enabled GPUs are not supported by GFX8 GPUs on ROCm. Thunderbolt 1 & 2 are based on PCIe 2.0.

2.3.5 Supported Operating Systems - New operating systems available

The ROCm 2.7.x platform supports the following operating systems:

- Ubuntu 16.04.5(Kernel 4.15) and 18.04.2(Kernel 4.18)
- CentOS 7.6 (Using devtoolset-7 runtime support)
- RHEL 7.6 (Using devtoolset-7 runtime support)

2.3.5.1 ROCm support in upstream Linux kernels

As of ROCm 1.9.0, the ROCm user-level software is compatible with the AMD drivers in certain upstream Linux kernels. As such, users have the option of either using the ROCK kernel driver that are part of AMD’s ROCm repositories or using the upstream driver and only installing ROCm user-level utilities from AMD’s ROCm repositories.

These releases of the upstream Linux kernel support the following GPUs in ROCm:

- 4.17: Fiji, Polaris 10, Polaris 11
- 4.18: Fiji, Polaris 10, Polaris 11, Vega10
- 4.20: Fiji, Polaris 10, Polaris 11, Vega10, Vega 7nm

The upstream driver may be useful for running ROCm software on systems that are not compatible with the kernel driver available in AMD’s repositories. For users that have the option of using either AMD’s or the upstreamed driver, there are various tradeoffs to take into consideration:

	Using AMD's *rock-dkms* package	Using the upstream kernel driver
Pros	More GPU features, and they are enabled earlier	Includes the latest Linux kernel features
	Tested by AMD on supported distributions	May work on other distributions and with custom kernels
	Supported GPUs enabled regardless of kernel version	
	Includes the latest GPU firmware	
Cons	May not work on all Linx distributions or versions	Features and hardware support varies depending on kernel version
	Not currently supported on kernels newer than 4.18.	Limits GPU's usage of system memory to 3/8 of system memory
		IPC and RDMA capabilities not yet enabled
		Not tested by AMD to the same level as *rock-dkms* package
		Does not include most up-to-date firmware

2.3.6 Installing from AMD ROCm repositories

AMD hosts both Debian and RPM repositories for the ROCm 2.7.x packages at this time.

The packages in the Debian repository have been signed to ensure package integrity.

2.3.6.1 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:

- ROCm Core Components
  - ROCk Kernel Driver: `rock-dkms`
  - ROCr Runtime: `hsa-rocr-dev`, `hsa-ext-rocr-dev`
  - ROCt Thunk Interface: `hsakmt-roct`, `hsakmt-roct-dev`
- ROCm Support Software
  - ROCm SMI: `rocm-smi`
  - ROCm cmake: `rocm-cmake`
  - rocminfo: `rocminfo`
  - ROCm Bandwidth Test: `rocm_bandwidth_test`
- ROCm Development Tools
  - HCC compiler: `hcc`
  - HIP: `hip_base`, `hip_doc`, `hip_hcc`, `hip_samples`
  - ROCm Device Libraries: `rocm-device-libs`
  - ROCm OpenCL: `rocm-opencl`, `rocm-opencl-devel` (on RHEL/CentOS), `rocm-opencl-dev` (on Ubuntu)

  - ROCM Clang-OCL Kernel Compiler: `rocm-clang-ocl`
  - Asynchronous Task and Memory Interface (ATMI): `atmi`
  - ROCr Debug Agent: `rocr_debug_agent`
  - ROCm Code Object Manager: `comgr`
  - ROC Profiler: `rocprofiler-dev`
  - ROC Tracer: `roctracer-dev`
  - Radeon Compute Profiler: `rocm-profiler`

- ROCm Libraries
  - rocBLAS: `rocblas`
  - hipBLAS: `hipblas`
  - rocFFT: `rocfft`
  - rocRAND: `rocrand`
  - rocSPARSE: `rocsparse`
  - hipSPARSE: `hipsparse`
  - rocALUTION: `rocalution:`
  - MIOpenGEMM: `miopengemm`
  - MIOpen: `MIOpen-HIP` (for the HIP version), `MIOpen-OpenCL` (for the OpenCL version)
  - ROCm SMI Lib: `rocm_smi_lib64`
  - RCCL: `rccl`
  - MIVisionX: `mivisionx`
  - rocThrust: `rocThrust`
  - hipCUB: `hipCUB`

To make it easier to install ROCm, the AMD binary repos 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
      |--hsa-rocr-dev
      |--hsa-ext-rocr-dev
      |--hsakmt-roct
      |--hsakmt-roct-dev
      |--rocm-cmake
      |--rocm-device-libs
      |--hcc
```

(continues on next page)

(continued from previous page)

```
      |--hip_base
      |--hip_doc
      |--hip_hcc
      |--hip_samples
      |--rocm-smi
      |--hsa-amd-aqlprofile
      |--comgr
      |--rocr_debug_agent
      |--rocm-utils
          |-- rocminfo
          \-- rocm-clang-ocl # This will cause OpenCL to be installed

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

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

2.3.6.2 Ubuntu Support - installing from a Debian repository

The following directions show how to install ROCm on supported Debian-based systems such as Ubuntu 18.04. 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. As such, users may want to skip the `rocm-dkms` and `rock-dkms` packages, as described above, and instead **‘use the upstream kernel driver‘_**.

2.3.6.2.1 First make sure your system is up to date

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

2.3.6.2.2 Add the ROCm apt repository

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

```
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
```

The gpg key might change, so it may need to be updated when installing a new release. If the key signature verification is failed while update, please re-add the key from ROCm apt repository. The current rocm.gpg.key is not available in a standard key ring distribution, but has the following sha1sum hash:

```
f7f8147431c75e505c58a6f3a3548510869357a6 rocm.gpg.key
```

2.3.6.2.3 Install

Next, update the apt repository list and install the `rocm-dkms` meta-package:

```
sudo apt update
sudo apt install rocm-dkms
```

2.3.6.2.4 Next set your permissions

Users will need to be in the `video` group in order to have access to the GPU. As such, you should ensure that your user account is a member of the `video` group prior to using ROCm. You can find which groups you are a member of with the following command:

```
groups
```

To add yourself to the video group you will need the sudo password and can use the following command:

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

You may want to ensure that any future users you add to your system are put into the "video" group by default. To do that, you can run the following commands:

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

Once complete, reboot your system.

2.3.6.2.5 Test basic ROCm installation

After rebooting the system run the following commands to verify that the ROCm installation was successful. If you see your GPUs listed by both of these commands, you should be ready to go!

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

Note that, to make running ROCm programs easier, you may wish to put 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
```

If you have an install issue please read this FAQ.

2.3.6.2.6 Performing an OpenCL-only Installation of ROCm

Some users may want to install a subset of the full ROCm installation. In particular, 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 apt-get install dkms rock-dkms rocm-opencl-dev
```

2.3.6.2.7 How to uninstall from Ubuntu 16.04 or Ubuntu 18.04

To uninstall the ROCm packages installed in the above directions, you can execute;

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

2.3.6.2.8 Installing development packages for cross compilation

It is often useful to develop and test on different systems. For example, some development or build systems may not have an AMD GPU installed. In this scenario, you may prefer to avoid installing the ROCK kernel driver to your development system.

In this case, install the development subset of packages:

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

> **Note:** To execute ROCm enabled apps you will require a system with the full ROCm driver stack installed

2.3.6.2.9 Using Debian-based ROCm with upstream kernel drivers

As described in the above section about upstream Linux kernel support, users may want to try installing ROCm user-level software without installing AMD's custom ROCK kernel driver. Users who do want to use upstream kernels can run the following commands instead of installing `rocm-dkms`

```
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
```

2.3.6.3 CentOS/RHEL 7 (7.6) Support

The following directions show how to install ROCm on supported RPM-based systems such as CentOS 7.6. These directions may not work as written on unsupported RPM-based distributions. For example, Fedora may work but may not be compatible with the `rock-dkms` kernel driver. As such, users may want to skip the `rocm-dkms` and `rock-dkms` packages, as described above, and instead use the upstream kernel driver.

Support for CentOS/RHEL 7 was added in ROCm 1.8, but ROCm requires a special runtime environment provided by the RHEL Software Collections and additional dkms support packages to properly install and run.

2.3.6.3.1 Preparing RHEL 7 (7.6) for installation

RHEL is a subscription-based operating system, and you must enable several external repositories to enable installation of the devtoolset-7 environment and the DKMS support files. These steps are not required for CentOS.

First, the subscription for RHEL must be enabled and attached to a pool id. Please see Obtaining an RHEL image and license page for instructions on registering your system with the RHEL subscription server and attaching to a pool id.

Second, enable the following repositories:

```
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
```

Third, enable additional repositories by downloading and installing the epel-release-latest-7 repository RPM:

```
sudo rpm -ivh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

2.3.6.3.2 Install and setup Devtoolset-7

To setup the Devtoolset-7 environment, follow the instructions on this page:

https://www.softwarecollections.org/en/scls/rhscl/devtoolset-7/

Note that devtoolset-7 is a Software Collections package, and it is not supported by AMD.

2.3.6.3.3 Prepare CentOS/RHEL (7.6) for DKMS Install

Installing kernel drivers on CentOS/RHEL 7.6 requires dkms tool being installed:

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

2.3.6.3.4 Installing ROCm on the system

It is recommended to remove previous ROCm installations before installing the latest version to ensure a smooth installation.

At this point ROCm can be installed on the target system. 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
```

The repo's URL should point to the location of the repositories repodata database. Install ROCm components using these commands:

```
sudo yum install rocm-dkms
```

The rock-dkms component should be installed and the `/dev/kfd` device should be available on reboot.

2.3.6.3.5 Set up permissions

Ensure that your user account is a member of the "video" or "wheel" group prior to using the ROCm driver. You can find which groups you are a member of with the following command:

```
groups
```

To add yourself to the video (or wheel) group you will need the sudo password and can use the following command:

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

You may want to ensure that any future users you add to your system are put into the "video" group by default. To do that, you can run the following commands:

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

Current release supports CentOS/RHEL 7.6. If users want to update the OS version, they should completely remove ROCm packages before updating to the latest version of the OS, to avoid DKMS related issues.

Once complete, reboot your system.

2.3.6.3.6 Test basic ROCm installation

After rebooting the system run the following commands to verify that the ROCm installation was successful. If you see your GPUs listed by both of these commands, you should be ready to go!

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

Note that, to make running ROCm programs easier, you may wish to put 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
```

If you have an install issue please read this FAQ.

2.3.6.3.7 Performing an OpenCL-only Installation of ROCm

Some users may want to install a subset of the full ROCm installation. In particular, 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
```

2.3.6.3.8 Compiling applications using HCC, HIP, and other ROCm software

To compile applications or samples, please use gcc-7.2 provided by the devtoolset-7 environment. To do this, compile all applications after running this command:

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

2.3.6.3.9 How to uninstall ROCm from CentOS/RHEL 7.6

To uninstall the ROCm packages installed by the above directions, you can execute:

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

2.3.6.3.10 Installing development packages for cross compilation

It is often useful to develop and test on different systems. For example, some development or build systems may not have an AMD GPU installed. In this scenario, you may prefer to avoid installing the ROCK kernel driver to your development system.

In this case, install the development subset of packages:

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

> **Note:** To execute ROCm enabled apps you will require a system with the full ROCm driver stack installed

2.3.6.3.11 Using ROCm with upstream kernel drivers

As described in the above section about upstream Linux kernel support, use rs may want to try installing ROCm user-level software without installing AMD's custom ROCK kernel driver. Users who do want to use upstream kernels can run the following commands instead of installing `rocm-dkms`

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

2.3.7 Known issues / workarounds

2.3.7.1 rocprofiler –hiptrace and –hsatrace fails to load roctracer library

In ROCm 2.7, rocprofiler –hiptrace and –hsatrace fails to load roctracer library. The workaround is to create two links under '/opt/rocm': $ ln -s /opt/rocm/roctracer/lib/libroctracer64.so /opt/rocm/rocprofiler/lib/libroctracer64.so $ ln -s /opt/rocm/roctracer/tool/libtracer_tool.so /opt/rocm/rocprofiler/tool/libtracer_tool.so

2.3.7.2 rocFFT unit tests - memory access fault

Known failure with some power-of-2 size transforms in 1D real FFTs. This issue has been fixed in master branch of public rocFFT repo: https://github.com/ROCmSoftwarePlatform/rocFFT

2.3.8 Closed source components

The ROCm platform relies on a few closed source components to provide functionality such as HSA image support. These components are only available through the ROCm repositories, and they will either be deprecated or become open source components in the future. These components are made available in the following packages:

- hsa-ext-rocr-dev

2.3.9 Getting ROCm source code

ROCm is built from open source software. As such, it is possible to make modifications to the various components of ROCm by downloading the source code, making modifications to it, and rebuilding the components. The source code for ROCm components can be cloned from each of the GitHub repositories using git. In order to make it easier to download the correct versions of each of these tools, this ROCm repository contains a repo manifest file, default.xml. Interested users can thus use this manifest file to download the source code for all of the ROCm software.

2.3.9.1 Installing repo

Google's repo tool allows you to manage multiple git repositories simultaneously. You can install it by executing the following example commands:

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

Note that you can choose a different folder to install repo into if you desire. `~/bin/` is simply used as an example.

2.3.9.2 Downloading the ROCm source code

The following example shows how to use the `repo` binary downloaded above to download all of the ROCm source code. If you chose a directory other than `~/bin/` to install `repo`, you should use that directory below.

```
mkdir -p ~/ROCm/
cd ~/ROCm/
~/bin/repo init -u https://github.com/RadeonOpenCompute/ROCm.git -b roc-2.7.0
repo sync
```

This will cause repo to download all of the open source code associated with this ROCm release. You may want to ensure that you have ssh-keys configured on your machine for your GitHub ID.

2.3.9.3 Building the ROCm source code

Each ROCm component repository contains directions for building that component. As such, you should go to the repository you are interested in building to find how to build it.

That said, AMD also offers a project that demonstrates how to download, build, package, and install ROCm software on various distributions. The scripts here may be useful for anyone looking to build ROCm components.

2.3.10 Deprecation Notice

2.3.10.1 HCC

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 end of the year, 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.

2.3.10.2 hipThrust

hip-thrust has been removed in ROCm2.7.

2.3.11 Final notes

- OpenCL Runtime and Compiler will be submitted to the Khronos Group for conformance testing prior to its final release.

2.4 Programming Guide

2.4.1 ROCm Languages

2.4.1.1 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.

2.4.1.2 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 multicore 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 device 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](https://github.com/RadeonOpenCompute/HCC-Native-GCN-ISA) compilation

2.4.1.3 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.

2.4.1.4 HIP: Heterogeneous-Computing Interface for Portability

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 GPU 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.

2.4.1.5 When to Use HIP

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.

2.4.1.6 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.

2.4.1.7 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).

2.4.1.8 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.

2.4.1.9 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

2.4.1.10 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.

2.4.1.11 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!

2.4.1.12 Table Comparing Syntax for Different Compute APIs

Term	CUDA	HIP	HC	C++AMP	OpenCL
Device	int deviceId	int deviceId	hc::accelerator	concurrency:: accelerator	cl_device
Queue	cudaStream_t	hipStream_t	hc:: accelerator_view	concurrency:: accelerator_view	cl_command_queue
Event	cudaEvent_t	hipEvent_t	hc:: completion_future	concurrency:: completion_future	cl_event
Memory	void *	void *	void *; hc::array; hc::array_view	concurrency::array; concurrency::array_view	cl_mem
	grid block thread warp	grid block thread warp	extent tile thread wavefront	extent tile thread N/A	NDRange work-group work-item sub-group
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	hipBlockDim_x	t_ext.tile_dim[0]	t_idx.tile_dim0	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_** (default)	__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	<<< >>>	hipLaunchKernel	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_idx.barrier()	barrier(CLK_LOCAL_MEMFENC
Atomic Builtins	atomicAdd	atomicAdd	hc::atomic_fetch_add	concurrency:: atomic_fetch_add	atomic_add
Precise Math	cos(f)	cos(f)	hc:: precise_math::cos(f)	concurrency:: precise_math::cos(f)	cos(f)
Fast Math	__cos(f)	__cos(f)	hc::fast_math::cos(f)	concurrency:: fast_math::cos(f)	native_cos(f)

2.4.1.13 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.**

2.4.2 HCC Programming Guide

2.4.2.1 HCC: Heterogeneous Compute Compiler

HCC is an Open Source, Optimizing C++ Compiler for Heterogeneous Compute

HCC supports heterogeneous offload to AMD APUs and discrete GPUs via HSA enabled runtimes and drivers. It is an ISO compliant C++ 11/14 compiler. It is based on Clang, the LLVM Compiler Infrastructure and the “libc++” C++ standard library.

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.

2.4.2.2 Accelerator Modes Supported

HC (Heterogeneous Compute) C++ API

Inspired by C++ AMP and C++17, this is the default C++ compute API for the HCC compiler. HC has some important differences from C++ AMP including removing the “restrict” keyword, supporting additional data types in kernels, providing more control over synchronization and data movement, and providing pointer-based memory allocation. It is designed to expose cutting edge compute capabilities on Boltzmann and HSA devices to developers while offering the productivity and usability of C++.

HIP

HIP provides a set of tools and API for converting CUDA applications into a portable C++ API. An application using the HIP API could be compiled by hcc to target AMD GPUs. Please refer to HIP’s repository for more information.

C++ AMP

NOTE The supported for C++AMP is being deprecated. The ROCm 1.9 release is the last release of HCC supporting C++AMP.

Microsoft C++ AMP is a C++ accelerator API with support for GPU offload. This mode is compatible with Version 1.2 of the C++ AMP specification.

C++ Parallel STL

HCC provides an initial implementation of the parallel algorithms described in the ISO C++ Extensions for Parallelism, which enables parallel acceleration for certain STL algorithms.

2.4.2.3 Platform Requirements

Accelerated applications could be run on Radeon discrete GPUs from the Fiji family (AMD R9 Nano, R9 Fury, R9 Fury X, FirePro S9300 x2, Polaris 10, Polaris 11) paired with an Intel Haswell CPU or newer. HCC would work with AMD HSA APUs (Kaveri, Carrizo); however, they are not our main support platform and some of the more advanced compute capabilities may not be available on the APUs.

HCC currently only works on Linux and with the open source ROCK kernel driver and the ROCR runtime (see Installation for details). It will not work with the closed source AMD graphics driver.

2.4.2.4 Compiler Backends

This backend compiles GPU kernels into native GCN ISA, which can be directly executed on the GPU hardware. It's being actively developed by the Radeon Technology Group in LLVM.

2.4.2.5 Installation

Prerequisites

Before continuing with the installation, please make sure any previously installed hcc compiler has been removed from on your system. Install ROCm and make sure it works correctly.

2.4.2.6 Ubuntu

Ubuntu 14.04

Support for 14.04 has been deprecated.

Ubuntu 16.04

Follow the instruction here to setup the ROCm apt repository and install the rocm or the rocm-dev meta-package.

Fedora 24

Follow the instruction here to setup the ROCm apt repository and install the rocm or the rocm-dev meta-package.

RHEL 7.4/CentOS 7

Follow the instruction here to setup the ROCm apt repository and install the rocm or the rocm-dev meta-package for RHEL/CentOS. Currently, HCC support for RHEL 7.4 and CentOS 7 is experimental and the compiler has to be built from source. Note: CentOS 7 cmake is outdated, will need to use alternate cmake3.

openSUSE Leap 42.3

Currently, HCC support for openSUSE is experimental and the compiler has to be built from source.

2.4.2.7 Download HCC

> The project now employs git submodules to manage external components it depends upon. It it advised to add –recursive when you clone the project so all submodules are fetched automatically.

For example

```
# automatically fetches all submodules
git clone --recursive -b clang_tot_upgrade https://github.com/RadeonOpenCompute/hcc.
↪git
```

2.4.2.8 Build HCC from source

First, install the build dependencies

```
# Ubuntu 14.04
sudo apt-get install git cmake make g++  g++-multilib gcc-multilib libc++-dev libc++1␣
↪libc++abi-dev libc++abi1 python findutils libelf1 libpci3 file debianutils␣
↪libunwind8-dev hsa-rocr-dev hsa-ext-rocr-dev hsakmt-roct-dev pkg-config rocm-utils
```

```
# Ubuntu 16.04
sudo apt-get install git cmake make g++  g++-multilib gcc-multilib python findutils␣
↪libelf1 libpci3 file debianutils libunwind- dev hsa-rocr-dev hsa-ext-rocr-dev␣
↪hsakmt-roct-dev pkg-config rocm-utils
```

```
# Fedora 23/24
sudo dnf install git cmake make gcc-c++ python findutils elfutils-libelf pciutils-
↪libs file pth rpm-build libunwind-devel hsa- rocr- dev hsa-ext-rocr-dev hsakmt-roct-
↪dev pkgconfig rocm-utils
```

Clone the HCC source tree

```
# automatically fetches all submodules
git clone --recursive -b clang_tot_upgrade https://github.com/RadeonOpenCompute/hcc.
↪git
```

Create a build directory and run cmake in that directory to configure the build

```
mkdir build;
cd build;
cmake ../hcc
```

Compile HCC

```
make -j
```

Run the unit tests

```
make test
```

Create an installer package (DEB or RPM file)

```
make package
```

To configure and build HCC from source, use the following steps

```
mkdir -p build; cd build
# NUM_BUILD_THREADS is optional
# set the number to your CPU core numbers time 2 is recommended
# in this example we set it to 96
  cmake -DNUM_BUILD_THREADS=96 \
  -DCMAKE_BUILD_TYPE=Release \
```

(continues on next page)

(continued from previous page)

```
..
make
```

To install it, use the following steps

```
sudo make install
```

2.4.2.9 Use HCC

For C++AMP source codes

```
hcc `clamp-config --cxxflags --ldflags` foo.cpp
```

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

For HC source codes

```
hcc `hcc-config --cxxflags --ldflags` foo.cpp
```

In case you build HCC from source and want to use the compiled binaries directly in the build directory:

For C++AMP source codes

```
# notice the --build flag
bin/hcc `bin/clamp-config --build --cxxflags --ldflags` foo.cpp
```

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

For HC source codes

```
# notice the --build flag
bin/hcc `bin/hcc-config --build --cxxflags --ldflags` foo.cpp
```

Compiling for Different GPU Architectures

By default, HCC will auto-detect all the GPU's local to the compiling machine and set the correct GPU architectures. Users could use the –amdgpu-target=<GCN Version> option to compile for a specific architecture and to disable the auto-detection. The following table shows the different versions currently supported by HCC.

There exists an environment variable HCC_AMDGPU_TARGET to override the default GPU architecture globally for HCC; however, the usage of this environment variable is NOT recommended as it is unsupported and it will be deprecated in a future release.

GCN Version	GPU/APU Family	Examples of Radeon GPU
gfx701	GFX7	FirePro W8100, FirePro W9100, Radeon R9 290, Radeon R9 390
gfx801	Carrizo APU	FX-8800P
gfx803	GFX8	R9 Fury, R9 Fury X, R9 Nano, FirePro S9300 x2, Radeon RX 480, Radeon RX 470, Radeon RX 460
gfx900	GFX9	Vega10

2.4.2.10 Multiple ISA

HCC now supports having multiple GCN ISAs in one executable file. You can do it in different ways: **use :: –amdgpu-target= command line option** It's possible to specify multiple –amdgpu-target= option.

Example

```
# ISA for Hawaii(gfx701), Carrizo(gfx801), Tonga(gfx802) and Fiji(gfx803) would
# be produced
hcc `hcc-config --cxxflags --ldflags` \
   --amdgpu-target=gfx701 \
   --amdgpu-target=gfx801 \
   --amdgpu-target=gfx802 \
   --amdgpu-target=gfx803 \
   foo.cpp
```

use :: HCC_AMDGPU_TARGET env var

Use, to delimit each AMDGPU target in HCC. Example

```
export HCC_AMDGPU_TARGET=gfx701,gfx801,gfx802,gfx803
# ISA for Hawaii(gfx701), Carrizo(gfx801), Tonga(gfx802) and Fiji(gfx803) would
# be produced
hcc `hcc-config --cxxflags --ldflags` foo.cpp
```

configure HCC using the CMake HSA_AMDGPU_GPU_TARGET variable

If you build HCC from source, it's possible to configure it to automatically produce multiple ISAs via :: HSA_AMDGPU_GPU_TARGET CMake variable. Use ; to delimit each AMDGPU target. Example

```
# ISA for Hawaii(gfx701), Carrizo(gfx801), Tonga(gfx802) and Fiji(gfx803) would
# be produced by default
cmake \
   -DCMAKE_BUILD_TYPE=Release \
   -DROCM_DEVICE_LIB_DIR=~hcc/ROCm-Device-Libs/build/dist/lib \
   -DHSA_AMDGPU_GPU_TARGET="gfx701;gfx801;gfx802;gfx803" \
   ../hcc
```

2.4.2.11 CodeXL Activity Logger

To enable the CodeXL Activity Logger, use the USE_CODEXL_ACTIVITY_LOGGER environment variable.

Configure the build in the following way

```
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DHSA_AMDGPU_GPU_TARGET=<AMD GPU ISA version string> \
  -DROCM_DEVICE_LIB_DIR=<location of the ROCm-Device-Libs bitcode> \
  -DUSE_CODEXL_ACTIVITY_LOGGER=1 \
  <ToT HCC checkout directory>
```

In your application compiled using hcc, include the CodeXL Activiy Logger header

```
#include <CXLActivityLogger.h>
```

For information about the usage of the Activity Logger for profiling, please refer to documentation

2.4.3 HC Best Practices

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.

C++AMP	HC
`Concurrency::accelerator`	`hc::accelerator`
`Concurrency::accelerator_view`	`hc::accelerator_view`
`Concurrency::extent`	`hc::extent`
`Concurrency::index`	`hc::index`
`Concurrency::completion_future`	`hc::completion_future`
`Concurrency::array`	`hc::array`
`Concurrency::array_view`	`hc::array_view`

2.4.3.1 HCC built-in macros

Built-in macros:

Macro	Meaning
`__HCC__`	always be 1
`__hcc_major__`	major version number of HCC
`__hcc_minor__`	minor version number of HCC
`__hcc_patchlevel__`	patchlevel of HCC
`__hcc_version__`	combined string of `__hcc_major__`, `__hcc_minor__`, `__hcc_patchlevel__`

The rule for `__hcc_patchlevel__` is: yyWW-(HCC driver git commit #)-(HCC clang git commit #)

- yy stands for the last 2 digits of the year
- WW stands for the week number of the year

Macros for language modes in use:

Macro	Meaning
`__KALMAR_AMP__`	1 in case in C++ AMP mode (-std=c++amp; **Removed from ROCm 2.0 onwards**)
`__KALMAR_HC__`	1 in case in HC mode (-hc)

Compilation mode: HCC is a single-source compiler where kernel codes and host codes can reside in the same file. Internally HCC would trigger 2 compilation iterations, and the following macros can be used by user programs to determine which mode the compiler is in.

Macro	Meaning
`__KALMAR_ACCELERATOR__`	not 0 in case the compiler runs in kernel code compilation mode
`__KALMAR_CPU__`	not 0 in case the compiler runs in host code compilation mode

2.4.3.2 HC-specific features

- 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

2.4.3.3 Differences between HC API and C++ AMP

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); howevever, 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 asynchronous 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.

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

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

```
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 `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:

> 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:
>
> > tiled_extent<8,8> t_ex(ex)

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

> > HC supports both static and dynamic tile size:
>
> 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

> tx = test_x ? tx_a : tx_b;
>
> ty = test_y ? ty_a : ty_b;

> tiled_extent<2> t_ex(ex, tx, ty);
>
> 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

> 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: .. code:: sh

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

2.4.3.4 HCC Profile Mode

HCC supports low-overhead profiler to trace or summarize command timestamp information to stderr for any HCC or HIP program. Tho profiler messages are interleaved with the trace output from the application - which is handy to identify the region-of-interest and can complement deeper analysis with the CodeXL GUI Additionally, the hcc profiler requires only console mode access and can be used on machine where graphics are not available or are hard to access.

Some other useful features:

- Calculates the actual bandwidth for memory transfers
- Identifies PeerToPeer memory copies
- Shows start / stop timestamps for each command (if requested)
- Shows barrier commands and the time they spent waiting to resolve (if requested)

Enable and configure

HCC_PROFILE=1 shows a summary of kernel and data commands when hcc exits (under development).
HCC_PROFILE=2 enables a profile message after each command (kernel or data movement) completes.

Additionally, the HCC_PROFILE_VERBOSE variable controls the information shown in the profile log. This is a bit-vector:
0x2 : Show start and stop timestamps for each command.
0x4 : Show the device.queue.cmdseqnum for each command.
0x8 : Show the short CPU TID for each command (not supported).
0x10 : Show logs for barrier commands.

Sample Output

2.4.3.4.1 Kernel Commands

This shows the simplest trace output for kernel commands with no additional verbosity flags

```
$ HCC_PROFILE=2 ./my-hcc-app ...
profile:  kernel;            Im2Col;   17.8 us;
profile:  kernel;  tg_betac_alphaab;   32.6 us;
profile:  kernel;     MIOpenConvUni;  125.4 us;
```

```
PROFILE:  TYPE;    KERNEL_NAME     ;  DURATION;
```

This example shows profiled kernel commands with full verbose output

```
$ HCC_PROFILE=2 HCC_PROFILE_VERBOSE=0xf ./my-hcc-app ...
profile:  kernel;            Im2Col;   17.8 us;  94859076277181; 94859076294941; #0.3.
 ↪1;
profile:  kernel;  tg_betac_alphaab;   32.6 us;  94859537593679; 94859537626319; #0.3.
 ↪2;
profile:  kernel;     MIOpenConvUni;  125.4 us;  94860077852212; 94860077977651; #0.3.
 ↪3;
```

```
PROFILE:  TYPE;    KERNEL_NAME     ;  DURATION;  START         ; STOP          ; ID
```

- PROFILE: always “profile:” to distinguish it from other output.
- TYPE: the command type : kernel, copy, copyslo, or barrier. The examples and descriptions in this section are all kernel commands.
- KERNEL_NAME: the (short) kernel name.
- DURATION: command duration measured in us. This is measured using the GPU timestamps and represents the command execution on the accelerator device.
- START: command start time in ns. (if HCC_PROFILE_VERBOSE & 0x2)
- STOP: command stop time in ns. (if HCC_PROFILE_VERBOSE & 0x2)
- ID: command id in device.queue.cmd format. (if HCC_PROFILE_VERBOSE & 0x4). The cmdsequm is a unique monotonically increasing number per-queue, so the triple of device.queue.cmdseqnum uniquely identifies the command during the process execution.

2.4.3.4.2 Memory Copy Commands

This example shows memory copy commands with full verbose output:

```
profile: copyslo; HostToDevice_sync_slow;   909.2 us; 94858703102; 94858704012; #0.0.
 ↪0; 2359296 bytes;  2.2 MB;   2.5 GB/s;
profile:    copy; DeviceToHost_sync_fast;   117.0 us; 94858726408; 94858726525; #0.0.
 ↪0; 1228800 bytes;  1.2 MB;  10.0 GB/s;
profile:    copy; DeviceToHost_sync_fast;     9.0 us; 94858726668; 94858726677; #0.0.
 ↪0; 400 bytes;      0.0 MB;   0.0 GB/s;
profile:    copy; HostToDevice_sync_fast;    15.2 us; 94858727639; 94858727654; #0.0.
 ↪0; 9600 bytes;     0.0 MB;   0.6 GB/s;
profile:    copy; HostToDevice_async_fast;  131.5 us; 94858729198; 94858729330; #0.6.
 ↪1; 1228800 bytes;  1.2 MB;   8.9 GB/s;
PROFILE:  TYPE;    COPY_NAME             ;  DURATION;       START;       STOP;  ID   ␣
 ↪; SIZE_BYTES;     SIZE_MB;  BANDWIDTH;
```

- PROFILE: always “profile:” to distinguish it from other output.
- TYPE: the command type : kernel, copy, copyslo,or barrier. The examples and descriptions in this section are all copy or copyslo commands.
- **COPY_NAME has 3 parts:**
  - Copy kind: HostToDevice, HostToHost, DeviceToHost, DeviceToDevice, or PeerToPeer. DeviceToDevice indicates the copy occurs on a single device while PeerToPeer indicates a copy between devices.
  - Sync or Async. Synchronous copies indicate the host waits for the completion for the copy. Asynchronous copies are launched by the host without waiting for the copy to complete.
  - Fast or Slow. Fast copies use the GPUs optimized copy routines from the hsa_amd_memory_copy routine. Slow copies typically involve unpinned host memory and can’t take the fast path.
  - For example *HostToDevice_async_fast*.
- DURATION: command duration measured in us. This is measured using the GPU timestamps and represents the command execution on the accelerator device.
- START: command start time in ns. (if HCC_PROFILE_VERBOSE & 0x2)
- STOP: command stop time in ns. (if HCC_PROFILE_VERBOSE & 0x2)
- ID: command id in device.queue.cmd format. (if HCC_PROFILE_VERBOSE & 0x4). The cmdsequm is a unique mononotically increasing number per-queue, so the triple of device.queue.cmdseqnum uniquely identifies the command during the process execution.
- SIZE_BYTES: the size of the transfer, measured in bytes.
- SIZE_MB: the size of the transfer, measured in megabytes.
- BANDWIDTH: the bandwidth of the transfer, measured in GB/s.

2.4.3.4.3 Barrier Commands

Barrier commands are only enabled if HCC_PROFILE_VERBOSE 0x10

An example barrier command with full vebosity

```
profile: barrier; deps:0_acq:none_rel:sys;  5.3 us;   94858731419410; 94858731424690;
↪# 0.0.2;
PROFILE:  TYPE;   BARRIER_NAME           ;  DURATION; START          ; STOP           ;␣
↪ID    ;
```

- PROFILE: always “profile:” to distinguish it from other output.
- TYPE: the command type: either kernel, copy, copyslo, or barrier. The examples and descriptions in this section are all copy commands. Copy indicates that the runtime used a call to the fast hsa memory copy routine while copyslo indicates that the copy was implemented with staging buffers or another less optimal path. copy computes the commands using device-side timestamps while copyslo computes the bandwidth based on host timestamps.
- **BARRIER_NAME has 3 parts:**
  - **deps:#** - the number of input dependencies into the barrier packet.
  - **acq:** - the acquire fence for the barrier. May be none, acc(accelerator or agent), sys(system). See HSA AQL spec for additional information.

  - **rel:** - the release fence for the barrier. May be none, acc(accelerator or agent), sys(system). See HSA AQL spec for additional information.

- DURATION: command duration measured in us. This is measured using the GPU timestamps from the time the barrier reaches the head of the queue to when it executes. Thus this includes the time to wait for all input dependencies, plus the previous command to complete, plus any fence operations performed by the barrier.
- START: command start time in ns. (if HCC_PROFILE_VERBOSE & 0x2)
- STOP: command stop time in ns. (if HCC_PROFILE_VERBOSE & 0x2)
- ID: the command id in device.queue.cmd format. (if HCC_PROFILE_VERBOSE & 0x4). The cmdsequm is a unique mononotically increasing number per-queue, so the triple of device.queue.cmdseqnum uniquely identifies the command during the process execution.

2.4.3.4.4 Overhead

The hcc profiler does not add any additional synchronization between commands or queues. Profile information is recorded when a command is deleted. The profile mode will allocate a signal for each command to record the timestamp information. This can add 1-2 us to the overall program execution for command which do not already use a completion signal. However, the command duration (start-stop) is still accurate. Trace mode will generate strings to stderr which will likely impact the overall application exection time. However, the GPU duration and timestamps are still valid. Summary mode accumulates statistics into an array and should have little impact on application execution time.

2.4.3.4.5 Additional Details and tips

- Commands are logged in the order they are removed from the internal HCC command tracker. Typically this is the same order that commands are dispatched, though sometimes these may diverge. For example, commands from different devices,queues, or cpu threads may be interleaved on the hcc trace display to stderr. If a single view in timeline order is required, enable and sort by the profiler START timestamps (HCC_PROFILE_VERBOSE=0x2)
- If the application keeps a reference to a completion_future, then the command timestamp may be reported significantly after it occurs.
- HCC_PROFILE has an (untested) feature to write to a log file.

2.4.3.5 API documentation

API reference of HCC

2.4.4 HIP Programing Guide

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

2.4.5 HIP Best Practices

- HIP-porting-guide
- HIP-terminology
- hip_profiling
- HIP_Debugging
- Kernel_language
- HIP-Terms
- HIP-bug
- hipporting-driver-api
- CUDAAPIHIP
- CUDAAPIHIPTEXTURE
- HIP-FAQ
- HIP-Term2

2.4.6 OpenCL Programing Guide

- Opencl-Programming-Guide

2.4.7 OpenCL Best Practices

- Optimization-Opencl

2.5 ROCm GPU Tuning Guides

2.5.1 GFX7 Tuning Guide

2.5.2 GFX8 Tuning Guide

2.5.3 Vega Tuning Guide

2.6 GCN ISA Manuals

2.6.1 GCN 1.1

ISA Manual for Hawaii pdf

2.6.2 GCN 2.0

ISA Manual for Fiji and Polaris pdf

2.6.3 Vega

- testdocbook

2.6.4 Inline GCN ISA Assembly Guide

2.6.4.1 The Art of AMDGCN Assembly: How to Bend the Machine to Your Will

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.

2.6.4.2 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:

```
__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] ];
}
```

2.6.4.3 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:

```
/*
* 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:

.amd_kernel_code_t
enable_sgpr_kernarg_segment_ptr = 1
is_ptr64 = 1
compute_pgm_rsrc1_vgprs = 1
compute_pgm_rsrc1_sgprs = 0
compute_pgm_rsrc2_user_sgpr = 2
kernarg_segment_byte_size = 24
wavefront_sgpr_count = 8
workitem_vgpr_count = 5
.end_amd_kernel_code_t

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)
v_lshlrev_b32  v1, 2, v1
ds_bpermute_b32  v1, v1, v2
v_add_u32     v3, vcc, s4, v0
v_mov_b32     v2, s5
v_addc_u32    v4, vcc, v2, 0, vcc
s_waitcnt     lgkmcnt(0)
flat_store_dword  v[3:4], v1
s_endpgm
```

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. initial_state

2.6.4.4 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:

```
compute_pgm_rsrc1_vgprs = (workitem_vgpr_count-1)/4

compute_pgm_rsrc1_sgprs = (wavefront_sgpr_count-1)/8
```

Now consider the corresponding 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␣
↪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
flat_load_dword  v2, v[3:4] // load in[workitem_id] into v2
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)

(continued from previous page)

```
// i.e. v[3:4] = &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
```

2.6.4.5 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.

2.7 ROCm API References

2.7.1 ROCr System Runtime API

- ROCr-API

2.7.2 HCC Language Runtime API

- HCC-API

2.7.3 HIP Language Runtime API

- HIP-API

2.7.4 HIP Math API

- HIP-MATH

2.7.5 Math Library API's

- hcRNG
- clBLAS
- clSPARSE_api

2.7.6 Deep Learning API's

- MIOpen API
- MIOpenGEMM API

2.8 ROCm Tools

2.8.1 HCC

HCC is an Open Source, Optimizing C++ Compiler for Heterogeneous Compute

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 C++ AMP, HC, C++ 17 ParallelSTL, or OpenMP, and transforms it into the AMD GCN ISA.

The project is based on LLVM+CLANG. For more information, please visit the HCCwiki

2.8.1.1 Download HCC

The project now employs git submodules to manage external components it depends upon. It it advised to add –recursive when you clone the project so all submodules are fetched automatically.

For example:

```
# automatically fetches all submodules
git clone --recursive -b clang_tot_upgrade https://github.com/RadeonOpenCompute/hcc.
↪git
```

For more information about git submodules, please refer to git documentation.

2.8.1.2 Build HCC from source

To configure and build HCC from source, use the following steps:

```
mkdir -p build; cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

To install it, use the following steps:

```
sudo make install
```

2.8.1.3 Use HCC

For C++AMP source codes:

```
hcc `clamp-config --cxxflags --ldflags` foo.cpp
```

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

For HC source codes:

```
hcc `hcc-config --cxxflags --ldflags` foo.cpp
```

In case you build HCC from source and want to use the compiled binaries directly in the build directory:

For C++AMP source codes:

```
# notice the --build flag
bin/hcc `bin/clamp-config --build --cxxflags --ldflags` foo.cpp
```

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

For HC source codes:

```
# notice the --build flag
bin/hcc `bin/hcc-config --build --cxxflags --ldflags` foo.cpp
```

2.8.1.4 Multiple ISA

HCC now supports having multiple GCN ISAs in one executable file. You can do it in different ways: **use ‘‘–amdgpu-target=‘‘ command line option**

It’s possible to specify multiple ‘‘ –amdgpu-target= ‘‘ option.

Example:

```
# ISA for Hawaii(gfx701), Carrizo(gfx801), Tonga(gfx802) and Fiji(gfx803) would
# be produced
 hcc `hcc-config --cxxflags --ldflags` \
   --amdgpu-target=gfx701 \
   --amdgpu-target=gfx801 \
   --amdgpu-target=gfx802 \
   --amdgpu-target=gfx803 \
   foo.cpp
```

use ‘‘HCC_AMDGPU_TARGET‘‘ env var

use `,` to delimit each AMDGPU target in HCC. Example:

```
export HCC_AMDGPU_TARGET=gfx701,gfx801,gfx802,gfx803
# ISA for Hawaii(gfx701), Carrizo(gfx801), Tonga(gfx802) and Fiji(gfx803) would
# be produced
hcc `hcc-config --cxxflags --ldflags` foo.cpp
```

configure HCC use CMake ‘‘HSA_AMDGPU_GPU_TARGET‘‘ variable

If you build HCC from source, it’s possible to configure it to automatically produce multiple ISAs via *HSA_AMDGPU_GPU_TARGET* CMake variable.

Use `;` to delimit each AMDGPU target. Example:

```
# ISA for Hawaii(gfx701), Carrizo(gfx801), Tonga(gfx802) and Fiji(gfx803) would
# be produced by default
cmake \
   -DCMAKE_BUILD_TYPE=Release \
   -DROCM_DEVICE_LIB_DIR=~hcc/ROCm-Device-Libs/build/dist/lib \
   -DHSA_AMDGPU_GPU_TARGET="gfx701;gfx801;gfx802;gfx803" \
   ../hcc
```

2.8.1.5 CodeXL Activity Logger

To enable the CodeXL Activity Logger, use the USE_CODEXL_ACTIVITY_LOGGER environment variable.

Configure the build in the following way:

```
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DHSA_AMDGPU_GPU_TARGET=<AMD GPU ISA version string> \
  -DROCM_DEVICE_LIB_DIR=<location of the ROCm-Device-Libs bitcode> \
  -DUSE_CODEXL_ACTIVITY_LOGGER=1 \
  <ToT HCC checkout directory>
```

In your application compiled using hcc, include the CodeXL Activiy Logger header:

```
#include <CXLActivityLogger.h>
```

For information about the usage of the Activity Logger for profiling, please refer to its documentation.

2.8.1.6 HCC with ThinLTO Linking

To enable the ThinLTO link time, use the KMTHINLTO environment variable.

Set up your environment in the following way:

```
export KMTHINLTO=1
```

ThinLTO Phase 1 - Implemented

For applications compiled using hcc, ThinLTO could significantly improve link-time performance. This implementation will maintain kernels in their .bc file format, create module-summaries for each, perform llvm-lto's cross-module function importing and then perform clamp-device (which uses opt and llc tools) on each of the kernel files. These files are linked with lld into one .hsaco per target specified.

ThinLTO Phase 2 - Under development This ThinLTO implementation which will use llvm-lto LLVM tool to replace clamp-device bash script. It adds an optllc option into ThinLTOGenerator, which will perform in-program opt and codegen in parallel.

2.8.2 GCN Assembler and Disassembler

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.

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:

```
__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] ];
}
```

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:

```
/*
* 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;
```

(continues on next page)

(continued from previous page)

```
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:

.amd_kernel_code_t
enable_sgpr_kernarg_segment_ptr = 1
is_ptr64 = 1
compute_pgm_rsrc1_vgprs = 1
compute_pgm_rsrc1_sgprs = 0
compute_pgm_rsrc2_user_sgpr = 2
kernarg_segment_byte_size = 24
wavefront_sgpr_count = 8
workitem_vgpr_count = 5
.end_amd_kernel_code_t

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
```

(continues on next page)

(continued from previous page)

```
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)
v_lshlrev_b32  v1, 2, v1
ds_bpermute_b32  v1, v1, v2
v_add_u32     v3, vcc, s4, v0
v_mov_b32     v2, s5
v_addc_u32    v4, vcc, v2, 0, vcc
s_waitcnt     lgkmcnt(0)
flat_store_dword  v[3:4], v1
s_endpgm
```

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. initial_state

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:

```
compute_pgm_rsrc1_vgprs = (workitem_vgpr_count-1)/4

compute_pgm_rsrc1_sgprs = (wavefront_sgpr_count-1)/8
```

Now consider the corresponding 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 s[2:3] from kernarg
v_lshlrev_b32  v0, 2, v0              // v0 *= 4;
s_waitcnt     lgkmcnt(0)              // wait for memory reads to finish
```

(continues on next page)

(continued from previous page)

```
// 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
flat_load_dword  v2, v[3:4] // load in[workitem_id] into v2
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
// i.e. v[3:4] = &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
```

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.

2.8.3 GCN Assembler Tools

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 convertable 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 lld as linker and produces AMD Code Object (v2).

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.

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.

Refer to examples/api/assemble.

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:

```
llvm-mc -arch=amdgcn -mcpu=fiji -filetype=obj -o asm.o asm.s
andphdrs asm.o asm.co
```

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]
```

- LLVM Use Guide for AMDGPU Back-End
- **AMD ISA Documents**
    - AMD GCN3 Instruction Set Architecture (2016)
    - AMD_Southern_Islands_Instruction_Set_Architecture

2.8.4 ROC Profiler

ROC profiler library. Profiling with perf-counters and derived metrics. Library supports GFX8/GFX9.

HW specific low-level performance analysis interface for profiling of GPU compute applications. The profiling includes HW performance counters with complex performance metrics and HW traces.

Profiling tool 'rocprof':

- 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
- Kernel execution is serialized

  - HSA API/activity stats and tracing
  - Output results in CSV and JSON chrome tracing formats

2.8.4.1 Download

To clone ROC Profiler from GitHub use the folowing command:

```
git clone https://github.com/ROCm-Developer-Tools/rocprofiler
```

The library source tree:

- **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 controll
  - util - Test utils
  - simple_convolution - Simple convolution test kernel

2.8.4.2 Build

Build environment:

```
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 configure, build, install to /opt/rocm/rocprofiler:

```
mkdir -p build
cd build
export CMAKE_PREFIX_PATH=/opt/rocm
cmake -DCMAKE_INSTALL_PREFIX=/opt/rocm ..
make
sudo make install
```

To test the built library:

```
cd build
./run.sh
```

To enable error messages logging to '/tmp/rocprofiler_log.txt':

```
export ROCPROFILER_LOG=1
```

To enable verbose tracing:

```
export ROCPROFILER_TRACE=1
```

2.8.4.3 Profiling Tool 'rocprof' Usage

The following shows the command-line usage of the 'rocprof' tool:

```
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

-i <.txt|.xml file> - input file
    Input file .txt format, automatically rerun application for every pmc line:

      # Perf counters group 1
      pmc : Wavefronts VALUInsts SALUInsts SFetchInsts FlatVMemInsts LDSInsts FlatLDSInsts GDSInsts VALUUtilization FetchSize
      # Perf counters group 2
      pmc : 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>
```

(continues on next page)

(continued from previous page)

```
-o <output file> - output CSV file [<input file base>.csv]
-d <data directory> - directory where profiler store profiling data including thread␣
↪treaces [/tmp]
    The data directory is renoving autonatically 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 disoatches timestamps, dispatch/
↪begin/end/complete [off]
--ctx-limit <max number> - maximum number of outstanding contexts [0 - unlimited]
--heartbeat <rate sec> - to print progress heartbeats [0 - disabled]

--hsa-trace - to trace HSA, generates API execution stats and JSON file viewable in␣
↪chrome tracing
  Requires to set three options '--hsa-trace --stats --timestamp on'
  Will be simplified to just one option in the next release
  Generated files: <output name>.stats.csv <output name>.hsa_stats.txt <output name>.
↪json

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'.
An example of 'rpl_rc.xml':
  <defaults
    basenames=off
    timestamp=off
    ctx-limit=0
    heartbeat=0
  ></defaults>
```

2.8.5 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 (`ctrl c`) or SIGTERM (`kill -15`) to print wavefront state of aborted GPU dispatches.
- It is enabled on Vega10 GPUs on ROCm2.7.

To use the ROCr Debug Agent set the following environment variable:

```
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:

```
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 llvm-objdump tool. Ensure that a version that supports AMD GCN GPUs is on your `$PATH`. For example, for ROCm 2.7:

```
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 `__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: 0x00C00000    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
```

(continues on next page)

(continued from previous page)

```
    .
    .
    .

Lane 0x3C
     v0: 0x0000001F    v1: 0x18DB4400    v2: 0x18DB4400    v3: 0x00000000
     v4: 0x00000000    v5: 0x00000000    v6: 0x00700000    v7: 0x00800000
Lane 0x3D
     v0: 0x00000020    v1: 0x18DB4400    v2: 0x18DB4400    v3: 0x00000000
     v4: 0x00000000    v5: 0x00000000    v6: 0x00700000    v7: 0x00800000
Lane 0x3E
     v0: 0x00000021    v1: 0x18DB4400    v2: 0x18DB4400    v3: 0x00000000
     v4: 0x00000000    v5: 0x00000000    v6: 0x00700000    v7: 0x00800000
Lane 0x3F
     v0: 0x00000022    v1: 0x18DB4400    v2: 0x18DB4400    v3: 0x00000000
     v4: 0x00000000    v5: 0x00000000    v6: 0x00700000    v7: 0x00800000

Faulty Code Object:

/tmp/ROCm_Tmp_PID_5764/ROCm_Code_Object_0:      file format ELF64-amdgpu-hsacobj

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)
```

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:

```
export ROCM_DEBUG_ENABLE_LINUX_SIGNALS=0
unset ROCM_DEBUG_ENABLE_LINUX_SIGNALS
```

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 environent 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
```

2.8.6 ROCm-GDB

The ROCm-GDB is being revised to work with the ROCr Debug Agent to support debugging GPU kernels on Radeon Open Compute platforms (ROCm) and will be available in an upcoming release.

2.8.7 Radeon Compute Profiler

The Radeon Compute Profiler (RCP) is a performance analysis tool that gathers data from the API run-time and GPU for OpenCL™ and ROCm/HSA applications. This information can be used by developers to discover bottlenecks in the application and to find ways to optimize the application's performance.

Please see the RCP GitHub repository for more information.

2.8.8 ROC Tracer

ROC-tracer 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 asyncronous activity.

The API provides functionality for registering the runtimes API callbacks and asyncronous activity records pool support.

The library source tree:

- inc/roctracer.h - Library public API
- **src - Library sources**
  - core - Library API sources
  - util - Library utils sources
- **test - test suit**
  - MatrixTranspose - test based on HIP MatrixTranspose sample

2.8.8.1 Documentation

```
- API description: inc/roctracer.h
- Code example: test/MatrixTranspose/MatrixTranspose.cpp
```

2.8.8.2 To build and run test

```
 cd <your path>

- CLone development branches of roctracer and HIP/HCC:
 git clone -b amd-master https://github.com/ROCmSoftwarePlatform/roctracer.git
 git clone -b master https://github.com/ROCm-Developer-Tools/HIP.git
 git clone --recursive -b clang_tot_upgrade https://github.com/RadeonOpenCompute/hcc.
↪git

- Set environment:
 export HIP_PATH=<your path>/HIP
 export HCC_HOME=<your path>/hcc/build
 export CMAKE_PREFIX_PATH=/opt/rocm

- Build HCC:
 cd <your path>/hcc && mkdir build && cd build &&
 cmake -DUSE_PROF_API=1 -DPROF_API_HEADER_PATH=<your path>/roctracer/inc/ext .. &&␣
↪make -j <nproc>

- Build HIP:
```

(continues on next page)

(continued from previous page)

```
 cd <your path>/HIP && mkdir build && cd build &&
 cmake -DUSE_PROF_API=1 -DPROF_API_HEADER_PATH=<your path>/roctracer/inc/ext .. && make -j <nproc>
 ln -s <your path>/HIP/build <your path>/HIP/lib

- Build ROCtracer
 cd <your path>/roctracer && mkdir build && cd build && cmake -DCMAKE_INSTALL_PREFIX=/opt/rocm .. && make -j <nproc>

- To build and run test
 make mytest
 run.sh

- To install
 make install
 or
 make package && dpkg -i *.deb
```

2.8.9 CodeXL

CodeXL is a comprehensive tool suite that enables developers to harness the benefits of GPUs and APUs. It includes powerful GPU debugging, comprehensive GPU profiling, and static OpenCL™, OpenGL®, Vulkan® and DirectX® kernel/shader analysis capabilities, enhancing accessibility for software developers to enter the era of heterogeneous computing. CodeXL is available as a standalone user interface application for Windows® and Linux®.

Please see the CodeXL GitHub repository for more information.

2.8.10 GPUPerfAPI

The GPU Performance API (GPUPerfAPI, or GPA) is a powerful library, providing access to GPU Performance Counters. It can help analyze the performance and execution characteristics of applications using a Radeon™ GPU. This library is used by Radeon Compute Profiler and CodeXL as well as several third-party tools.

Please see the GPA GitHub repository for more information.

2.8.11 ROCm Binary Utilities

Documentation need to be updated.

2.8.12 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.

2.8.12.1 AMD OpenVX (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.

2.8.12.2 AMD OpenVX Extensions

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_loomsl: 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.

2.8.12.2.1 Features

- The code is highly optimized for both x86 CPU and OpenCL for GPU
- Supported hardware spans the range from low power embedded APUs (like the new G series) to laptop, desktop and workstation graphics
- Supports Windows, Linux, and OS X
- Includes a “graph optimizer” that looks at the entire processing pipeline and removes/replaces/merges functions to improve performance and minimize bandwidth at runtime
- Scripting support allows for rapid prototyping, without re-compiling at production performance levels.

2.8.12.2.2 Pre-requisites:

- CPU: SSE4.1 or above CPU, 64-bit.
- **GPU: Radeon Professional Graphics Cards or Vega Family of Products (16GB required for vx_loomsl and vx_nn librarie**
    - Windows: install the latest drivers and OpenCL SDK Download.
    - Linux: install ROCm.
- **OpenCV 3 (optional) download for RunVX**
    - Set OpenCV_DIR environment variable to OpenCV/build folder.

2.8.12.2.3 Build Instructions

Build this project to generate AMD OpenVX library and RunVX executable.

- Refer to openvx/include/VX for Khronos OpenVX standard header files.
- Refer to openvx/include/vx_ext_amd.h for vendor extensions in AMD OpenVX library.
- Refer to runvx/README.md for RunVX details.
- Refer to runcl/README.md for RunCL details.

2.8.12.2.4 Build using Visual Studio Professional 2013 on 64-bit Windows 10/8.1/7

- Install OpenCV 3 with contrib download for RunVX tool to support camera capture and image display (optional)
- OpenCV_DIR environment variable should point to OpenCV/build folder
- Use amdovx-core/amdovx.sln to build for x64 platform
- f AMD GPU (or OpenCL) is not available, set build flag ENABLE_OPENCL=0 in openvx/openvx.vcxproj and runvx/runvx.vcxproj.

2.8.12.2.5 Build using CMake

- Install CMake 2.8 or newer download.
- Install OpenCV 3 with contrib download for RunVX tool to support camera capture and image display (optional)
- OpenCV_DIR environment variable should point to OpenCV/build folder
- Install libssl-dev on linux (optional)
- Use CMake to configure and generate Makefile
- If AMD GPU (or OpenCL) is not available, use build flag -DCMAKE_DISABLE_FIND_PACKAGE_OpenCL=TRUE.

2.8.12.2.6 Radeon Loom Stitching Library (vx_loomsl)

Radeon Loom Stitching Library (beta preview) is a highly optimized library for 360 degree video stitching applications. This library consists of:

- Live Stitch API: stitching framework built on top of OpenVX kernels (see live_stitch_api.h for API)
- OpenVX module [vx_loomsl]: additional OpenVX kernels needed for 360 degree video stitching

The loom_shell command-line tool can be used to build your application quickly. It provides direct access to Live Stitch API by encapsulating the calls to enable rapid prototyping.

This software is provided under a MIT-style license, see the file COPYRIGHT.txt for details.

Features

- Real-time live 360 degree video stitching optimized for Radeon Pro Graphics
- Upto 31 cameras
- Upto 7680x3840 output resolution
- RGB and YUV 4:2:2 image formats
- Overlay other videos on top of stitched video

- Support for 3rd party LoomIO plug-ins for camera capture and stitched output
- Support PtGui project export/import for camera calibration

Live Stitch API: Simple Example

Let's consider a 360 rig that has 3 1080p cameras with Circular FishEye lenses. The below example demonstrates how to stitch images from these cameras into a 4K Equirectangular buffer.

```
#include "vx_loomsl/live_stitch_api.h"
  #include "utils/loom_shell/loom_shell_util.h"

  int main()
  {
    # define camera orientation and lens parameters
    camera_params cam1_par = { { 120,0,90,0,0,0},{176,1094,547,0,-37,ptgui_lens_
↪fisheye_circ,-0.1719,0.1539,1.0177} };
    camera_params cam2_par = { {   0,0,90,0,0,0},{176,1094,547,0,-37,ptgui_lens_
↪fisheye_circ,-0.1719,0.1539,1.0177} };
    camera_params cam3_par = { {-120,0,90,0,0,0},{176,1094,547,0,-37,ptgui_lens_
↪fisheye_circ,-0.1719,0.1539,1.0177} };

    # create a live stitch instance and initialize
    ls_context context;
    context = lsCreateContext();
    lsSetOutputConfig(context,VX_DF_IMAGE_RGB,3840,1920);
    lsSetCameraConfig(context,3,1,VX_DF_IMAGE_RGB,1920,1080*3);
    lsSetCameraParams(context, 0, &cam1_par);
    lsSetCameraParams(context, 1, &cam2_par);
    lsSetCameraParams(context, 2, &cam3_par);
    lsInitialize(context);

    # Get OpenCL context and create OpenCL buffers for input and output
    cl_context opencl_context;
    cl_mem buf[2];
    lsGetOpenCLContext(context,&opencl_context);
    createBuffer(opencl_context,3*1920*1080*3, &buf[0]);
    createBuffer(opencl_context,3*3840*1920  , &buf[1]);

    # load CAM00.bmp, CAM01.bmp, and CAM02.bmp (1920x1080 each) into buf[0]
    loadBufferFromMultipleImages(buf[0],"CAM%02d.bmp",3,1,VX_DF_IMAGE_RGB,1920,
↪1080*3);

    # set input and output buffers and stitch a frame
    lsSetCameraBuffer(context, &buf[0]);
    lsSetOutputBuffer(context, &buf[1]);
    lsScheduleFrame(context);
    lsWaitForCompletion(context);

    # save the stitched output into "output.bmp"
    saveBufferToImage(buf[1],"output.bmp",VX_DF_IMAGE_RGB,3840,1920);

    # release resources
    releaseBuffer(&buf[0]);
    releaseBuffer(&buf[1]);
    lsReleaseContext(&context);

    return 0;
  }
```

Live Stitch API: Real-time Live Stitch using LoomIO

This example makes use of a 3rd party LoomIO plug-ins for live camera capture and display.

```
#include "vx_loomsl/live_stitch_api.h"
  int main()
  {
      // create context, configure, and initialize
      ls_context context;
      context = lsCreateContext();
      lsSetOutputConfig(context, VX_DF_IMAGE_RGB, 3840, 1920);
      lsSetCameraConfig(context, 16, 1, VX_DF_IMAGE_RGB, 1920, 1080 * 16);
      lsImportConfiguration(context, "pts", "myrig.pts");
      lsSetCameraModule(context, "vx_loomio_bm", "com.amd.loomio_bm.capture", "30,0,0,
↪16");
      lsSetOutputModule(context, "vx_loomio_bm", "com.amd.loomio_bm.display", "30,0,0
↪");
      lsInitialize(context);

      // process live from camera until aborted by input capture plug-in
      for(;;) {
        vx_status status;
        status = lsScheduleFrame(context);
        if (status != VX_SUCCESS) break;
        status = lsWaitForCompletion(context);
        if (status != VX_SUCCESS) break;
      }

      // release the context
      lsReleaseContext(&context);

      return 0;
  }
```

2.8.12.2.7 OpenVX Neural Network Extension Library (vx_nn)

vx_nn is an OpenVX Neural Network extension module. This implementation supports only floating-point tensor datatype and does not support 8-bit and 16-bit fixed-point datatypes specified in the OpenVX specification.

List of supported tensor and neural network layers:

Layer name | Function|Kernel name | | ——|————|———— | |

Activation|vxActivationLayer|org.khronos.nn_extension.activation_layer | | Argmax|vxArgmaxLayerNode|com.amd.nn_extension.argma | | Batch Normalization|vxBatchNormalizationLayer|com.amd.nn_extension.batch_normalization_layer | | Concat|vxConcatLayer|com.amd.nn_extension.concat_layer | | Convolution|vxConvolutionLayer|org.khronos.nn_extension.convolution_laye | | Deconvolution|vxDeconvolutionLayer|org.khronos.nn_extension.deconvolution_layer | | Fully Connected|vxFullyConnectedLayer|org.khronos.nn_extension.fully_connected_layer | | Local Response Normalization|vxNormalizationLayer|org.khronos.nn_extension.normalization_layer | | Pooling|vxPoolingLayer|org.khronos.nn_extension.pooling_layer | | ROI Pooling|vxROIPoolingLayer|org.khronos.nn_extension.roi_pooling_layer | | Scale|vxScaleLayer|com.amd.nn_extension.scale_layer | | Slice|vxSliceLayer|com.amd.nn_extension.slice_layer | | Softmax|vxSoftmaxLayer|org.khronos.nn_extension.softmax_layer | | Tensor Add|vxTensorAddNode|org.khronos.openvx.tensor_add | | Tensor Convert Depth|vxTensorConvertDepthNode|org.khronos.openvx.tensor_convert_depth | | Tensor Convert from Image|vxConvertImageToTensorNode|com.amd.nn_extension.convert_image_to_tensor | | Tensor Convert to Image|vxConvertTensorToImageNode|com.amd.nn_extension.convert_tensor_to_image | | Tensor Multiply|vxTensorMultiplyNode|org.khronos.openvx.tensor_multiply | | Tensor Subtract|vxTensorSubtractNode|org.khronos.openvx.tensor_subtract | | Upsample Nearest Neighborhood|vxUpsampleNearestLayer|com.amd.nn_extension.upsample_nearest_layer |

Example 1: Convert an image to a tensor of type float32

Use the below GDF with RunVX.

```
import vx_nn
data input  = image:32,32,RGB2
data output = tensor:4,{32,32,3,1},VX_TYPE_FLOAT32,0
data a = scalar:FLOAT32,1.0
data b = scalar:FLOAT32,0.0
data reverse_channel_order = scalar:BOOL,0
read input input.png
node com.amd.nn_extension.convert_image_to_tensor input output a b reverse_channel_
↪order
write output input.f32
```

Example 2: 2x2 Upsample a tensor of type float32

Use the below GDF with RunVX.

```
import vx_nn
data input  = tensor:4,{80,80,3,1},VX_TYPE_FLOAT32,0
data output = tensor:4,{160,160,3,1},VX_TYPE_FLOAT32,0
read input  tensor.f32
node com.amd.nn_extension.upsample_nearest_layer input output
write output upsample.f32
```

2.8.12.2.8 AMD Module for OpenCV-interop from OpenVX (vx_opencv)

The vx_opencv is an OpenVX module that implemented a mechanism to access OpenCV functionality as OpenVX kernels. These kernels can be access from within OpenVX framework using OpenVX API call vxLoadKernels (context, "vx_opencv").

List of OpenCV-interop kernels

The following is a list of OpenCV functions that have been included in the vx_opencv module.

```
bilateralFilter            org.opencv.bilateralfilter
blur                       org.opencv.blur
boxfilter                  org.opencv.boxfilter
buildPyramid               org.opencv.buildpyramid
Dilate                     org.opencv.dilate
Erode                      org.opencv.erode
filter2D                   org.opencv.filter2d
GaussianBlur               org.opencv.gaussianblur
MedianBlur                 org.opencv.medianblur
morphologyEx               org.opencv.morphologyex
Laplacian                  org.opencv.laplacian
pyrDown                    org.opencv.pyrdown
pyrUp                      org.opencv.pyrup
sepFilter2D                org.opencv.sepfilter2d
Sobel                      org.opencv.sobel
```

(continues on next page)

(continued from previous page)

```
Scharr                      org.opencv.scharr
FAST                        org.opencv.fast
MSER                        org.opencv.mser_detect
ORB                         org.opencv.orb_detect
ORB_Compute                 org.opencv.orb_compute
BRISK                       org.opencv.brisk_detect
BRISK_Compute               org.opencv.brisk_compute
SimpleBlobDetector          org.opencv.simple_blob_detect
SimpleBlobDetector_Init     org.opencv.simple_blob_detect_initialize
SIFT_Detect                 org.opencv.sift_detect
SIFT_Compute                org.opencv.sift_compute
SURF_Detect                 org.opencv.surf_detect
SURF_Compute                org.opencv.surf_compute
STAR_FEATURE_Detector       org.opencv.star_detect
Canny                       org.opencv.canny
GoodFeature_Detector        org.opencv.good_features_to_track
buildOpticalFlowPyramid     org.opencv.buildopticalflowpyramid
DistanceTransform           org.opencv.distancetransform
Convert_Scale_Abs           org.opencv.convertscaleabs
addWeighted                 org.opencv.addweighted
Transpose                   org.opencv.transpose
Resize                      org.opencv.resize
AdaptiveThreshold           org.opencv.adaptivethreshold
Threshold                   org.opencv.threshold
cvtcolor                    org.opencv.cvtcolor
Flip                        org.opencv.flip
fastNlMeansDenoising        org.opencv.fastnlmeansdenoising
fastNlMeansDenoisingColored org.opencv.fastnlmeansdenoisingcolored
AbsDiff                     org.opencv.absdiff
Compare                     org.opencv.compare
bitwise_and                 org.opencv.bitwise_and
bitwise_not                 org.opencv.bitwise_not
bitwise_or                  org.opencv.bitwise_or
bitwise_xor                 org.opencv.bitwise_xor
Add                         org.opencv.add
Subtract                    org.opencv.subtract
Multiply                    org.opencv.multiply
Divide                      org.opencv.divide
WarpAffine                  org.opencv.warpaffine
WarpPerspective             org.opencv.warpperspective
```

2.8.12.2.8.1 Build Instructions

Pre-requisites

- OpenCV 3 download.
- CMake 2.8 or newer download.
- Build amdovx-core project at the same level folder as amdovx-modules build folder
- OpenCV_DIR environment variable should point to OpenCV/build folder

Build using Visual Studio Professional 2013 on 64-bit Windows 10/8.1/7

Use amdovx-modules/vx_opencv/vx_opencv.sln to build for x64 platform

Build using CMake on Linux (Ubuntu 15.10 64-bit)

- Use CMake to configure and generate Makefile

2.8.12.3 Applications

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.

2.8.12.3.1 Cloud Inference Application (cloud_inference)

- Cloud Inference Server: sample Inference Server
- Cloud Inference Client: sample Inference Client Application

2.8.12.3.2 Convert Neural Net models into AMD NNIR and OpenVX Code

This tool converts ONNX or Caffe models to AMD NNIR format and OpenVX code.

You need MIVisionX libraries to be able to build and run the generated OpenVX code.

Dependencies

- numpy
- onnx (0.2.1+)

How to use?

To convert an ONNX model into AMD NNIR model:

```
% python onnx2nnir.py model.pb nnirModelFolder
```

To convert a caffemodel into AMD NNIR model:

```
% python caffe2nnir.py <net.caffeModel> <nnirOutputFolder> --input-dims n,c,h,w [--
↪verbose 0|1]
```

To update batch size in AMD NNIR model:

```
% python nnir-update.py --batch-size N nnirModelFolder nnirModelFolderN
```

To fuse operations in AMD NNIR model (like batch normalization into convolution):

```
% python nnir-update.py --fuse-ops 1 nnirModelFolderN nnirModelFolderFused
```

To workaround groups using slice and concat operations in AMD NNIR model:

```
% python nnir-update.py --slice-groups 1 nnirModelFolderFused nnirModelFolderSliced
```

To convert an AMD NNIR model into OpenVX C code:

```
% python --help
```

Usage: python nnir2openvx.py [OPTIONS] <nnirInputFolder> <outputFolder>

OPTIONS:

--argmax UINT8 – argmax at the end with 8-bit output

--argmax UINT16 – argmax at the end with 16-bit output

–argmax <fileNamePrefix>rgb.txt – argmax at the end with RGB color mapping using LUT –argmax <fileNamePrefix>rgba.txt – argmax at the end with RGBA color mapping using LUT –help – show this help message

LUT File Format (RGB): 8-bit R G B values one per each label in text format R0 G0 B0 R1 G1 B1 …

LUT File Format (RGBA): 8-bit R G B A values one per each label in text format R0 G0 B0 A0 R1 G1 B1 A1 …

Here are few examples of OpenVX C code generation

Generate OpenVX and test code that can be used dump and compare raw tensor data:

```
% python nnir2openvx.py nnirInputFolderFused openvxCodeFolder
% mkdir openvxCodeFolder/build
% cd openvxCodeFolder/build
% cmake ..
% make
% ./anntest
```

Usage: anntest <weights.bin> [<input-data-file(s)> [<output-data-file(s)>]]]

<input-data-file>: is filename to initialize tensor

.jpg or .png: decode and initialize for 3 channel tensors (use %04d in fileName to when batch-size > 1: batch index starts from 0)

other: initialize tensor with raw data from the file

<output-data-file>[,<reference-for-compare>,<maxErrorLimit>,<rmsErrorLimit>]: <referece-to-compare> is raw tensor data for comparision <maxErrorLimit> is max absolute error allowed <rmsErrorLimit> is max RMS error allowed <output-data-file> is filename for saving output tensor data

'-' to ignore other: save raw tensor into the file

% ./anntest ../weights.bin input.f32 output.f32,reference.f32,1e-6,1e-9 …

Generate OpenVX and test code with argmax that can be used dump and compare 16-bit argmax output tensor:

```
 % python nnir2openvx.py --argmax UINT16 nnirInputFolderFused openvxCodeFolder
% mkdir openvxCodeFolder/build
% cd openvxCodeFolder/build
% cmake ..
% make
% ./anntest
```

Usage: anntest <weights.bin> [<input-data-file(s)> [<output-data-file(s)>]]]

<input-data-file>: is filename to initialize tensor

.jpg or .png: decode and initialize for 3 channel tensors (use %04d in fileName to when batch-size > 1: batch index starts from 0)

other: initialize tensor with raw data from the file

<output-data-file>[,<reference-for-compare>,<percentErrorLimit>]: <referece-to-compare> is raw tensor data of argmax output for comparision <percentMismatchLimit> is max mismatch (percentage) allowed <output-data-file> is filename for saving output tensor data

'-' to ignore other: save raw tensor into the file

% ./anntest ../weights.bin input-%04d.png output.u16,reference.u16,0.01 …

Generate OpenVX and test code with argmax and LUT that is designed for semantic segmentation use cases. You can dump output in raw format or PNGs and additionally compare with reference data in raw format.

```
% python nnir2openvx.py --argmax lut-rgb.txt nnirInputFolderFused openvxCodeFolder
% mkdir openvxCodeFolder/build
% cd openvxCodeFolder/build
% cmake ..
% make
% ./anntest
```

Usage: anntest <weights.bin> [<input-data-file(s)> [<output-data-file(s)>]]]

<input-data-file>: is filename to initialize tensor

.jpg or .png: decode and initialize for 3 channel tensors (use %04d in fileName to when batch-size > 1: batch index starts from 0)

other: initialize tensor with raw data from the file

<output-data-file>[,<reference-for-compare>,<percentErrorLimit>]: <referece-to-compare> is raw tensor data of LUT output for comparision <percentMismatchLimit> is max mismatch (percentage) allowed <output-data-file> is filename for saving output tensor data

.png: save LUT output as PNG file(s) (use %04d in fileName when batch-size > 1: batch index starts from 0)

'-' to ignore other: save raw tensor into the file

% ./anntest ../weights.bin input-%04d.png output.rgb,reference.rgb,0.01 … % ./anntest ../weights.bin input-%04d.png output-%04d.png,reference.rgb,0.01 …

2.8.12.3.3 Currently supported

Models

Support the below models from https://github.com/onnx/models

- resnet
- inception
- alexnet
- densenet
- sqeezenet

Operators

Supported ONNX operators are:

- Conv
- Relu
- MaxPool
- AveragePool
- GlobalAveragePool
- LRN

- BatchNormalization
- Concat
- Sum
- Add
- Sub
- Mul
- Softmax
- Dropout

License

2.8.12.4 Samples

MIVisionX samples using OpenVX and OpenVX extension libraries

2.8.12.4.1 GDF - Graph Description Format

MIVisionX samples using runvx with GDF

skintonedetect.gdf

usage:

```
runvx skintonedetect.gdf
```

canny.gdf

usage:

```
runvx canny.gdf
```

skintonedetect-LIVE.gdf

Using live camera

usage:

```
runvx -frames:live skintonedetect-LIVE.gdf
```

canny-LIVE.gdf

Using live camera

usage:

```
runvx -frames:live canny-LIVE.gdf
```

OpenCV_orb-LIVE.gdf

Using live camera

usage:

```
runvx -frames:live OpenCV_orb-LIVE.gdf
```

2.8.12.5 MIVisionX Toolkit

AMD MIVisionX Toolkit, is a comprehensive set of help tools for neural net creation, development, training and deployment. The Toolkit provides you with help tools to design, develop, quantize, prune, retrain, and infer your neural network work in any framework. The Toolkit is designed to help you deploy your work to any AMD or 3rd party hardware, from embedded to servers.

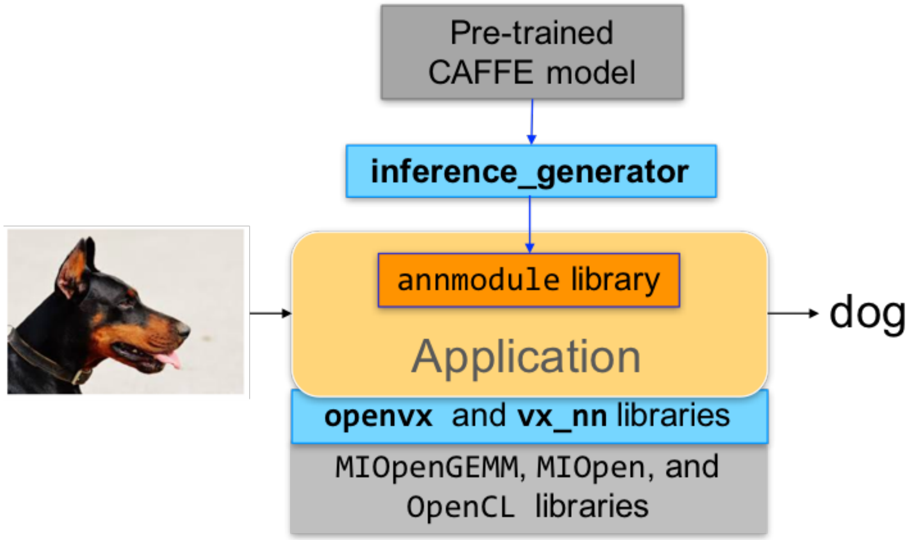
MIVisionX provides you with tools for accomplishing your tasks throughout the whole neural net life-cycle, from creating a model to deploying them for your target platforms.

2.8.12.6 Utilities

MIVisionX has utility applications which could be used by developers to test, quick prototype or develop sample applications.

- inference_generator : generate inference library from pre-trained CAFFE models
- loom_shell : an interpreter to prototype 360 degree video stitching applications using a script
- RunVX : command-line utility to execute OpenVX graph described in GDF text file
- RunCL : command-line utility to build, execute, and debug OpenCL programs

If you're interested in Neural Network Inference, start with the sample cloud inference application in apps folder.



2.8.12.7 Prerequisites

- 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.

2.8.12.7.1 Windows

- Windows 10
- Windows SDK
- Visual Studio 2017
- Install the latest drivers and *OpenCL SDK <https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases/tag/1.0>*_
- **OpenCV 3.4**
  - Set OpenCV_DIR environment variable to OpenCV/build folder
  - Add %OpenCV_DIR%x64vc14bin or %OpenCV_DIR%x64vc15bin to your PATH

2.8.12.7.2 Linux

- Install ROCm
- ROCm CMake, MIOpenGEMM & MIOpen for Neural Net Extensions (vx_nn)
- CMake 2.8 or newer download
- Qt Creator for Cloud Inference Client
- **Protobuf for inference generator & model compiler**
  - install libprotobuf-dev and protobuf-compiler needed for vx_nn
- **OpenCV 3.4**
  - Set OpenCV_DIR environment variable to OpenCV/build folder
- **FFMPEG n4.0.4 - Optional**
  - FFMPEG is required for amd_media & mv_deploy modules

2.8.12.7.3 Pre-requisites setup script - MIVisionX-setup.py

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 scipts 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.

2.8.12.7.4 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:1.8.1)]
                          --ffmpeg    [FFMPEG Installation - optional (default:no)␣
↪[options:Install ffmpeg - yes]]
```

2.8.12.8 Build MIVisionX

Note: use –installer yum for **CentOS** Refer to Wiki page for developer instructions.

2.8.12.9 Build & Install MIVisionX

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%x64vc14bin or %OpenCV_DIR%x64vc15bin 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

2.8.12.10 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

2.8.12.10.1 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

```
git clone https://github.com/GPUOpen-ProfessionalCompute-Libraries/MIVisionX.git
cd MIVisionX
```

```
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:1.8.1)]
                        --ffmpeg    [FFMPEG Installation - optional (default:no)␣
↪[options:Install ffmpeg - 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

2.8.12.10.2 Build 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/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)

2.8.12.10.3 Build annInferenceApp using Qt Creator

- build annInferenceApp.pro using Qt Creator
- or use annInferenceApp.py for simple tests

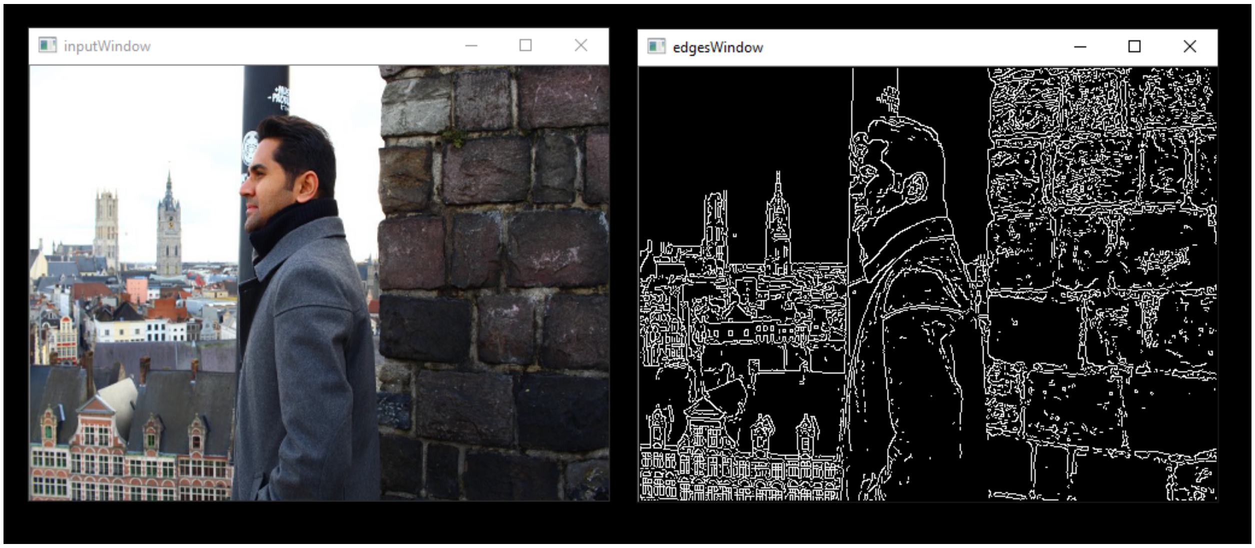
2.8.12.10.4 Build Radeon LOOM using Visual Studio Professional 2013 on 64-bit Windows 10/8.1/7

- Use loom.sln to build x64 platform

2.8.12.10.5 Verify the Installation

2.8.12.10.6 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**



```
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

2.8.12.11 Docker

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.

2.8.12.11.1 MIVisionX Docker

- Ubuntu 16.04
- Ubuntu 18.04
- CentOS 7.5
- CentOS 7.6

2.8.12.11.2 Docker Workflow Sample on Ubuntu 16.04

Prerequisites

- Ubuntu 16.04
- rocm supported hardware

2.8.12.11.3 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
```

(continues on next page)

(continued from previous page)

```
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/.
↪Xauthority
--volume /tmp/.X11-unix/:/tmp/.X11-unix mivisionx/ubuntu-16.04:latest
```

- Test display with MIVisionX sample

```
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
```

2.8.12.11.4 Supported Neural Net Layers

```
Layer name
Activation
Argmax
Batch Normalization
Concat
Convolution
Deconvolution
Fully Connected
Local Response Normalization (LRN)
Pooling
Scale
Slice
Softmax
```

(continues on next page)

(continued from previous page)

```
Tensor Add
Tensor Convert Depth
Tensor Convert from Image
Tensor Convert to Image
Tensor Multiply
Tensor Subtract
Upsample Nearest Neighborhood
```

2.8.12.12 Release Notes

2.8.12.12.1 Known issues

- Package (.deb & .rpm) install requires OpenCV v3.4.0 to execute AMD OpenCV extensions

2.8.12.12.2 Tested configurations

- Windows 10
- Linux: Ubuntu - 16.04/18.04 & CentOS - 7.5/7.6
- ROCm: rocm-dkms - 2.4.25
- rocm-cmake - github master:ac45c6e
- MIOpenGEMM - 1.1.5
- MIOpen - 1.8.1
- Protobuf - V3.5.2
- OpenCV - 3.4.0
- Dependencies for all the above packages

2.9 ROCm Libraries

2.9.1 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.

2.9.1.1 API design

Please refer to the rocFFTAPI for current documentation. Work in progress.

2.9.1.2 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 rocfft
```

2.9.1.3 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.

2.9.1.4 Manual build (all supported platforms)

If you use a distro other than Ubuntu, or would like more control over the build process, the 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.

2.9.1.5 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.

```
#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;
```

(continues on next page)

(continued from previous page)

```
        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;
        }

        // Free device buffer
        hipFree(x);

        return 0;
  }
```

2.9.1.6 API

This section provides details of the library API

2.9.1.6.1 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.

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

2.9.1.6.2 Library Setup and Cleanup

The following functions deals with initialization and cleanup of the library.

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.

2.9.1.6.3 Plan

The following functions are used to create and destroy plan objects.

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

2.9.1.6.4 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()`.

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

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 *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
- `[in] out_strides_size`: size of out_strides array (must be equal to transform dimensions)
- `[in] out_strides`: array of strides, in each dimension, of output buffer; if set to null ptr library chooses defaults
- `[in] out_distance`: distance between start of each data instance in output buffer

2.9.1.6.5 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_status **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

- `[in] plan`: plan handle
- `[inout] in_buffer`: array (of size 1 for interleaved data, of size 2 for planar data) of input buffers
- `[inout] out_buffer`: array (of size 1 for interleaved data, of size 2 for planar data) of output buffers, can be nullptr for inplace result placement
- `[in] info`: execution info handle created by rocfft_execution_info_create

2.9.1.6.6 Execution info

The execution api `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 *set* functions below need to be called before execution and *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

- `[out] info`: execution info handle

rocfft_status **rocfft_execution_info_destroy**(*rocfft_execution_info info*)

Destroy an execution info.

This API frees the execution info

Parameters

- `[in] info`: execution info handle

rocfft_status **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_status **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

2.9.1.6.7 Enumerations

This section provides all the enumerations used.

enum rocfft_status

rocfft status/error codes

Values:

rocfft_status_success

rocfft_status_failure

rocfft_status_invalid_arg_value

rocfft_status_invalid_dimensions

rocfft_status_invalid_array_type

rocfft_status_invalid_strides

rocfft_status_invalid_distance

rocfft_status_invalid_offset

enum rocfft_transform_type

Type of transform.

Values:

rocfft_transform_type_complex_forward

rocfft_transform_type_complex_inverse

rocfft_transform_type_real_forward

rocfft_transform_type_real_inverse

enum rocfft_precision

Precision.

Values:

rocfft_precision_single

rocfft_precision_double

enum rocfft_result_placement

Result placement.

Values:

rocfft_placement_inplace

rocfft_placement_notinplace

enum rocfft_array_type

Array type.

Values:

rocfft_array_type_complex_interleaved

rocfft_array_type_complex_planar

rocfft_array_type_real

rocfft_array_type_hermitian_interleaved

rocfft_array_type_hermitian_planar

enum rocfft_execution_mode

Execution mode.

Values:

rocfft_exec_mode_nonblocking

rocfft_exec_mode_nonblocking_with_flush

rocfft_exec_mode_blocking

2.9.2 rocBLAS

Please refer rocBLAS Github link

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.

2.9.2.1 Prerequisites

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

2.9.2.2 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
```

2.9.2.3 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)
```

2.9.2.4 Manual build (all supported platforms)

If you use a distro other than Ubuntu, or would like more control over the build process, the rocblaswiki 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.

2.9.2.5 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

2.9.2.6 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);
```

2.9.2.7 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.

2.9.2.8 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

2.9.2.9 API

This section provides details of the library API

2.9.2.9.1 Types

2.9.2.9.1.1 Definitions

2.9.2.9.1.2 rocblas_int

typedef int32_t **rocblas_int**

To specify whether int32 or int64 is used.

2.9.2.9.1.3 rocblas_long

typedef int64_t **rocblas_long**

2.9.2.9.1.4 rocblas_float_complex

typedef float2 **rocblas_float_complex**

2.9.2.9.1.5 rocblas_double_complex

typedef double2 **rocblas_double_complex**

2.9.2.9.1.6 rocblas_half

typedef uint16_t **rocblas_half**

2.9.2.9.1.7 rocblas_half_complex

typedef float2 **rocblas_half_complex**

2.9.2.9.1.8 rocblas_handle

typedef struct _rocblas_handle ***rocblas_handle**

2.9.2.9.1.9 Enums

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

2.9.2.9.1.10 rocblas_operation

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:

rocblas_operation_none = 111
Operate with the matrix.

rocblas_operation_transpose = 112
Operate with the transpose of the matrix.

rocblas_operation_conjugate_transpose = 113
Operate with the conjugate transpose of the matrix.

2.9.2.9.1.11 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:

rocblas_fill_upper = 121
Upper triangle.

rocblas_fill_lower = 122
Lower triangle.

rocblas_fill_full = 123

2.9.2.9.1.12 rocblas_diagonal

enum rocblas_diagonal

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

Values:

rocblas_diagonal_non_unit = 131
Non-unit triangular.

rocblas_diagonal_unit = 132
Unit triangular.

2.9.2.9.1.13 rocblas_side

enum rocblas_side

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

Values:

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

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

rocblas_side_both = 143

2.9.2.9.1.14 rocblas_status

enum rocblas_status
rocblas status codes definition

Values:

rocblas_status_success = 0
success

rocblas_status_invalid_handle = 1
handle not initialized, invalid or null

rocblas_status_not_implemented = 2
function is not implemented

rocblas_status_invalid_pointer = 3
invalid pointer parameter

rocblas_status_invalid_size = 4
invalid size parameter

rocblas_status_memory_error = 5
failed internal memory allocation, copy or dealloc

rocblas_status_internal_error = 6
other internal library failure

2.9.2.9.1.15 rocblas_datatype

enum rocblas_datatype
Indicates the precision width of data stored in a blas type.

Values:

rocblas_datatype_f16_r = 150

rocblas_datatype_f32_r = 151

rocblas_datatype_f64_r = 152

rocblas_datatype_f16_c = 153

rocblas_datatype_f32_c = 154

rocblas_datatype_f64_c = 155

rocblas_datatype_i8_r = 160

rocblas_datatype_u8_r = 161

rocblas_datatype_i32_r = 162

rocblas_datatype_u32_r = 163

rocblas_datatype_i8_c = 164

rocblas_datatype_u8_c = 165

rocblas_datatype_i32_c = 166

rocblas_datatype_u32_c = 167

2.9.2.9.1.16 rocblas_pointer_mode

enum rocblas_pointer_mode

Indicates the pointer is device pointer or host pointer.

Values:

rocblas_pointer_mode_host = 0

rocblas_pointer_mode_device = 1

2.9.2.9.1.17 rocblas_layer_mode

enum rocblas_layer_mode

Indicates if layer is active with bitmask.

Values:

rocblas_layer_mode_none = 0b0000000000

rocblas_layer_mode_log_trace = 0b0000000001

rocblas_layer_mode_log_bench = 0b0000000010

rocblas_layer_mode_log_profile = 0b0000000100

2.9.2.9.1.18 rocblas_gemm_algo

enum rocblas_gemm_algo

Indicates if layer is active with bitmask.

Values:

rocblas_gemm_algo_standard = 0b0000000000

2.9.2.9.2 Functions

2.9.2.9.2.1 Level 1 BLAS

2.9.2.9.2.2 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 scal the vector x[i] with scalar alpha, for i = 1 , … , n

```
x := alpha * x ,
```

Parameters

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

2.9.2.9.2.3 rocblas_<type>copy()

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

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 the vector x into the vector y, for i = 1 , … , n

```
y := x,
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[out] y`: pointer storing vector y on the GPU.
- `[in] incy`: rocblas_int specifies the increment for the elements of y.

2.9.2.9.2.4 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) perform dot product of vector x and y

```
result = x * y;
```

dotc perform dot product of complex vector x and complex y

```
result = conjugate (x) * y;
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the dot product. either on the host CPU or device GPU. return is 0.0 if n <= 0.

2.9.2.9.2.5 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 interchange vector x[i] and y[i], for i = 1 , … , n

```
y := x; x := y
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[inout] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[inout] y`: pointer storing vector y on the GPU.
- `[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 incx*, double **y*, *rocblas_int incy*)

2.9.2.9.2.6 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*)

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*)

BLAS Level 1 API.

axpy compute y := alpha * x + y

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.

2.9.2.9.2.7 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.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the asum product. either on the host CPU or device GPU. return is 0.0 if n, incx<=0.

2.9.2.9.2.8 rocblas_<type>nrm2()

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*)

BLAS Level 1 API.

nrm2 computes the euclidean norm of a real or complex vector := sqrt(x'*x) for real vector := sqrt(x**H*x) for complex vector

Parameters

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

2.9.2.9.2.9 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 real vector x or the sum of magnitude 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.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the amax index. either on the host CPU or device GPU. return is 0.0 if n, incx<=0.

2.9.2.9.2.10 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 real vector x or the sum of magnitude 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.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the amin index. either on the host CPU or device GPU. return is 0.0 if n, incx<=0.

2.9.2.9.2.11 Level 2 BLAS

2.9.2.9.2.12 rocblas_<type>gemv()

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*)

BLAS Level 2 API.

xGEMV performs one of the matrix-vector operations

```
y := alpha*A*x    + beta*y,   or
y := alpha*A**T*x + beta*y,   or
y := alpha*A**H*x + beta*y,
```

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
- `[in] m`: rocblas_int
- `[in] n`: rocblas_int
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[in] beta`: specifies the scalar beta.
- `[out] y`: pointer storing vector y on the GPU.
- `[in] incy`: rocblas_int specifies the increment for the elements of y.

2.9.2.9.2.13 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*x = alpha*b or A**T*x = alpha*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] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU, of dimension (lda, m)
- `[in] lda`: rocblas_int specifies the leading dimension of A. lda = max(1, m).
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.

2.9.2.9.2.14 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.

xHE(SY)MV performs the matrix-vector operation:

```
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(Symmetric) matrix.

BLAS Level 2 API

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] uplo`: rocblas_fill. specifies whether the upper or lower
- `[in] n`: rocblas_int.
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[in] beta`: specifies the scalar beta.
- `[out] y`: pointer storing vector y on the GPU.
- `[in] incy`: rocblas_int specifies the increment for the elements of y.

xGER performs the matrix-vector operations

```
A := A + alpha*x*y**T
```

where alpha is a 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] m`: rocblas_int
- `[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.
- `[in] y`: pointer storing vector y on the GPU.
- `[in] incy`: rocblas_int specifies the increment for the elements of y.
- `[inout] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.

2.9.2.9.2.15 rocblas_<type>syr()

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**(*rocblas_handle* *handle*, *rocblas_fill* *uplo*, *rocblas_int* *n*, **const** float **alpha*, **const** float **x*, *rocblas_int* *incx*, float **A*, *rocblas_int* *lda*)

BLAS Level 2 API.

xSYR performs the matrix-vector operations

```
A := A + alpha*x*x**T
```

where alpha is a scalars, 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.
- `[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.
- `[inout] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.

2.9.2.9.2.16 Level 3 BLAS

2.9.2.9.2.17 rocblas_<type>trtri_batched()

rocblas_status **rocblas_dtrtri_batched**(*rocblas_handle handle*, *rocblas_fill uplo*, *rocblas_diagonal diag*, *rocblas_int n*, **const** double **A*, *rocblas_int lda*, *rocblas_int stride_a*, double **invA*, *rocblas_int ldinvA*, *rocblas_int bsinvA*, *rocblas_int batch_count*)

rocblas_status **rocblas_strtri_batched**(*rocblas_handle handle*, *rocblas_fill uplo*, *rocblas_diagonal diag*, *rocblas_int n*, **const** float **A*, *rocblas_int lda*, *rocblas_int stride_a*, float **invA*, *rocblas_int ldinvA*, *rocblas_int bsinvA*, *rocblas_int batch_count*)

BLAS Level 3 API.

trtri compute the inverse of a matrix A

```
inv(A);
```

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`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] stride_a`: rocblas_int "batch stride a": stride from the start of one "A" matrix to the next
-

2.9.2.9.2.18 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

```
op(A)*X = alpha*B or  X*op(A) = alpha*B,
```

where alpha is a scalar, X and B are m by n matrices, A is triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A^T   or   op( A ) = A^H.
```

The matrix X is overwritten on B.

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: 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`: 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).
-

2.9.2.9.2.19 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 **beta*, 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 **beta*, float **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*)

BLAS Level 3 API.

xGEMM performs one of the matrix-matrix operations

```
C = alpha*op( A )*op( B ) + beta*C,
```

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 matrices, with op(A) an m by k matrix, 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 op(A)
- `[in] transB`: rocblas_operation, specifies the form of op(B)
- `[in] m`: rocblas_int, number or rows of matrices op(A) and C
- `[in] n`: rocblas_int, number of columns of matrices op(B) and C
- `[in] k`: rocblas_int, number of columns of matrix op(A) and number of rows of matrix op(B)
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int, specifies the leading dimension of A.
- `[in] B`: pointer storing matrix B on the GPU.
- `[in] ldb`: rocblas_int, specifies the leading dimension of B.
- `[in] beta`: specifies the scalar beta.
- `[inout] C`: pointer storing matrix C on the GPU.
- `[in] ldc`: rocblas_int, specifies the leading dimension of C.

2.9.2.9.2.20 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_int stride_a*, **const** double **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** double **beta*, double **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** float **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** float **beta*, float **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** *rocblas_half *B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** *rocblas_half *beta*, *rocblas_half *C*, *rocblas_int ldc*, *rocblas_int stride_c*, *rocblas_int batch_count*)

BLAS Level 3 API.

xGEMM_STRIDED_BATCHED performs one of the strided batched matrix-matrix operations

```
C[i*stride_c] = alpha*op( A[i*stride_a] )*op( B[i*stride_b] ) + beta*C[i*stride_
↪c], for i in
```

[0,batch_count-1]

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 dimention m.
- `[in] n`: rocblas_int. matrix dimention n.
- `[in] k`: rocblas_int. matrix dimention k.
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing strided batched matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of "A".
- `[in] stride_a`: rocblas_int stride from the start of one "A" matrix to the next
- `[in] B`: pointer storing strided batched matrix B on the GPU.
- `[in] ldb`: rocblas_int specifies the leading dimension of "B".
- `[in] stride_b`: rocblas_int stride from the start of one "B" matrix to the next
- `[in] beta`: specifies the scalar beta.
- `[inout] C`: pointer storing strided batched matrix C on the GPU.
- `[in] ldc`: rocblas_int specifies the leading dimension of "C".
- `[in] stride_c`: rocblas_int stride from the start of one "C" matrix to the next
- `[in] batch_count`: rocblas_int number of gemm operatons in the batch

2.9.2.9.2.21 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_int stride_a*, **const** double **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** double **beta*, double **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** float **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** float **beta*, float **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** *rocblas_half *B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** *rocblas_half *beta*, *rocblas_half *C*, *rocblas_int ldc*, *rocblas_int stride_c*, *rocblas_int batch_count*)

2.9.2.9.2.22 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*op( A ) + beta*op( B ),
```

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 matrices, with op(A) an m by n matrix, 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 op(A)
- `[in] transB`: rocblas_operation specifies the form of op(B)
- `[in] m`: rocblas_int.
- `[in] n`: rocblas_int.
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] beta`: specifies the scalar beta.
- `[in] B`: pointer storing matrix B on the GPU.
- `[in] ldb`: rocblas_int specifies the leading dimension of B.
- `[inout] C`: pointer storing matrix C on the GPU.
- `[in] ldc`: rocblas_int specifies the leading dimension of C.

2.9.2.9.2.23 BLAS Extensions

2.9.2.9.2.24 rocblas_gemm_ex()

rocblas_status **rocblas_gemm_ex**(*rocblas_handle handle*, *rocblas_operation trans_a*, *rocblas_operation trans_b*, *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*, size_t **workspace_size*, void **workspace*)

2.9.2.9.2.25 rocblas_gemm_strided_batched_ex()

rocblas_status **rocblas_gemm_strided_batched_ex**(*rocblas_handle handle*, *rocblas_operation trans_a*, *rocblas_operation trans_b*, *rocblas_int m*, *rocblas_int n*, *rocblas_int k*, **const** void **alpha*, **const** void **a*, *rocblas_datatype a_type*, *rocblas_int lda*, *rocblas_long stride_a*, **const** void **b*, *rocblas_datatype b_type*, *rocblas_int ldb*, *rocblas_long stride_b*, **const** void **beta*, **const** void **c*, *rocblas_datatype c_type*, *rocblas_int ldc*, *rocblas_long stride_c*, void **d*, *rocblas_datatype d_type*, *rocblas_int ldd*, *rocblas_long stride_d*, *rocblas_int batch_count*, *rocblas_datatype compute_type*, *rocblas_gemm_algo algo*, int32_t *solution_index*, uint32_t *flags*, size_t **workspace_size*, void **workspace*)

BLAS EX API.

GEMM_EX performs one of the matrix-matrix operations

```
D = alpha*op( A )*op( B ) + beta*C,
```

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, 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.

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 * specifies the scalar alpha. Same datatype as compute_type.
- `[in] a`: void * pointer storing matrix A on the GPU.
- `[in] a_type`: rocblas_datatype specifies the datatype of matrix A
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] b`: void * pointer storing matrix B on the GPU.
- `[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 * specifies the scalar beta. Same datatype as compute_type.
- `[in] c`: void * pointer storing matrix C on the GPU.
- `[in] c_type`: rocblas_datatype specifies the datatype of matrix C
- `[in] ldc`: rocblas_int specifies the leading dimension of C.
- `[out] d`: void * pointer storing matrix D on the GPU.
- `[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
- `[in] flags`: uint32_t reserved for future use
-

2.9.2.9.2.26 Build Information

2.9.2.9.2.27 rocblas_get_version_string()

rocblas_status **rocblas_get_version_string**(char **buf*, size_t *len*)

BLAS EX API.

GEMM_STRIDED_BATCHED_EX performs one of the strided_batched matrix-matrix operations

```
D[i*stride_d] = alpha*op(A[i*stride_a])*op(B[i*stride_b]) + beta*C[i*stride_c],␣
↪for i in
```

[0,batch_count-1]

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, C, and D are strided_batched matrices, with op(A) an m by k by batch_count strided_batched matrix, 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 batch_count.

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 * specifies the scalar alpha. Same datatype as compute_type.
- `[in] a`: void * pointer storing matrix A on the GPU.
- `[in] a_type`: rocblas_datatype specifies the datatype of matrix A
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] stride_a`: rocblas_long specifies stride from start of one "A" matrix to the next
- `[in] b`: void * pointer storing matrix B on the GPU.
- `[in] b_type`: rocblas_datatype specifies the datatype of matrix B
- `[in] ldb`: rocblas_int specifies the leading dimension of B.
- `[in] stride_b`: rocblas_long specifies stride from start of one "B" matrix to the next
- `[in] beta`: const void * specifies the scalar beta. Same datatype as compute_type.
- `[in] c`: void * pointer storing matrix C on the GPU.
- `[in] c_type`: rocblas_datatype specifies the datatype of matrix C

- `[in] ldc`: rocblas_int specifies the leading dimension of C.
- `[in] stride_c`: rocblas_long specifies stride from start of one "C" matrix to the next
- `[out] d`: void * pointer storing matrix D on the GPU.
- `[in] d_type`: rocblas_datatype specifies the datatype of matrix D
- `[in] ldd`: rocblas_int specifies the leading dimension of D.
- `[in] stride_d`: rocblas_long specifies stride from start of one "D" matrix to the next
- `[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
- `[in] flags`: uint32_t reserved for future use
-

2.9.2.9.2.28 Auxiliary

2.9.2.9.2.29 rocblas_pointer_to_mode()

rocblas_pointer_mode **rocblas_pointer_to_mode**(void **ptr*)

indicates whether the pointer is on the host or device. currently HIP API can only recoginize the input ptr on deive or not can not recoginize it is on host or not

2.9.2.9.2.30 rocblas_create_handle()

rocblas_status **rocblas_create_handle**(*rocblas_handle* **handle*)

2.9.2.9.2.31 rocblas_destroy_handle()

rocblas_status **rocblas_destroy_handle**(*rocblas_handle* *handle*)

2.9.2.9.2.32 rocblas_add_stream()

rocblas_status **rocblas_add_stream**(*rocblas_handle* *handle*, hipStream_t *stream*)

2.9.2.9.2.33 rocblas_set_stream()

rocblas_status **rocblas_set_stream**(*rocblas_handle* *handle*, hipStream_t *stream*)

2.9.2.9.2.34 rocblas_get_stream()

rocblas_status **rocblas_get_stream**(*rocblas_handle* *handle*, hipStream_t **stream*)

2.9.2.9.2.35 rocblas_set_pointer_mode()

rocblas_status **rocblas_set_pointer_mode**(*rocblas_handle handle*, *rocblas_pointer_mode pointer_mode*)

2.9.2.9.2.36 rocblas_get_pointer_mode()

rocblas_status **rocblas_get_pointer_mode**(*rocblas_handle handle*, *rocblas_pointer_mode *pointer_mode*)

2.9.2.9.2.37 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*)

2.9.2.9.2.38 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*)

2.9.2.9.2.39 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*)

2.9.2.9.2.40 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*)

2.9.2.10 All API

namespace rocblas

Functions

void **reinit_logs**()

file **rocblas-auxiliary.h**

#include <hip/hip_runtime_api.h>#include "rocblas-types.h" rocblas-auxiliary.h provides auxilary functions in rocblas

Defines

_ROCBLAS_AUXILIARY_H_

Functions

rocblas_pointer_mode **`rocblas_pointer_to_mode`**(void **ptr*)

indicates whether the pointer is on the host or device. currently HIP API can only recoginize the input ptr on deive or not can not recoginize it is on host or not

rocblas_status **`rocblas_create_handle`**(*rocblas_handle* **handle*)

rocblas_status **`rocblas_destroy_handle`**(*rocblas_handle handle*)

rocblas_status **`rocblas_add_stream`**(*rocblas_handle handle*, hipStream_t *stream*)

rocblas_status **`rocblas_set_stream`**(*rocblas_handle handle*, hipStream_t *stream*)

rocblas_status **`rocblas_get_stream`**(*rocblas_handle handle*, hipStream_t **stream*)

rocblas_status **`rocblas_set_pointer_mode`**(*rocblas_handle handle*, *rocblas_pointer_mode pointer_mode*)

rocblas_status **`rocblas_get_pointer_mode`**(*rocblas_handle handle*, *rocblas_pointer_mode* **pointer_mode*)

rocblas_status **`rocblas_set_vector`**(*rocblas_int n*, *rocblas_int elem_size*, **`const`** void **x*, *rocblas_int incx*, void **y*, *rocblas_int incy*)

rocblas_status **`rocblas_get_vector`**(*rocblas_int n*, *rocblas_int elem_size*, **`const`** void **x*, *rocblas_int incx*, void **y*, *rocblas_int incy*)

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*)

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*)

file **`rocblas-functions.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_FUNCTIONS_H_`

Functions

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

BLAS Level 1 API.

scal scal the vector x[i] with scalar alpha, for i = 1 , … , n

```
x := alpha * x ,
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[in] alpha`: specifies the scalar alpha.
- `[inout] x`: pointer storing vector x on the GPU.

- `[in] incx`: 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_scopy`**(*rocblas_handle* handle, *rocblas_int* *n*, **`const`** float **x*, *rocblas_int* *incx*, float **y*, *rocblas_int* *incy*)

BLAS Level 1 API.

copy copies the vector x into the vector y, for i = 1 , … , n

```
y := x,
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[out] y`: pointer storing vector y on the GPU.
- `[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_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) perform dot product of vector x and y

```
result = x * y;
```

dotc perform dot product of complex vector x and complex y

```
result = conjugate (x) * y;
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the dot product. either on the host CPU or device GPU. 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_sswap`**(*rocblas_handle* handle, *rocblas_int* *n*, float **x*, *rocblas_int* *incx*, float **y*, *rocblas_int* *incy*)

BLAS Level 1 API.

swap interchange vector x[i] and y[i], for i = 1 , … , n

```
y := x; x := y
```

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[inout] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[inout] y`: pointer storing vector y on the GPU.
- `[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 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*)

BLAS Level 1 API.

axpy compute y := alpha * x + y

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.

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

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_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.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.

- `[inout] result`: store the asum product. either on the host CPU or device GPU. return is 0.0 if n, incx<=0.

rocblas_status **rocblas_dasum**(*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*)

BLAS Level 1 API.

nrm2 computes the euclidean norm of a real or complex vector := sqrt(x'*x) for real vector := sqrt(x**H*x) for complex vector

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] n`: rocblas_int.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the nrm2 product. either on the host CPU or device GPU. 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_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 real vector x or the sum of magnitude 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.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the amax index. either on the host CPU or device GPU. return is 0.0 if n, incx<=0.

rocblas_status **rocblas_idamax**(*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 real vector x or the sum of magnitude 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.

- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: rocblas_int specifies the increment for the elements of y.
- `[inout] result`: store the amin index. either on the host CPU or device GPU. 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_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

```
y := alpha*A*x    + beta*y,   or
y := alpha*A**T*x + beta*y,   or
y := alpha*A**H*x + beta*y,
```

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
- `[in] m`: rocblas_int
- `[in] n`: rocblas_int
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[in] beta`: specifies the scalar beta.
- `[out] y`: pointer storing vector y on the GPU.
- `[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_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 = alpha*b or A**T*x = alpha*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] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU, of dimension (lda, m)
- `[in] lda`: rocblas_int specifies the leading dimension of A. lda = max(1, m).
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: 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_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.

xHE(SY)MV performs the matrix-vector operation:

```
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(Symmetric) matrix.

BLAS Level 2 API

Parameters

- `[in] handle`: rocblas_handle. handle to the rocblas library context queue.
- `[in] uplo`: rocblas_fill. specifies whether the upper or lower
- `[in] n`: rocblas_int.
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] x`: pointer storing vector x on the GPU.
- `[in] incx`: specifies the increment for the elements of x.
- `[in] beta`: specifies the scalar beta.
- `[out] y`: pointer storing vector y on the GPU.

- `[in] incy`: rocblas_int specifies the increment for the elements of y.

xGER performs the matrix-vector operations

```
A := A + alpha*x*y**T
```

where alpha is a 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] m`: rocblas_int
- `[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.
- `[in] y`: pointer storing vector y on the GPU.
- `[in] incy`: rocblas_int specifies the increment for the elements of y.
- `[inout] A`: pointer storing matrix A on the GPU.
- `[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_ssyr**(*rocblas_handle handle*, *rocblas_fill uplo*, *rocblas_int n*, **const** float **alpha*, **const** float **x*, *rocblas_int incx*, float **A*, *rocblas_int lda*)

BLAS Level 2 API.

xSYR performs the matrix-vector operations

```
A := A + alpha*x*x**T
```

where alpha is a scalars, 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.
- `[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.
- `[inout] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.

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_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`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
-

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*, *rocblas_int stride_a*, float **invA*, *rocblas_int ldinvA*, *rocblas_int bsinvA*, *rocblas_int batch_count*)

BLAS Level 3 API.

trtri compute the inverse of a matrix A

```
inv(A);
```

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`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] stride_a`: rocblas_int "batch stride a": stride from the start of one "A" matrix to the next
-

rocblas_status **rocblas_dtrtri_batched**(*rocblas_handle handle*, *rocblas_fill uplo*, *rocblas_diagonal diag*, *rocblas_int n*, **const** double **A*, *rocblas_int lda*, *rocblas_int stride_a*, double **invA*, *rocblas_int ldinvA*, *rocblas_int bsinvA*, *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

```
op(A)*X = alpha*B or  X*op(A) = alpha*B,
```

where alpha is a scalar, X and B are m by n matrices, A is triangular matrix and op(A) is one of

```
op( A ) = A   or   op( A ) = A^T   or   op( A ) = A^H.
```

The matrix X is overwritten on B.

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: 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`: 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).
-

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_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*)

BLAS Level 3 API.

xGEMM performs one of the matrix-matrix operations

```
C = alpha*op( A )*op( B ) + beta*C,
```

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 matrices, with op(A) an m by k matrix, 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 op(A)
- `[in] transB`: rocblas_operation, specifies the form of op(B)
- `[in] m`: rocblas_int, number or rows of matrices op(A) and C
- `[in] n`: rocblas_int, number of columns of matrices op(B) and C
- `[in] k`: rocblas_int, number of columns of matrix op(A) and number of rows of matrix op(B)
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int, specifies the leading dimension of A.
- `[in] B`: pointer storing matrix B on the GPU.
- `[in] ldb`: rocblas_int, specifies the leading dimension of B.
- `[in] beta`: specifies the scalar beta.
- `[inout] C`: pointer storing matrix C on the GPU.
- `[in] ldc`: rocblas_int, specifies the leading dimension of 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*)

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_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_int stride_a*, **const** *rocblas_half *B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** *rocblas_half *beta*, *rocblas_half *C*, *rocblas_int ldc*, *rocblas_int stride_c*, *rocblas_int batch_count*)

BLAS Level 3 API.

xGEMM_STRIDED_BATCHED performs one of the strided batched matrix-matrix operations

```
C[i*stride_c] = alpha*op( A[i*stride_a] )*op( B[i*stride_b] ) +␣
↪beta*C[i*stride_c], for i in
```

[0,batch_count-1]

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 dimention m.
- `[in] n`: rocblas_int. matrix dimention n.
- `[in] k`: rocblas_int. matrix dimention k.
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing strided batched matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of "A".
- `[in] stride_a`: rocblas_int stride from the start of one "A" matrix to the next
- `[in] B`: pointer storing strided batched matrix B on the GPU.
- `[in] ldb`: rocblas_int specifies the leading dimension of "B".
- `[in] stride_b`: rocblas_int stride from the start of one "B" matrix to the next
- `[in] beta`: specifies the scalar beta.
- `[inout] C`: pointer storing strided batched matrix C on the GPU.
- `[in] ldc`: rocblas_int specifies the leading dimension of "C".
- `[in] stride_c`: rocblas_int stride from the start of one "C" matrix to the next

- `[in] batch_count`: rocblas_int number of gemm operatons in the batch

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_int stride_a*, **const** float **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** float **beta*, float **C*, *rocblas_int ldc*, *rocblas_int stride_c*, *rocblas_int batch_count*)

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_int stride_a*, **const** double **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** double **beta*, double **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** *rocblas_half* **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** *rocblas_half* **beta*, *rocblas_half* **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** float **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** float **beta*, float **C*, *rocblas_int ldc*, *rocblas_int 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_int stride_a*, **const** double **B*, *rocblas_int ldb*, *rocblas_int stride_b*, **const** double **beta*, double **C*, *rocblas_int ldc*, *rocblas_int 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*op( A ) + beta*op( B ),
```

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 matrices, with op(A) an m by n matrix, 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 op(A)
- `[in] transB`: rocblas_operation specifies the form of op(B)
- `[in] m`: rocblas_int.
- `[in] n`: rocblas_int.
- `[in] alpha`: specifies the scalar alpha.
- `[in] A`: pointer storing matrix A on the GPU.
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] beta`: specifies the scalar beta.
- `[in] B`: pointer storing matrix B on the GPU.
- `[in] ldb`: rocblas_int specifies the leading dimension of B.
- `[inout] C`: pointer storing matrix C on the GPU.
- `[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 trans_a*, *rocblas_operation trans_b*, *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*, size_t **workspace_size*, void **workspace*) *rocblas_status* **rocblas_gemm_strided_batched_ex**(*rocblas_handle* handle, *rocblas_operation* *trans_a*, *rocblas_operation* *trans_b*, *rocblas_int* *m*, *rocblas_int* *n*, *rocblas_int* *k*, **const** void **alpha*, **const** void **a*, *rocblas_datatype* *a_type*, *rocblas_int* *lda*, *rocblas_long* *stride_a*, **const** void **b*, *rocblas_datatype* *b_type*, *rocblas_int* *ldb*, *rocblas_long* *stride_b*, **const** void **beta*, **const** void **c*, *rocblas_datatype* *c_type*, *rocblas_int* *ldc*, *rocblas_long* *stride_c*, void **d*, *rocblas_datatype* *d_type*, *rocblas_int* *ldd*, *rocblas_long* *stride_d*, *rocblas_int* batch_count, *rocblas_datatype* *compute_type*, *rocblas_gemm_algo* *algo*, int32_t *solution_index*, uint32_t *flags*, size_t **workspace_size*, void **workspace*)

BLAS EX API.

GEMM_EX performs one of the matrix-matrix operations

```
D = alpha*op( A )*op( B ) + beta*C,
```

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, 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.

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 * specifies the scalar alpha. Same datatype as compute_type.
- `[in] a`: void * pointer storing matrix A on the GPU.
- `[in] a_type`: rocblas_datatype specifies the datatype of matrix A
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] b`: void * pointer storing matrix B on the GPU.
- `[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 * specifies the scalar beta. Same datatype as compute_type.
- `[in] c`: void * pointer storing matrix C on the GPU.

- `[in] c_type`: rocblas_datatype specifies the datatype of matrix C
- `[in] ldc`: rocblas_int specifies the leading dimension of C.
- `[out] d`: void * pointer storing matrix D on the GPU.
- `[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
- `[in] flags`: uint32_t reserved for future use
-

rocblas_status **`rocblas_get_version_string`**(char **buf*, size_t *len*)

BLAS EX API.

GEMM_STRIDED_BATCHED_EX performs one of the strided_batched matrix-matrix operations

```
D[i*stride_d] = alpha*op(A[i*stride_a])*op(B[i*stride_b]) + beta*C[i*stride_
↪c], for i in
```

[0,batch_count-1]

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, C, and D are strided_batched matrices, with op(A) an m by k by batch_count strided_batched matrix, 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 batch_count.

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 * specifies the scalar alpha. Same datatype as compute_type.
- `[in] a`: void * pointer storing matrix A on the GPU.
- `[in] a_type`: rocblas_datatype specifies the datatype of matrix A
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] stride_a`: rocblas_long specifies stride from start of one “A” matrix to the next

- `[in] b`: void * pointer storing matrix B on the GPU.
- `[in] b_type`: rocblas_datatype specifies the datatype of matrix B
- `[in] ldb`: rocblas_int specifies the leading dimension of B.
- `[in] stride_b`: rocblas_long specifies stride from start of one "B" matrix to the next
- `[in] beta`: const void * specifies the scalar beta. Same datatype as compute_type.
- `[in] c`: void * pointer storing matrix C on the GPU.
- `[in] c_type`: rocblas_datatype specifies the datatype of matrix C
- `[in] ldc`: rocblas_int specifies the leading dimension of C.
- `[in] stride_c`: rocblas_long specifies stride from start of one "C" matrix to the next
- `[out] d`: void * pointer storing matrix D on the GPU.
- `[in] d_type`: rocblas_datatype specifies the datatype of matrix D
- `[in] ldd`: rocblas_int specifies the leading dimension of D.
- `[in] stride_d`: rocblas_long specifies stride from start of one "D" matrix to the next
- `[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
- `[in] flags`: uint32_t reserved for future use
-

file **rocblas-types.h**

#include <stddef.h>#include <stdint.h>#include <hip/hip_vector_types.h> *rocblas-types.h* defines data types used by rocblas

Defines

_ROCBLAS_TYPES_H_

Typedefs

typedef int32_t **rocblas_int**

To specify whether int32 or int64 is used.

typedef int64_t **rocblas_long**

typedef float2 **rocblas_float_complex**

typedef double2 **rocblas_double_complex**

typedef uint16_t **rocblas_half**

typedef float2 **rocblas_half_complex**

typedef struct _rocblas_handle ***rocblas_handle**

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:

`rocblas_operation_none` = 111

Operate with the matrix.

`rocblas_operation_transpose` = 112

Operate with the transpose of the matrix.

`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:

`rocblas_fill_upper` = 121

Upper triangle.

`rocblas_fill_lower` = 122

Lower triangle.

`rocblas_fill_full` = 123

`enum rocblas_diagonal`

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

Values:

`rocblas_diagonal_non_unit` = 131

Non-unit triangular.

`rocblas_diagonal_unit` = 132

Unit triangular.

`enum rocblas_side`

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

Values:

`rocblas_side_left` = 141

Multiply general matrix by symmetric, Hermitian or triangular matrix on the left.

`rocblas_side_right` = 142

Multiply general matrix by symmetric, Hermitian or triangular matrix on the right.

`rocblas_side_both` = 143

`enum rocblas_status`

rocblas status codes definition

Values:

`rocblas_status_success` = 0

success

`rocblas_status_invalid_handle` = 1
handle not initialized, invalid or null

`rocblas_status_not_implemented` = 2
function is not implemented

`rocblas_status_invalid_pointer` = 3
invalid pointer parameter

`rocblas_status_invalid_size` = 4
invalid size parameter

`rocblas_status_memory_error` = 5
failed internal memory allocation, copy or dealloc

`rocblas_status_internal_error` = 6
other internal library failure

`enum rocblas_datatype`
Indicates the precision width of data stored in a blas type.

Values:

`rocblas_datatype_f16_r` = 150

`rocblas_datatype_f32_r` = 151

`rocblas_datatype_f64_r` = 152

`rocblas_datatype_f16_c` = 153

`rocblas_datatype_f32_c` = 154

`rocblas_datatype_f64_c` = 155

`rocblas_datatype_i8_r` = 160

`rocblas_datatype_u8_r` = 161

`rocblas_datatype_i32_r` = 162

`rocblas_datatype_u32_r` = 163

`rocblas_datatype_i8_c` = 164

`rocblas_datatype_u8_c` = 165

`rocblas_datatype_i32_c` = 166

`rocblas_datatype_u32_c` = 167

`enum rocblas_pointer_mode`
Indicates the pointer is device pointer or host pointer.

Values:

`rocblas_pointer_mode_host` = 0

`rocblas_pointer_mode_device` = 1

`enum rocblas_layer_mode`
Indicates if layer is active with bitmask.

Values:

`rocblas_layer_mode_none` = 0b0000000000

`rocblas_layer_mode_log_trace` = 0b0000000001

rocblas_layer_mode_log_bench = 0b0000000010

rocblas_layer_mode_log_profile = 0b0000000100

enum rocblas_gemm_algo

Indicates if layer is active with bitmask.

Values:

rocblas_gemm_algo_standard = 0b0000000000

file **rocblas.h**

#include <stdbool.h>#include "rocblas-export.h"#include "rocblas-version.h"#include "rocblas-types.h"#include "rocblas-auxiliary.h"#include "rocblas-functions.h" *rocblas.h* includes other *.h and exposes a common interface

Defines

_ROCBLAS_H_

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*)

BLAS EX API.

GEMM_STRIDED_BATCHED_EX performs one of the strided_batched matrix-matrix operations

```
D[i*stride_d] = alpha*op(A[i*stride_a])*op(B[i*stride_b]) + beta*C[i*stride_
↪c], for i in
```

[0,batch_count-1]

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, C, and D are strided_batched matrices, with op(A) an m by k by batch_count strided_batched matrix, 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 batch_count.

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 * specifies the scalar alpha. Same datatype as compute_type.
- `[in] a`: void * pointer storing matrix A on the GPU.
- `[in] a_type`: rocblas_datatype specifies the datatype of matrix A
- `[in] lda`: rocblas_int specifies the leading dimension of A.
- `[in] stride_a`: rocblas_long specifies stride from start of one “A” matrix to the next
- `[in] b`: void * pointer storing matrix B on the GPU.
- `[in] b_type`: rocblas_datatype specifies the datatype of matrix B
- `[in] ldb`: rocblas_int specifies the leading dimension of B.
- `[in] stride_b`: rocblas_long specifies stride from start of one “B” matrix to the next
- `[in] beta`: const void * specifies the scalar beta. Same datatype as compute_type.
- `[in] c`: void * pointer storing matrix C on the GPU.
- `[in] c_type`: rocblas_datatype specifies the datatype of matrix C
- `[in] ldc`: rocblas_int specifies the leading dimension of C.
- `[in] stride_c`: rocblas_long specifies stride from start of one “C” matrix to the next
- `[out] d`: void * pointer storing matrix D on the GPU.
- `[in] d_type`: rocblas_datatype specifies the datatype of matrix D
- `[in] ldd`: rocblas_int specifies the leading dimension of D.
- `[in] stride_d`: rocblas_long specifies stride from start of one “D” matrix to the next
- `[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
- `[in] flags`: uint32_t reserved for future use
-

file **`handle.cpp`**

#include “handle.h”#include <cstdlib>

Functions

`static` 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 suceeds, 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 **`rocblas_auxiliary.cpp`**

#include <stdio.h>#include <hip/hip_runtime.h>#include "definitions.h"#include "rocblas-types.h"#include "handle.h"#include "logging.h"#include "utility.h"#include "rocblas_unique_ptr.hpp"#include "rocblas-auxiliary.h"

Functions

rocblas_pointer_mode **`rocblas_pointer_to_mode`**(void **ptr*)

indicates whether the pointer is on the host or device. currently HIP API can only recoginize the input ptr on deive or not can not recoginize it is on host or not

rocblas_status **`rocblas_get_pointer_mode`**(*rocblas_handle handle*, *rocblas_pointer_mode *mode*)

rocblas_status **`rocblas_set_pointer_mode`**(*rocblas_handle handle*, *rocblas_pointer_mode mode*)

rocblas_status **`rocblas_create_handle`**(*rocblas_handle *handle*)

rocblas_status **`rocblas_destroy_handle`**(*rocblas_handle handle*)

rocblas_status **`rocblas_set_stream`**(*rocblas_handle handle*, hipStream_t *stream_id*)

rocblas_status **`rocblas_get_stream`**(*rocblas_handle handle*, hipStream_t **stream_id*)

`__global__ void copy_void_ptr_vector_kernel(rocblas_int n, rocblas_int elem_size, cons`

rocblas_status **`rocblas_set_vector`**(*rocblas_int n*, *rocblas_int elem_size*, **`const`** void **x_h*, *rocblas_int incx*, void **y_d*, *rocblas_int incy*)

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*)

`__global__ void copy_void_ptr_matrix_kernel(rocblas_int rows, rocblas_int cols, size_t`

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*)

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*)

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**

2.9.3 hipBLAS

2.9.3.1 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.

2.9.3.1.1 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
```

2.9.3.1.2 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.

2.9.3.2 Build

2.9.3.2.1 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
```

2.9.3.2.2 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)
```

2.9.3.2.3 Build Library Using Individual Commands

```
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
```

2.9.3.2.4 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)

```
mkdir -p [HIPBLAS_BUILD_DIR]/release/deps
cd [HIPBLAS_BUILD_DIR]/release/deps
ccmake -DBUILD_BOOST=OFF [HIPBLAS_SOURCE]/deps   # 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>"`

```
# Default install location is in /opt/rocm, use -DCMAKE_INSTALL_PREFIX=<path> to␣
↪specify other
CXX=/opt/rocm/bin/hcc ccmake -DBUILD_CLIENTS_TESTS=ON -DBUILD_CLIENTS_BENCHMARKS=ON␣
↪[HIPBLAS_SOURCE]
make -j$(nproc)
sudo make install   # sudo required if installing into system directory such as /opt/
↪rocm
```

2.9.3.2.5 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

2.9.3.3 Running

2.9.3.3.1 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.

2.9.3.3.2 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

2.9.3.3.3 GEMV API

```
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 );
```

2.9.3.3.4 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

```
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.

2.9.4 Tensile

2.9.4.1 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 a 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_BenchmarkProblems: has all the problems descriptions and executables generated during benchmarking, where you can re-launch exe to reproduce results.

    2_BenchmarkData: has the raw performance results.

    3_LibraryLogic: has optimal kernel configurations yaml file and Winner*.csv. Usually rocBLAS takes the yaml files from this folder.

    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.

2.9.4.1.1 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
```

2.9.4.2 Benchmark Config example

Tensile uses an incremental and "programmable" benchmarking protocol.

2.9.4.2.1 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 ]
```

(continues on next page)

(continued from previous page)

```
        - 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:
```

2.9.4.2.2 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.

2.9.4.2.3 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.

2.9.4.2.4 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.

2.9.4.2.5 Solution / Kernel Parameters

See: Kernel Parameters.

2.9.4.2.6 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.

2.9.4.3 Benchmark Protocol

2.9.4.3.1 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.

2.9.4.3.2 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.

2.9.4.3.3 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
```

(continues on next page)

(continued from previous page)

```
- # Benchmark Size-Group
 InitialSolutionParameters:
   - WorkGroup: [ [ 16, 16, 1 ] ]
   - NumLoadsCoalescedA: [ 1 ]
   - NumLoadsCoalescedB: [ 1 ]
   - 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] ]
```

2.9.4.3.4 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.

2.9.4.3.5 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:

```
Range: [ [16, 128], [16, 128], [16, 128] ] # n = 512
Range: [ [16, 128], 0, 0] # n = 8
Range: [ [16, 16, 16, 5760], 0, [1024, 1024, 4096] ] # n = 108
```

2.9.4.3.6 Benchmark Common 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.

2.9.4.3.7 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.

2.9.4.3.8 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.

2.9.4.3.9 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.

2.9.4.3.10 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.

2.9.4.3.11 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.

2.9.4.4 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.

2.9.4.5 Dependencies

2.9.4.5.1 CMake

- CMake 2.8

2.9.4.5.2 Python

(One time only)

- Ubuntu: sudo apt install python2.7 python-yaml
- CentOS: sudo yum install python PyYAML

- Fedora: sudo dnf install python PyYAML

2.9.4.5.3 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

2.9.4.6 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
```

2.9.4.7 Kernel Parameters

2.9.4.7.1 Solution / Kernel Parameters

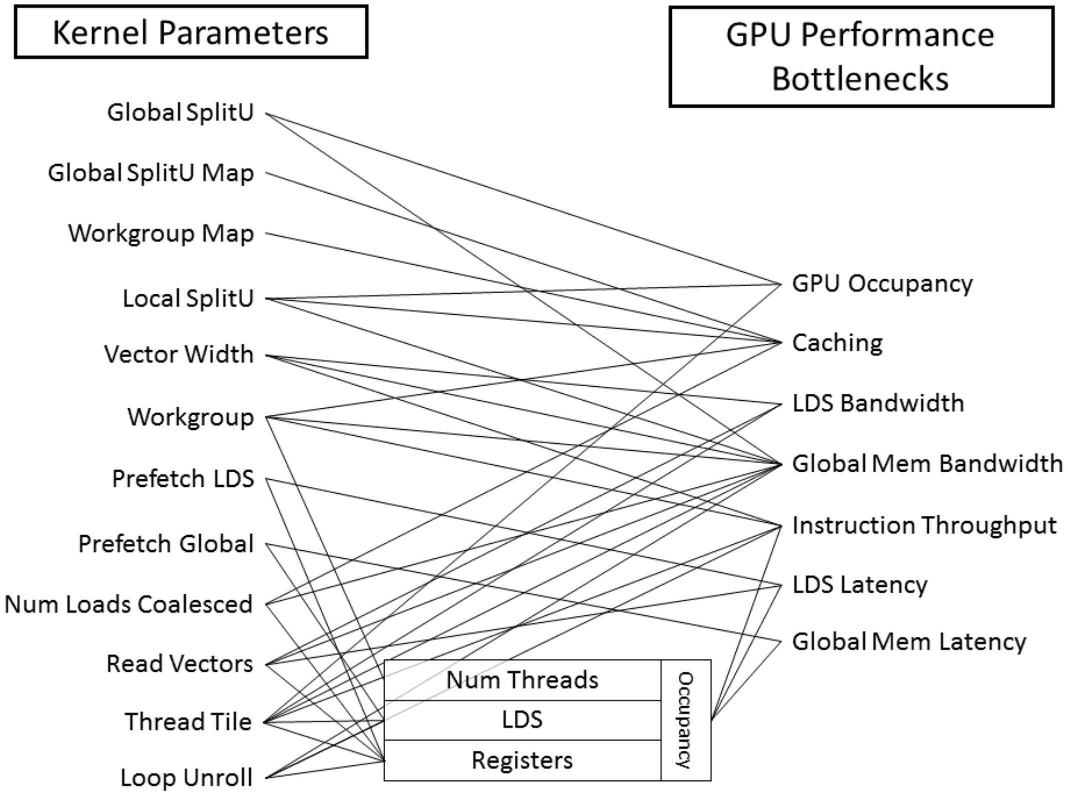
- **LoopDoWhile:** True=DoWhile loop, False=While or For loop
- **LoopTail:** Additional loop with LoopUnroll=1.
- **EdgeType:** Branch, ShiftPtr or None
- **WorkGroup:** [dim0, dim1, LocalSplitU]
- **ThreadTile:** [dim0, dim1]
- **GlobalSplitU:** Split up summation among work-groups to create more concurrency. This option launches a kernel to handle the beta scaling, then a second kernel where the writes to global memory are atomic.
- **PrefetchGlobalRead:** True means outer loop should prefetch global data one iteration ahead.
- **PrefetchLocalRead:** True means inner loop should prefetch lds data one iteration ahead.
- **WorkGroupMapping:** In what order will work-groups compute C; affects cacheing.

- **LoopUnroll:** How many iterations to unroll inner loop; helps loading coalesced memory.
- **MacroTile:** Derrived from WorkGroup*ThreadTile.
- **DepthU:** Derrived from LoopUnroll*SplitU.
- **NumLoadsCoalescedA,B:** Number of loads from A in coalesced dimension.
- **GlobalReadCoalesceGroupA,B:** True means adjacent threads map to adjacent global read elements (but, if transposing data then write to lds is scattered).
- **GlobalReadCoalesceVectorA,B:** True means vector components map to adjacent global read elements (but, if transposing data then write to lds is scattered).
- **VectorWidth:** 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.
- **KernelLanguage:** Whether kernels should be written in source code (HIP, OpenCL) or assembly (gfx803, gfx900, . . .).

The exhaustive list of solution parameters and their defaults is stored in Common.py.

2.9.4.7.2 Kernel Parameters Affect Performance

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.



2.9.4.7.3 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.

2.9.4.7.4 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.

2.9.4.7.5 GPU Kernel Dimension

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. Withing the GPU kernel, dimensions-2 is reconstituted back into whatever dimensions it represents.

2.9.4.8 Languages

2.9.4.8.1 Tensile Benchmarking is Python

The benchmarking module, Tensile.py, is written in python. The python scripts write solution, kernels, cmake files and all other C/C++ files used for benchmarking.

2.9.4.8.2 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.

2.9.4.8.3 Device Languages

The device languages Tensile supports for the gpu kernels is

- OpenCL 1.2
- HIP
- Assembly
  - gfx803

  - gfx900

2.9.4.9 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.

2.9.4.10 Problem Nomenclature

2.9.4.10.1 Example Problems

- Standard GEMM has 4 variants (2 free indices (i, j) and 1 summation index l)
  1. N(N:nontranspose)N: C[i,j] = Sum[l] A[i,l] * B[l,j]
  2. NT(T:transpose): C[i,j] = Sum[l] A[i,l] * B[j, l]
  3. TN: C[i,j] = Sum[l] A[l, i] * B[l,j]
  4. TT: C[i,j] = Sum[l] A[l, i] * B[j, l]
- C[i,j,k] = Sum[l] A[i,l,k] * B[l,j,k] (batched-GEMM; 2 free indices, 1 batched index k and 1 summation index l)
- C[i,j] = Sum[k,l] A[i,k,l] * B[j,l,k] (2D summation)
- C[i,j,k,l,m] = Sum[n] A[i,k,m,l,n] * B[j,k,l,n,m] (GEMM with 3 batched indices)
- C[i,j,k,l,m] = Sum[n,o] A[i,k,m,o,n] * B[j,m,l,n,o] (4 free indices, 2 summation indices and 1 batched index)
- C[i,j,k,l] = Sum[m,n] A[i,j,m,n,l] * B[m,n,k,j,l] (batched image convolution mapped to 7D tensor contraction)
- and even crazier

2.9.4.10.2 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 (Sum[k]) and in tensors A and B. It is along these indices that we perform the inner products (pairwise multiply then sum).

2.9.4.10.3 Limitations

Problem supported by Tensile must meet the following conditions:

There must be at least one pair of free indices.

2.9.4.11 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:

```
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?

2.9.4.12 Versioning

Tensile follows semantic versioning practices, i.e. Major.Minor.Patch, in BenchmarkConfig.yaml files, LibraryConfig.yaml files and in cmake find_package. Tensile is compatible with a "MinimumRequiredVersion" if Tensile.Major==MRV.Major and Tensile.Minor.Patch >= MRV.Minor.Patch.

- **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.

2.9.5 rocThrust

HIP back-end for Thrust(alpha release)

2.9.5.1 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.

2.9.5.2 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.

2.9.5.3 Hardware

Visit the following link for ROCm hardware requirements:

2.9.5.4 Build And Install

For build and install:

```
git clone https://github.com/ROCmSoftwarePlatform/rocThrust

# Go to rocThrust directory, create and go to the build directory.
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 ../.
```

(continues on next page)

(continued from previous page)

```

# Build
make -j4
# Optionally, run tests if they're enabled.
ctest --output-on-failure

# Package
make package

# Install
[sudo] make install
```

2.9.5.5 Using rocThrust In A Project

```
Recommended way of including rocThrust into a CMake project is by using its package␣
↪configuration files.

# 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)
```

2.9.5.6 Running Unit Tests

```
# Go to rocThrust build directory
cd rocThrust; cd build

# To run all tests
ctest

# To run unit tests for rocThrust
./test/<unit-test-name>
```

2.9.5.7 Documentation

Documentation is available here.

2.9.5.8 Support

Bugs and feature requests can be reported through the issue tracker.

2.9.6 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.

2.9.6.1 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.

2.9.6.2 Build And Install

```
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 'CXX=<path-to-hcc>
↪'
# 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.
ctest --output-on-failure

# Package
make package
```

(continues on next page)

(continued from previous page)

```
# Install
[sudo] make install
```

2.9.6.3 Using hipCUB In A Project

Recommended way of including hipCUB into a CMake project is by using its package configuration files.

```
# 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:

```
#include <hipcub/hipcub.hpp>
```

CUB or rocPRIM headers are included by hipCUB depending on the current HIP platform.

2.9.6.4 Running Unit Tests

```
# Go to hipCUB build directory
cd hipCUB; cd build

# To run all tests
ctest

# To run unit tests for hipCUB
./test/hipcub/<unit-test-name>
```

2.9.6.5 Documentation

```
# go to hipCUB doc directory
cd hipCUB; cd doc

# run doxygen
doxygen Doxyfile

# open html/index.html
```

2.9.6.6 Support

Bugs and feature requests can be reported through the issue tracker.

2.9.6.7 Contributions and License

Contributions of any kind are most welcome! More details are found at CONTRIBUTING and LICENSE.

2.9.7 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.

2.9.7.1 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.

2.9.7.2 Building ROCm SMI

2.9.7.2.1 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 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
```

The built library will appear in the build folder.

2.9.7.3 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.

2.9.7.4 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
```

To run the test, execute the program rsmitst that is built from the steps above. Make sure ROCm SMI library is in your library search path when executing the test program.

2.9.7.5 Usage Basics

2.9.7.5.1 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.

2.9.7.5.2 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:

```
#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);
```

(continues on next page)

(continued from previous page)

```
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;
}
```

2.9.8 RCCL

ROCm Communication Collectives Library

2.9.8.1 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.

2.9.8.2 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)

2.9.8.3 Quickstart RCCL Build

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)

2.9.8.4 Manual build

2.9.8.4.1 To build the library :

```
$ git clone https://github.com/ROCmSoftwarePlatform/rccl.git
$ cd rccl
$ mkdir build
$ cd build
$ CXX=/opt/rocm/bin/hcc cmake -DCMAKE_INSTALL_PREFIX=$PWD/rccl-install ..
$ make -j 8
```

You may substitute a path of your own choosing for CMAKE_INSTALL_PREFIX. Note: ensure rocm-cmake is installed,

```
apt install rocm-cmake.
```

2.9.8.5 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:

```
$ 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 librccl.so.

2.9.8.6 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:

```
$ 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.

2.9.8.7 Library and API Documentation

Please refer to the Library documentation for current documentation.

2.9.8.8 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.

2.9.9 rocALUTION

2.9.9.1 Introduction

2.9.9.2 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/recursive), 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

2.9.9.3 Building and Installing

2.9.9.4 Installing from AMD ROCm repositories

TODO, not yet available

2.9.9.5 Building rocALUTION from Open-Source repository

2.9.9.6 Download rocALUTION

The rocALUTION source code is available at the rocALUTION github page. Download the master branch using:

```
git clone -b master https://github.com/ROCmSoftwarePlatform/rocALUTION.git
cd rocALUTION
```

Note that if you want to contribute to rocALUTION, you will need to checkout the develop branch instead of the master branch. See rocalution_contributing for further details. 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.

2.9.9.7 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.

Command	Description
./install.sh -h	Print help information.
./install.sh -d	Build dependencies and library in your local directory. The *-d* flag only needs to be **|br|** used once. For subsequent invocations of *install.sh* it is not necessary to rebuild the **|br|** dependencies.
./install.sh	Build library in your local directory. It is assumed dependencies are available.
./install.sh -i	Build library, then build and install rocALUTION package in */opt/rocm/rocalution*. You will **|br|** be prompted for sudo access. This will install for all users.
./install.sh –host	Build library in your local directory without HIP support. It is assumed dependencies **|br|** are available.
./install.sh –mpi	Build library in your local directory with HIP and MPI support. It is assumed **|br|** dependencies are available.

2.9.9.8 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.

Command	Description
./install.sh -h	Print help information.
./install.sh -dc	Build dependencies, library and client in your local directory. The *-d* flag only needs to **|br|** be used once. For subsequent invocations of *install.sh* it is not necessary to rebuild the **|br|** dependencies.
./install.sh -c	Build library and client in your local directory. It is assumed dependencies are available.
./install.sh -idc	Build library, dependencies and client, then build and install rocALUTION package in **|br|** */opt/rocm/rocalution*. You will be prompted for sudo access. This will install for all users.
./install.sh -ic	Build library and client, then build and install rocALUTION package in **|br|** *opt/rocm/rocalution*. You will be prompted for sudo access. This will install for all users.

2.9.9.9 Using individual commands to build rocALUTION

CMake 3.5 or later is required in order to build rocALUTION.

rocALUTION can be built with cmake using the following commands:

```
# Create and change to build directory
mkdir -p build/release ; cd build/release
```

(continues on next page)

(continued from previous page)

```
# 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:

```
# 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.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path_to_rocalution>
```

2.9.9.10 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-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:** For Carrizo - HCC RUNTIME ERROR: Failed to find compatible kernel

   Solution: Add the following to the cmake command when configuring: *-DCMAKE_CXX_FLAGS=”–amdgpu-target=gfx801”*

3. **Issue:** For MI25 (Vega10 Server) - HCC RUNTIME ERROR: Failed to find compatible kernel

   Solution: *export HCC_AMDGPU_TARGET=gfx900*

4. **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

5. **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

6. **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

2.9.9.11 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

wget ftp://math.nist.gov/pub/MatrixMarket2/Harwell-Boeing/laplace/gr_30_30.mtx.gz
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

2.9.9.12 API

This section provides details of the library API

2.9.9.12.1 Host Utility Functions

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.

2.9.9.12.2 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 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

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

void `rocalution::`**`info_rocalution`**(void)

Print info about rocALUTION.

`info_rocalution` prints information about the rocALUTION platform

void `rocalution::`**`info_rocalution`**(**`const struct`** Rocalution_Backend_Descriptor *backend_descriptor*)

Print info about specific rocALUTION backend descriptor.

`info_rocalution` prints information about the rocALUTION platform of the specific backend descriptor.

Parameters

- `[in] backend_descriptor`: rocALUTION backend descriptor

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.

2.9.9.12.3 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 >*

virtual void `rocalution::BaseRocalution::`**MoveToAccelerator**(void) = 0

Move the object to the accelerator backend.

virtual void `rocalution::BaseRocalution::`**MoveToHost**(void) = 0

Move the object to the host backend.

virtual void `rocalution::BaseRocalution::`**MoveToAcceleratorAsync**(void)

Move the object to the accelerator backend with async move.

virtual void `rocalution::BaseRocalution::`**MoveToHostAsync**(void)

Move the object to the host backend with async move.

virtual void `rocalution::BaseRocalution::`**Sync**(void)

Sync (the async move)

virtual void `rocalution::BaseRocalution::`**CloneBackend**(**const** BaseRocalution<ValueType> &*src*)

Clone the Backend descriptor from another object.

With `CloneBackend`, the backend can be cloned without copying any data. This is especially useful, if several objects should reside on the same backend, but keep their original data.

Example

```
LocalVector<ValueType> vec;
LocalMatrix<ValueType> mat;

// Allocate and initialize vec and mat
// ...

LocalVector<ValueType> tmp;
// By cloning backend, tmp and vec will have the same backend as mat
tmp.CloneBackend(mat);
vec.CloneBackend(mat);

// The following matrix vector multiplication will be performed on the backend
// selected in mat
mat.Apply(vec, &tmp);
```

Parameters

- `[in] src`: Object, where the backend should be cloned from.

virtual 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

```
mat.Info();
vec.Info();
```

virtual void `rocalution::`*BaseRocalution*`::`**Clear**(void) = 0

Clear (free all data) the object.

2.9.9.12.4 Operator

template<typename **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 >*

virtual IndexType2 `rocalution::`*Operator*`::`**GetM**(void) **const** = 0

Return the number of rows in the matrix/stencil.

virtual IndexType2 `rocalution::`*Operator*`::`**GetN**(void) **const** = 0

Return the number of columns in the matrix/stencil.

virtual IndexType2 `rocalution::`*Operator*`::`**GetNnz**(void) **const** = 0

Return the number of non-zeros in the matrix/stencil.

virtual int `rocalution::`*Operator*`::`**GetLocalM**(void) **const**

Return the number of rows in the local matrix/stencil.

virtual int `rocalution::`*Operator*`::`**GetLocalN**(void) **const**

Return the number of columns in the local matrix/stencil.

virtual int `rocalution::`*Operator*`::`**GetLocalNnz**(void) **const**

Return the number of non-zeros in the local matrix/stencil.

virtual int `rocalution::`*Operator*`::`**GetGhostM**(void) **const**

Return the number of rows in the ghost matrix/stencil.

virtual int `rocalution::`*Operator*`::`**GetGhostN**(void) **const**

Return the number of columns in the ghost matrix/stencil.

virtual int `rocalution::`*Operator*`::`**GetGhostNnz**(void) **const**

Return the number of non-zeros in the ghost matrix/stencil.

virtual void `rocalution::`*Operator*`::`**Apply**(**const** *LocalVector*<ValueType> &*in*, *LocalVector*<ValueType> **out*) **const**

Apply the operator, out = Operator(in), where in and out are local vectors.

virtual void `rocalution::`*Operator*`::`**ApplyAdd**(**const** *LocalVector*<ValueType> &*in*, ValueType *scalar*, *LocalVector*<ValueType> **out*) **const**

Apply and add the operator, out += scalar * Operator(in), where in and out are local vectors.

virtual void `rocalution::`*Operator*`::`**Apply**(**const** *GlobalVector*<ValueType> &*in*, *GlobalVector*<ValueType> **out*) **const**

Apply the operator, out = Operator(in), where in and out are global vectors.

virtual void `rocalution::`*Operator*`::`**ApplyAdd**(**const** *GlobalVector*<ValueType> &*in*, ValueType *scalar*, *GlobalVector*<ValueType> **out*) **const**

Apply and add the operator, out += scalar * Operator(in), where in and out are global vectors.

2.9.9.12.5 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 >*

virtual IndexType2 `rocalution::`*Vector*`::`**GetSize**(void) **const** = 0

Return the size of the vector.

virtual int `rocalution::`*Vector*`::`**GetLocalSize**(void) **const**

Return the size of the local vector.

virtual int `rocalution::`*Vector*`::`**GetGhostSize**(void) **const**

Return the size of the ghost vector.

virtual 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.

virtual void `rocalution::`*Vector*`::`**Zeros**(void) = 0

Set all values of the vector to 0.

virtual void `rocalution::`*Vector*`::`**Ones**(void) = 0

Set all values of the vector to 1.

virtual void `rocalution::`*Vector*`::`**SetValues**(ValueType *val*) = 0

Set all values of the vector to given argument.

virtual 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].

virtual 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.

virtual void rocalution::*Vector*::**ReadFileASCII**(**const** std::string *filename*) = 0

Read vector from ASCII file.

Read a vector from ASCII file.

Example

```
LocalVector<ValueType> vec;
vec.ReadFileASCII("my_vector.dat");
```

Parameters

- `[in] filename`: name of the file containing the ASCII data.

virtual void rocalution::*Vector*::**WriteFileASCII**(**const** std::string *filename*) **const** = 0

Write vector to ASCII file.

Write a vector to ASCII file.

Example

```
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.

virtual 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.

virtual 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

```
// Header
out << "#rocALUTION binary vector file" << std::endl;

// rocALUTION version
out.write((char*)&version, sizeof(int));

```

(continues on next page)

(continued from previous page)

```
// 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

```
LocalVector<ValueType> vec;

// Allocate and fill vec
// ...

vec.WriteFileBinary("my_vector.bin");
```

Parameters

- `[in] filename`: name of the file to write the data to.

virtual void `rocalution::`*`Vector`*`::`**`CopyFrom`**`(`**`const`** *LocalVector*<ValueType> &*src*`)`

Copy vector from another vector.

`CopyFrom` copies values from another vector.

Note This function allows cross platform copying. One of the objects could be allocated on the accelerator backend.

Example

```
LocalVector<ValueType> vec1, vec2;

// Allocate and initialize vec1 and vec2
// ...

// Move vec1 to accelerator
// vec1.MoveToAccelerator();

// Now, vec1 is on the accelerator (if available)
// and vec2 is on the host

// Copy vec1 to vec2 (or vice versa) will move data between host and
// accelerator backend
vec1.CopyFrom(vec2);
```

Parameters

- `[in] src`: *Vector*, where values should be copied from.

virtual void `rocalution::`*`Vector`*`::`**`CopyFrom`**`(`**`const`** *GlobalVector*<ValueType> &*src*`)`

Copy vector from another vector.

`CopyFrom` copies values from another vector.

Note This function allows cross platform copying. One of the objects could be allocated on the accelerator backend.

Example

```
LocalVector<ValueType> vec1, vec2;

// Allocate and initialize vec1 and vec2
// ...

// Move vec1 to accelerator
// vec1.MoveToAccelerator();

// Now, vec1 is on the accelerator (if available)
// and vec2 is on the host

// Copy vec1 to vec2 (or vice versa) will move data between host and
// accelerator backend
vec1.CopyFrom(vec2);
```

Parameters

- `[in] src`: *Vector*, where values should be copied from.

virtual void `rocalution::`*`Vector`*`::`**`CopyFromAsync`**(**const** *LocalVector*<ValueType> &*src*)

Async copy from another local vector.

virtual void `rocalution::`*`Vector`*`::`**`CopyFromFloat`**(**const** *LocalVector*<float> &*src*)

Copy values from another local float vector.

virtual void `rocalution::`*`Vector`*`::`**`CopyFromDouble`**(**const** *LocalVector*<double> &*src*)

Copy values from another local double vector.

virtual void `rocalution::`*`Vector`*`::`**`CopyFrom`**(**const** *LocalVector*<ValueType> &*src*, int *src_offset*, int *dst_offset*, int *size*)

Copy vector from another vector with offsets and size.

`CopyFrom` copies values with specific source and destination offsets and sizes from another vector.

Note This function allows cross platform copying. One of the objects could be allocated on the accelerator backend.

Parameters

- `[in] src`: *Vector*, where values should be copied from.
- `[in] src_offset`: source offset.
- `[in] dst_offset`: destination offset.
- `[in] size`: number of entries to be copied.

virtual void `rocalution::`*`Vector`*`::`**`CloneFrom`**(**const** *LocalVector*<ValueType> &*src*)

Clone the vector.

`CloneFrom` clones the entire vector, with data and backend descriptor from another *Vector*.

Example

```
LocalVector<ValueType> vec;

// Allocate and initialize vec (host or accelerator)
// ...

LocalVector<ValueType> tmp;
```

(continues on next page)

(continued from previous page)

```
// By cloning vec, tmp will have identical values and will be on the same
// backend as vec
tmp.CloneFrom(vec);
```

Parameters

- `[in] src`: *Vector* to clone from.

virtual void `rocalution::`*Vector*`::`**CloneFrom**(**const** *GlobalVector*<ValueType> &*src*)

Clone the vector.

`CloneFrom` clones the entire vector, with data and backend descriptor from another *Vector*.

Example

```
LocalVector<ValueType> vec;

// Allocate and initialize vec (host or accelerator)
// ...

LocalVector<ValueType> tmp;

// By cloning vec, tmp will have identical values and will be on the same
// backend as vec
tmp.CloneFrom(vec);
```

Parameters

- `[in] src`: *Vector* to clone from.

virtual void `rocalution::`*Vector*`::`**AddScale**(**const** *LocalVector*<ValueType> &*x*, ValueType *alpha*)

Perform vector update of type this = this + alpha * x.

virtual void `rocalution::`*Vector*`::`**AddScale**(**const** *GlobalVector*<ValueType> &*x*, ValueType *alpha*)

Perform vector update of type this = this + alpha * x.

virtual void `rocalution::`*Vector*`::`**ScaleAdd**(ValueType *alpha*, **const** *LocalVector*<ValueType> &*x*)

Perform vector update of type this = alpha * this + x.

virtual void `rocalution::`*Vector*`::`**ScaleAdd**(ValueType *alpha*, **const** *GlobalVector*<ValueType> &*x*)

Perform vector update of type this = alpha * this + x.

virtual void `rocalution::`*Vector*`::`**ScaleAddScale**(ValueType *alpha*, **const** *LocalVector*<ValueType> &*x*, ValueType *beta*)

Perform vector update of type this = alpha * this + x * beta.

virtual void `rocalution::`*Vector*`::`**ScaleAddScale**(ValueType *alpha*, **const** *GlobalVector*<ValueType> &*x*, ValueType *beta*)

Perform vector update of type this = alpha * this + x * beta.

virtual void `rocalution::`*Vector*`::`**ScaleAddScale**(ValueType *alpha*, **const** *LocalVector*<ValueType> &*x*, ValueType *beta*, int *src_offset*, int *dst_offset*, int *size*)

Perform vector update of type this = alpha * this + x * beta with offsets.

virtual void rocalution::*Vector*::**ScaleAddScale**(ValueType *alpha*, **const** *GlobalVector*<ValueType> &*x*, ValueType *beta*, int *src_offset*, int *dst_offset*, int *size*)

Perform vector update of type this = alpha * this + x * beta with offsets.

virtual void rocalution::*Vector*::**ScaleAdd2**(ValueType *alpha*, **const** *LocalVector*<ValueType> &*x*, ValueType *beta*, **const** *LocalVector*<ValueType> &*y*, ValueType *gamma*)

Perform vector update of type this = alpha * this + x * beta + y * gamma.

virtual void rocalution::*Vector*::**ScaleAdd2**(ValueType *alpha*, **const** *GlobalVector*<ValueType> &*x*, ValueType *beta*, **const** *GlobalVector*<ValueType> &*y*, ValueType *gamma*)

Perform vector update of type this = alpha * this + x * beta + y * gamma.

virtual void rocalution::*Vector*::**Scale**(ValueType *alpha*) = 0

Perform vector scaling this = alpha * this.

virtual ValueType rocalution::*Vector*::**Dot**(**const** *LocalVector*<ValueType> &*x*) **const**

Compute dot (scalar) product, return this^T y.

virtual ValueType rocalution::*Vector*::**Dot**(**const** *GlobalVector*<ValueType> &*x*) **const**

Compute dot (scalar) product, return this^T y.

virtual ValueType rocalution::*Vector*::**DotNonConj**(**const** *LocalVector*<ValueType> &*x*) **const**

Compute non-conjugate dot (scalar) product, return this^T y.

virtual ValueType rocalution::*Vector*::**DotNonConj**(**const** *GlobalVector*<ValueType> &*x*) **const**

Compute non-conjugate dot (scalar) product, return this^T y.

virtual ValueType rocalution::*Vector*::**Norm**(void) **const** = 0

Compute L_2 norm of the vector, return = srqt(this^T this)

virtual ValueType rocalution::*Vector*::**Reduce**(void) **const** = 0

Reduce the vector.

virtual ValueType rocalution::*Vector*::**Asum**(void) **const** = 0

Compute the sum of absolute values of the vector, return = sum(|this|)

virtual int rocalution::*Vector*::**Amax**(ValueType &*value*) **const** = 0

Compute the absolute max of the vector, return = index(max(|this|))

virtual void rocalution::*Vector*::**PointWiseMult**(**const** *LocalVector*<ValueType> &*x*)

Perform point-wise multiplication (element-wise) of this = this * x.

virtual void rocalution::*Vector*::**PointWiseMult**(**const** *GlobalVector*<ValueType> &*x*)

Perform point-wise multiplication (element-wise) of this = this * x.

virtual void rocalution::*Vector*::**PointWiseMult**(**const** *LocalVector*<ValueType> &*x*, **const** *LocalVector*<ValueType> &*y*)

Perform point-wise multiplication (element-wise) of this = x * y.

virtual void rocalution::*Vector*::**PointWiseMult**(**const** *GlobalVector*<ValueType> &*x*, **const** *GlobalVector*<ValueType> &*y*)

Perform point-wise multiplication (element-wise) of this = x * y.

virtual void rocalution::*Vector*::**Power**(double *power*) = 0

Perform power operation to a vector.

2.9.9.12.6 Local Matrix

template<typename **ValueType**>
class LocalMatrix : **public** rocalution::*Operator*<*ValueType*>

LocalMatrix class.

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`**(void) **const**

Return the matrix format id (see matrix_formats.hpp)

bool `rocalution::`*`LocalMatrix`*`::`**`Check`**(void) **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

```
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

```
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`*`::`**`AllocateMCSR`**`(`**`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)

(continued from previous page)

```
mat.Clear();

mat.AllocateCOO("my COO matrix", 200, 100, 100);
mat.Clear();
```

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

```
// 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`*`::`**`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

```
// 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
```

(continues on next page)

(continued from previous page)

```
// invalid
mat.SetDataPtrCSR(&csr_row_ptr, &csr_col, &csr_val, "my_matrix", 345, 100,␣
↪100);
```

void rocalution::*LocalMatrix*::**SetDataPtrMCSR**(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

```
// 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

```
// 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

```
// 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

```
// 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

```
// 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`*`::`**`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

```
// 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`*`::`**`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

```
// rocALUTION CSR matrix object
LocalMatrix<ValueType> mat;

```

(continues on next page)

(continued from previous page)

```
// 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
```

(continues on next page)

(continued from previous page)

```
// ...

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*`::`**`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

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

```
LocalVector<int> conn;

mat.ConnectivityOrder(&conn);
mat.Permute(conn);
```

Parameters

- [out] permutation: permutation vector for connectivity reordering

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

```
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

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

```
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

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

```
LocalVector<int> zbp;
int size;

mat.ZeroBlockPermutation(size, &zbp);
mat.Permute(zbp);
```

Parameters

- [out] size:
- [out] permutation: permutation vector for zero block permutation

void rocalution::*LocalMatrix*::**ILU0Factorize**(void)

Perform ILU(0) factorization.

void rocalution::*LocalMatrix*::**LUFactorize**(void)

Perform *LU* factorization.

void rocalution::*LocalMatrix*::**ILUTFactorize**(double *t*, int *maxrow*)

Perform ILU(t,m) factorization based on threshold and maximum number of elements per row.

void rocalution::*LocalMatrix*::**ILUpFactorize**(int *p*, bool *level* = true)

Perform ILU(p) factorization based on power.

void rocalution::*LocalMatrix*::**LUAnalyse**(void)

Analyse the structure (level-scheduling)

void rocalution::*LocalMatrix*::**LUAnalyseClear**(void)

Delete the analysed data (see LUAnalyse)

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.

void rocalution::*LocalMatrix*::**ICFactorize**(*LocalVector*<ValueType> **inv_diag*)

Perform IC(0) factorization.

void rocalution::*LocalMatrix*::**LLAnalyse**(void)

Analyse the structure (level-scheduling)

void rocalution::*LocalMatrix*::**LLAnalyseClear**(void)

Delete the analysed data (see LLAnalyse)

void rocalution::*LocalMatrix*::**LLSolve**(**const** *LocalVector*<ValueType> &*in*, *LocalVector*<ValueType> **out*) **const**

Solve LL^T out = in; if level-scheduling algorithm is provided then the graph traversing is performed in parallel.

void rocalution::*LocalMatrix*::**LLSolve**(**const** *LocalVector*<ValueType> &*in*, **const** *LocalVector*<ValueType> &*inv_diag*, *LocalVector*<ValueType> **out*) **const**

Solve LL^T out = in; if level-scheduling algorithm is provided then the graph traversing is performed in parallel.

void 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;

void rocalution::*LocalMatrix*::**LAnalyseClear**(void)

Delete the analysed data (see LAnalyse) L-part.

void 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.

void 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;

void rocalution::*LocalMatrix*::**UAnalyseClear**(void)

Delete the analysed data (see UAnalyse) U-part.

void 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.

void rocalution::*LocalMatrix*::**Householder**(int *idx*, ValueType &*beta*, *LocalVector*<ValueType> **vec*) **const**

Compute Householder vector.

void rocalution::*LocalMatrix*::**QRDecompose**(void)

QR Decomposition.

void rocalution::*LocalMatrix*::**QRSolve**(**const** *LocalVector*<ValueType> &*in*, *LocalVector*<ValueType> **out*) **const**

Solve *QR* out = in.

void rocalution::*LocalMatrix*::**Invert**(void)

Matrix inversion using *QR* decomposition.

void rocalution::*LocalMatrix*::**ReadFileMTX**(**const** std::string *filename*)

Read matrix from MTX (Matrix Market Format) file.

Read a matrix from Matrix Market Format file.

Example

```
LocalMatrix<ValueType> mat;
mat.ReadFileMTX("my_matrix.mtx");
```

Parameters

- `[in] filename`: name of the file containing the MTX data.

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

```
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.

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

```
LocalMatrix<ValueType> mat;
mat.ReadFileCSR("my_matrix.csr");
```

Parameters

- `[in] filename`: name of the file containing the data.

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

```
// 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.

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.

void `rocalution::`*`LocalMatrix`*`::`**`CopyFromAsync`**`(`**`const`** LocalMatrix<ValueType> &*src*`)`

Async copy matrix (values and structure) from another *LocalMatrix*.

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

```
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_offsets*, **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] row_offset`: CSR matrix row offset pointers.
- `[in] col`: CSR matrix column indices.
- `[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`*`::`**`CreateFromMap`**(**const** *LocalVector*<int> &*map*, int *n*, int *m*)
Create a restriction matrix operator based on an int vector map.

void `rocalution::`*`LocalMatrix`*`::`**`CreateFromMap`**(**const** *LocalVector*<int> &*map*, int *n*, int *m*, LocalMatrix<ValueType> **pro*)
Create a restriction and prolongation matrix operator based on an int vector map.

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** *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*::**AMGAggregation**(**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.

void rocalution::*LocalMatrix*::**InitialPairwiseAggregation**(ValueType *beta*, int *&nc*, *LocalVector*<int> **G*, int *&Gsize*, int ***rG*, int *&rGsize*, int *ordering*) **const**

Initial Pairwise Aggregation scheme.

void rocalution::*LocalMatrix*::**InitialPairwiseAggregation**(**const** LocalMatrix<ValueType> *&mat*, ValueType *beta*, int *&nc*, *LocalVector*<int> **G*, int *&Gsize*, int ***rG*, int *&rGsize*, int *ordering*) **const**

Initial Pairwise Aggregation scheme for split matrices.

void rocalution::*LocalMatrix*::**FurtherPairwiseAggregation**(ValueType *beta*, int *&nc*, *LocalVector*<int> **G*, int *&Gsize*, int ***rG*, int *&rGsize*, int *ordering*) **const**

Further Pairwise Aggregation scheme.

void rocalution::*LocalMatrix*::**FurtherPairwiseAggregation**(**const** LocalMatrix<ValueType> *&mat*, ValueType *beta*, int *&nc*, *LocalVector*<int> **G*, int *&Gsize*, int ***rG*, int *&rGsize*, int *ordering*) **const**

Further Pairwise Aggregation scheme for split matrices.

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.

2.9.9.12.7 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>

rocalution::*LocalStencil*::**LocalStencil**(unsigned int *type*)

Initialize a local stencil with a type.

int rocalution::*LocalStencil*::**GetNDim**(void) **const**

Return the dimension of the stencil.

void rocalution::*LocalStencil*::**SetGrid**(int *size*)

Set the stencil grid size.

2.9.9.12.8 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>

`rocalution::`*GlobalMatrix*`::`**GlobalMatrix**(**const** *ParallelManager* &*pm*)

Initialize a global matrix with a parallel manager.

virtual 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 strcture 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.

2.9.9.12.9 Local Vector

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>

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

```
LocalVector<ValueType> vec;

vec.Allocate("my vector", 100);
vec.Clear();
```

Parameters

- `[in] name`: object name
- `[in] size`: number of elements in the vector

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

```
// 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);
```

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

```
// rocALUTION local vector object
LocalVector<ValueType> vec;

// Allocate the vector
vec.Allocate("my_vector", 100);

// Fill vector
// ...

ValueType* ptr_vec = NULL;

// Get (steal) the data from the vector, this will leave the local vector␣
↪object empty
vec.LeaveDataPtr(&ptr_vec);
```

ValueType &`rocalution::`*`LocalVector`*`::`**`operator[]`**(int *i*)

Access operator (only for host data)

The elements in the vector can be accessed via [] operators, when the vector is allocated on the host.

Return value at index `i`

Example

```
// rocALUTION local vector object
LocalVector<ValueType> vec;

// Allocate vector
vec.Allocate("my_vector", 100);

// Initialize vector with 1
vec.Ones();

// Set even elements to -1
for(int i = 0; i < vec.GetSize(); i += 2)
{
  vec[i] = -1;
}
```

Parameters

- `[in] i`: access data at index `i`

const ValueType &`rocalution::`*`LocalVector`*`::`**`operator[]`**(int *i*) **const**

Access operator (only for host data)

The elements in the vector can be accessed via [] operators, when the vector is allocated on the host.

Return value at index `i`

Example

```
// rocALUTION local vector object
LocalVector<ValueType> vec;

// Allocate vector
vec.Allocate("my_vector", 100);

// Initialize vector with 1
vec.Ones();

// Set even elements to -1
for(int i = 0; i < vec.GetSize(); i += 2)
{
  vec[i] = -1;
}
```

Parameters

- `[in] i`: access data at index `i`

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.

Copy (import) vector data that is described in one array (values). The object data has to be allocated with *Allocate()*, using the corresponding size of the data, first.

Parameters

- `[in] data`: data to be imported.

void `rocalution::`*LocalVector*`::`**CopyToData**(ValueType **data*) **const**

Copy (export) vector.

Copy (export) vector data that is described in one array (values). The output array has to be allocated, using the corresponding size of the data, first. Size can be obtain by *GetSize()*.

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.

2.9.9.12.10 Global Vector

template<typename **ValueType**>

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>

`rocalution::`*`GlobalVector`*`::`**`GlobalVector`**(**const** *ParallelManager* &*pm*)

Initialize a global vector with a parallel manager.

virtual 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.

ValueType &`rocalution::`*`GlobalVector`*`::`**`operator[]`**(int *i*)

Access operator (only for host data)

const ValueType &`rocalution::`*`GlobalVector`*`::`**`operator[]`**(int *i*) **const**

Access operator (only for host data)

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.

2.9.9.12.11 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.

2.9.9.12.12 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 >*

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()*

virtual void `rocalution::`*`Solver`*`::`**`Print`**(void) **const** = 0
Print information about the solver.

virtual 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** rocalution::*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 $|r_k|_{L_p} \epsilon_{abs}$
- 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 ∞. The norm computation can be set with *SetResidualNorm()* with 1 for L_1, 2 for L_2 and 3 for L_∞. For the computation with L_∞, the index of the maximum value can be obtained with *GetAmaxResidualIndex()*. If this function is called and L_∞ 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 *rocalution::BaseMultiGrid< OperatorType, VectorType, ValueType >*, *rocalution::BiCGStab< OperatorType, VectorType, ValueType >*, *rocalution::BiCGStabl< OperatorType, VectorType, ValueType >*, *rocalution::CG< OperatorType, VectorType, ValueType >*, *rocalution::Chebyshev< OperatorType, VectorType, ValueType >*, *rocalution::CR< OperatorType, VectorType, ValueType >*, *rocalution::FCG< OperatorType, VectorType, ValueType >*, *rocalution::FGMRES< OperatorType, VectorType, ValueType >*, *rocalution::FixedPoint< OperatorType, VectorType, ValueType >*, *rocalution::GMRES< OperatorType, VectorType, ValueType >*, *rocalution::IDR< OperatorType, VectorType, ValueType >*, *rocalution::QMRCGStab< OperatorType, VectorType, ValueType >*

void `rocalution::`*`IterativeLinearSolver`*`::`**`Init`**(double *abs_tol*, double *rel_tol*, double *div_tol*, int *max_iter*)

Initialize the solver with absolute/relative/divergence tolerance and maximum number of iterations.

void `rocalution::`*`IterativeLinearSolver`*`::`**`Init`**(double *abs_tol*, double *rel_tol*, double *div_tol*, int *min_iter*, int *max_iter*)

Initialize the solver with absolute/relative/divergence tolerance and minimum/maximum number of iterations.

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_∞ norm.

- resnorm = 1 -> L_1 norm
- resnorm = 2 -> L_2 norm
- resnorm = 3 -> L_∞ norm

void `rocalution::`*`IterativeLinearSolver`*`::`**`RecordResidualHistory`**(void)

Record the residual history.

void `rocalution::`*`IterativeLinearSolver`*`::`**`RecordHistory`**(**const** std::string *filename*) **const**

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 *Operator* x = rhs.

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_∞ 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

$$x_{k+1} = M^{-1}(b - Nx_k).$$

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 ω.

template<class **OperatorTypeH**, class **VectorTypeH**, typename **ValueTypeH**, class **OperatorTypeL**, class **VectorTypeL**,
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 implemeting 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*
- VectorTypeH: - can be *LocalVector*
- ValueTypeH: - can be double
- OperatorTypeL: - can be *LocalMatrix*
- VectorTypeL: - can be *LocalVector*
- ValueTypeL: - can be float

void rocalution::*MixedPrecisionDC*::**Set**(*Solver*<OperatorTypeL, VectorTypeL, ValueTypeL> &*Solver_L*)

Set the inner solver for $Ad_k = r_k$.

template<class **OperatorType**, class **VectorType**, typename **ValueType**>
class Chebyshev : **public** rocalution::*IterativeLinearSolver*<*OperatorType*, *VectorType*, *ValueType*>

Chebyshev Iteration Scheme.

The *Chebyshev* Iteration scheme (also known as acceleration scheme) is similar to the *CG* method but requires minimum and maximum eigenvalues of the operator. templates

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::`*`Chebyshev`*`::`**`Set`**(ValueType *lambda_min*, ValueType *lambda_max*)

Set the minimum and maximum eigenvalues of the operator.

template<class **OperatorType**, class **VectorType**, typename **ValueType**>
class BiCGStab : **public** rocalution::*IterativeLinearSolver*<*OperatorType*, *VectorType*, *ValueType*>

Bi-Conjugate Gradient Stabilized Method.

The Bi-Conjugate Gradient Stabilized method is a variation of CGS and solves sparse (non) symmetric linear systems $Ax = b$. 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 BiCGStabl : **public** rocalution::*IterativeLinearSolver*<*OperatorType*, *VectorType*, *ValueType*>

Bi-Conjugate Gradient Stabilized (l) Method.

The Bi-Conjugate Gradient Stabilized (l) method is a generalization of *BiCGStab* for solving sparse (non) symmetric linear systems $Ax = b$. It minimizes residuals over l-dimensional Krylov subspaces. The degree l can be set with *SetOrder()*. bicgstabl

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>

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 $\mathcal{K}_m(r_0, A)$, 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 *SetBasisSize()*. The default size is 30.

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::`*GMRES*`::`**`SetBasisSize`**(int *size_basis*)

Set the size of the Krylov subspace basis.

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 *SetBasisSize()*. The default size is 30.

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::`*FGMRES*`::`**`SetBasisSize`**(int *size_basis*)

Set the size of the Krylov subspace basis.

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 *SetShadowSpace()*. The default size of the shadow space is 4.

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::`*IDR*`::`**`SetShadowSpace`**(int *s*)

Set the size of the Shadow Space.

void `rocalution::`*IDR*`::`**`SetRandomSeed`**(unsigned long long *seed*)

Set random seed for ONB creation (seed must be greater than 0)

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`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, ValueType >*, *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.

virtual void `rocalution::`*`BaseMultiGrid`*`::`**`SetRestrictOperator`**(OperatorType ***op*) = 0

Set the restriction operator for each level.

virtual void `rocalution::`*`BaseMultiGrid`*`::`**`SetProlongOperator`**(OperatorType ***op*) = 0

Set the prolongation operator for each level.

virtual 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 `rocalution::`*`SAAMG`*`::`**`SetInterpRelax`**(ValueType *relax*)

Set the relaxation parameter.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class RugeStuebenAMG` : **`public`** rocalution::*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. 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::`*`RugeStuebenAMG`*`::`**`SetCouplingStrength`**(ValueType *eps*)
Set coupling strength.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class PairwiseAMG` : **`public`** rocalution::*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. pairwiseamg

Template Parameters

- `OperatorType`: - can be *LocalMatrix*
- `VectorType`: - can be *LocalVector*
- `ValueType`: - can be float, double, std::complex<float> or std::complex<double>

void `rocalution::`*`PairwiseAMG`*`::`**`SetBeta`**(ValueType *beta*)
Set beta for pairwise aggregation.

void `rocalution::`*`PairwiseAMG`*`::`**`SetOrdering`**(unsigned int *ordering*)
Set re-ordering for aggregation.

void `rocalution::`*`PairwiseAMG`*`::`**`SetCoarseningFactor`**(double *factor*)
Set target coarsening factor.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class GlobalPairwiseAMG` : **`public`** rocalution::*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. pairwiseamg

Template Parameters

- `OperatorType`: - can be *GlobalMatrix*
- `VectorType`: - can be *GlobalVector*
- `ValueType`: - can be float, double, std::complex<float> or std::complex<double>

void `rocalution::`*`GlobalPairwiseAMG`*`::`**`SetBeta`**(ValueType *beta*)
Set beta for pairwise aggregation.

void `rocalution::`*`GlobalPairwiseAMG`*`::`**`SetOrdering`**(**`const`** _aggregation_ordering *ordering*)
Set re-ordering for aggregation.

void `rocalution::`*`GlobalPairwiseAMG`*`::`**`SetCoarseningFactor`**(double *factor*)
Set target coarsening factor.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>

class DirectLinearSolver : **public** rocalution::*Solver*<*OperatorType*, *VectorType*, *ValueType*>

Base class for all direct linear solvers.

The library provides three direct methods - *LU*, *QR* and *Inversion* (based on *QR* decomposition). The user can pass a sparse matrix, internally it will be converted to dense and then the selected method will be applied. These methods are not very optimal and due to the fact that the matrix is converted to a dense format, these methods should be used only for very small matrices.

Template Parameters

- `OperatorType`: - can be *LocalMatrix*
- `VectorType`: - can be *LocalVector*
- `ValueType`: - can be float, double, std::complex<float> or std::complex<double>

Subclassed by *rocalution::Inversion< OperatorType, VectorType, ValueType >*, *rocalution::LU< OperatorType, VectorType, ValueType >*, *rocalution::QR< OperatorType, VectorType, ValueType >*

template<class **OperatorType**, class **VectorType**, typename **ValueType**>
class Inversion : **public** rocalution::*DirectLinearSolver*<*OperatorType*, *VectorType*, *ValueType*>

Matrix *Inversion*.

Full matrix inversion based on *QR* decomposition.

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 LU : **public** rocalution::*DirectLinearSolver*<*OperatorType*, *VectorType*, *ValueType*>

LU Decomposition.

Lower-Upper Decomposition factors a given square matrix into lower and upper triangular matrix, such that $A = LU$.

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 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>

2.9.9.12.13 Preconditioners

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::GS< OperatorType, VectorType, ValueType >*, *rocalution::IC< OperatorType, VectorType, ValueType >*, *rocalution::ILU< OperatorType, VectorType, ValueType >*, *rocalution::ILUT< OperatorType, VectorType, ValueType >*, *rocalution::Jacobi< OperatorType, VectorType, ValueType >*, *rocalution::MultiColored< OperatorType, VectorType, ValueType >*, *rocalution::MultiElimination< OperatorType, VectorType, ValueType >*, *rocalution::SGS< OperatorType, VectorType, ValueType >*, *rocalution::SPAI< OperatorType, VectorType, ValueType >*, *rocalution::TNS< OperatorType, VectorType, ValueType >*, *rocalution::VariablePreconditioner< OperatorType, VectorType, ValueType >*

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. chebpoly

Template Parameters

- `OperatorType`: - can be *LocalMatrix*
- `VectorType`: - can be *LocalVector*
- `ValueType`: - can be float, double, std::complex<float> or std::complex<double>

void `rocalution::`*`AIChebyshev`*`::`**`Set`**(int *p*, ValueType *lambda_min*, ValueType *lambda_max*)

Set order, min and max eigenvalues.

template<class **OperatorType**, class **VectorType**, typename **ValueType**>
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>

void `rocalution::`*`FSAI`*`::`**`Set`**`(`int *power*`)`

Set the power of the system matrix sparsity pattern.

void `rocalution::`*`FSAI`*`::`**`Set`**`(`**`const`** OperatorType &*pattern*`)`

Set an external sparsity pattern.

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`** rocalution::*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 from, 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 `rocalution::`*`TNS`*`::`**`Set`**`(`bool *imp*`)`

Set implicit (true) or explicit (false) computation.

void `rocalution::`*`TNS`*`::`**`SetPrecondMatrixFormat`**`(`unsigned int *mat_format*`)`

Set the matrix format of the preconditioner.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class AS` : **`public`** rocalution::*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 $A_i = R_i^T AR_i$, where R_i are restriction operators. Those restriction operators produce sub-matrices wich 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>

Subclassed by *rocalution::RAS< OperatorType, VectorType, ValueType >*

void `rocalution::`*`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`** rocalution::*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 $A_i = R_i^T AR_i$, 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>

void `rocalution::`*`BlockJacobi`*`::`**`Set`**`(`*Solver*<*LocalMatrix*<ValueType>, *LocalVector*<ValueType>, ValueType> &*precond*`)`

Set local preconditioner.

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

$$P = \begin{pmatrix} A_d & 0 & . & 0 \\ B_1 & B_d & . & 0 \\ . & . & . & . \\ Z_1 & Z_2 & . & Z_d \end{pmatrix}$$

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 *SetLSolver()* (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>

void `rocalution::`*`BlockPreconditioner`*`::`**`Set`**(int *n*, **`const`** int **size*, *Solver*<OperatorType, VectorType, ValueType> ***D_solver*)

Set number, size and diagonal solver.

void `rocalution::`*`BlockPreconditioner`*`::`**`SetDiagonalSolver`**(void)

Set diagonal solver mode.

void `rocalution::`*`BlockPreconditioner`*`::`**`SetLSolver`**(void)

Set lower triangular sweep mode.

void `rocalution::`*`BlockPreconditioner`*`::`**`SetExternalLastMatrix`**(**`const`** OperatorType &*mat*)

Set external last block matrix.

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*>

Symmetric Gauss-Seidel / Symmetric Successive Over-Relaxation Method.

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 $A = LU - R$.

Template Parameters

- `OperatorType`: - can be *LocalMatrix*
- `VectorType`: - can be *LocalVector*
- `ValueType`: - can be float, double, std::complex<float> or std::complex<double>

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)

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>

void `rocalution::`*`ILUT`*`::`**`Set`**(double *t*)

Set drop-off threshold.

void `rocalution::`*`ILUT`*`::`**`Set`**(double *t*, int *maxrow*)

Set drop-off threshold and maximum fill-ins per row.

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 *LocalVector*
- `ValueType`: - can be float, double, std::complex<float> or std::complex<double>

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class VariablePreconditioner` : **`public`** rocalution::*Preconditioner*<*OperatorType*, *VectorType*, *ValueType*>

Variable *Preconditioner*.

The 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 `rocalution::`*`VariablePreconditioner`*`::`**`SetPreconditioner`**(int *n*, *Solver*<OperatorType, VectorType, ValueType> ***precond*)

Set the preconditioner sequence.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class MultiColored` : **`public`** rocalution::*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 *rocalution::MultiColoredILU< OperatorType, VectorType, ValueType >*, *rocalution::MultiColoredSGS< OperatorType, VectorType, ValueType >*

void `rocalution::`*`MultiColored`*`::`**`SetPrecondMatrixFormat`**(unsigned int *mat_format*)

Set a specific matrix type of the decomposed block matrices.

void `rocalution::`*`MultiColored`*`::`**`SetDecomposition`**(bool *decomp*)

Set if the preconditioner should be decomposed or not.

template<class **`OperatorType`**, class **`VectorType`**, typename **`ValueType`**>
`class MultiColoredSGS` : **`public`** rocalution::*MultiColored*<*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*
- `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>

void rocalution::*MultiColoredILU*::**Set**(int *p*)

Initialize a multi-colored *ILU*(p, p+1) preconditioner.

void rocalution::*MultiColoredILU*::**Set**(int *p*, int *q*, bool *level* = true)

Initialize a multi-colored ILU(p, q) preconditioner.

level = true will perform the factorization with levels level = false will perform the factorization only on the power(q)-pattern

template<class **OperatorType**, class **VectorType**, typename **ValueType**>
class MultiElimination : **public** rocalution::*Preconditioner*<*OperatorType*, *VectorType*, *ValueType*>

Multi-Elimination Incomplete *LU* Factorization *Preconditioner*.

The Multi-Elimination Incomplete *LU* preconditioner is based on the following decomposition

$$A = \begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ ED^{-1} & I \end{pmatrix} \times \begin{pmatrix} D & F \\ 0 & \hat{A} \end{pmatrix},$$

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 rocalution::*MultiElimination*::**GetSizeDiagBlock**(void) **const**

Returns the size of the first (diagonal) block of the preconditioner.

int rocalution::*MultiElimination*::**GetLevel**(void) **const**

Return the depth of the current level.

void rocalution::*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 rocalution::*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** rocalution::*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 `rocalution::`*`DiagJacobiSaddlePointPrecond`*`::`**`Set`** (*Solver*<OperatorType, VectorType, ValueType> &*K_Solver*, *Solver*<OperatorType, VectorType, ValueType> &*S_Solver*)

Initialize solver for K and S.

2.9.10 rocSPARSE

2.9.10.1 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.
- 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

2.9.10.2 Device and Stream Management

hipSetDevice() and *hipGetDevice()* are HIP device management APIs. They are NOT part of the rocSPARSE API.

2.9.10.2.1 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.

2.9.10.2.2 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.

2.9.10.2.3 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.

2.9.10.2.4 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.

2.9.10.3 Building and Installing

2.9.10.3.1 Installing from AMD ROCm repositories

rocSPARSE can be installed from AMD ROCm repositories by

```
sudo apt install rocsparse
```

2.9.10.3.2 Building rocSPARSE from Open-Source repository

Download rocSPARSE

The rocSPARSE source code is available at the rocSPARSE github page. Download the master branch using:

```
git clone -b master https://github.com/ROCmSoftwarePlatform/rocSPARSE.git
cd rocSPARSE
```

Note that if you want to contribute to rocSPARSE, you will need to checkout the develop branch instead of the master branch.

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.

2.9.10.3.3 Using *install.sh* to build dependencies + library

The following table lists common uses of *install.sh* to build dependencies + library.

2.9.10.3.4 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.

Command	Description
./install.sh -h	Print help information.
./install.sh -dc	Build dependencies, library and client in your local directory. The *-d* flag only needs to be **|br|** used once. For subsequent invocations of *install.sh* it is not necessary to rebuild the **|br|** dependencies.
./install.sh -c	Build library and client in your local directory. It is assumed dependencies are available.
./install.sh -idc	Build library, dependencies and client, then build and install rocSPARSE package in **|br|** */opt/rocm/rocsparse*. You will be prompted for sudo access. This will install for all users.
./install.sh -ic	Build library and client, then build and install rocSPARSE package in *opt/rocm/rocsparse*. **|br|** You will be prompted for sudo access. This will install for all users.

2.9.10.3.5 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:

```
# 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 client.

rocSPARSE with dependencies and client can be built using the following commands:

```
# Install boost on Ubuntu
sudo apt install libboost-program-options-dev
# Install boost on Fedora
```

(continues on next page)

(continued from previous page)

```
sudo dnf install boost-program-options

# Install googletest
mkdir -p build/release/deps ; cd build/release/deps
cmake -DBUILD_BOOST=OFF ../../../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
```

2.9.10.3.6 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:** For Carrizo - HCC RUNTIME ERROR: Failed to find compatible kernel

   Solution: Add the following to the cmake command when configuring: *-DCMAKE_CXX_FLAGS=”–amdgpu-target=gfx801”*

3. **Issue:** For MI25 (Vega10 Server) - HCC RUNTIME ERROR: Failed to find compatible kernel

   Solution: *export HCC_AMDGPU_TARGET=gfx900*

4. **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

2.9.10.4 Unit tests

To run unit tests, rocSPARSE has to be built with option -DBUILD_CLIENTS_TESTS=ON.

```
# Go to rocSPARSE build directory
cd rocSPARSE; cd build/release

# Run all tests
./clients/staging/rocsparse-test
```

2.9.10.5 Benchmarks

To run benchmarks, rocSPARSE has to be built with option -DBUILD_CLIENTS_BENCHMARKS=ON.

```
# Go to rocSPARSE build directory
cd rocSPARSE/build/release

# Run benchmark, e.g.
./clients/staging/rocsparse-bench -f hybmv --laplacian-dim 2000 -i 200
```

2.9.10.6 Storage Formats

2.9.10.6.1 COO storage format

The Coordinate (COO) storage format represents a $m \times n$ matrix by

m	number of rows (integer).
n	number of columns (integer).
nnz	number of non-zero elements (integer).
coo_val	array of `nnz` elements containing the data (floating point).
coo_row_ind	array of `nnz` elements containing the row indices (integer).
coo_col_ind	array of `nnz` elements containing the column indices (integer).

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 3×5 matrix and the corresponding COO structures, with $m = 3, n = 5$ and $\text{nnz} = 8$ 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

$$\begin{array}{ll} coo\_val[8] & = \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0\} \\ coo\_row\_ind[8] & = \{0, 0, 0, 1, 1, 2, 2, 2\} \\ coo\_col\_ind[8] & = \{0, 1, 3, 1, 2, 0, 3, 4\} \end{array}$$

2.9.10.6.2 CSR storage format

The Compressed Sparse Row (CSR) storage format represents a $m \times n$ matrix by

m	number of rows (integer).
n	number of columns (integer).
nnz	number of non-zero elements (integer).
csr_val	array of `nnz` elements containing the data (floating point).
csr_row_ptr	array of `m+1` elements that point to the start of every row (integer).
csr_col_ind	array of `nnz` elements containing the column indices (integer).

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 3×5 matrix and the corresponding CSR structures, with $m = 3, n = 5$ and $\text{nnz} = 8$ 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

$$\begin{array}{ll} csr\_val[8] & = \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0\} \\ csr\_row\_ptr[4] & = \{1, 4, 6, 9\} \\ csr\_col\_ind[8] & = \{1, 2, 4, 2, 3, 1, 4, 5\} \end{array}$$

2.9.10.6.3 ELL storage format

The Ellpack-Itpack (ELL) storage format represents a $m \times n$ matrix by

m	number of rows (integer).
n	number of columns (integer).
ell_width	maximum number of non-zero elements per row (integer)
ell_val	array of `m times ell_width` elements containing the data (floating point).
ell_col_ind	array of `m times ell_width` elements containing the column indices (integer).

The ELL matrix is assumed to be stored in column-major format. Rows with less than `ell_width` non-zero elements are padded with zeros (`ell_val`) and -1 (`ell_col_ind`). Consider the following 3×5 matrix and the corresponding ELL structures, with $m = 3, n = 5$ and

m	number of rows (integer).
n	number of columns (integer).
nnz	number of non-zero elements of the COO part (integer)
ell_width	maximum number of non-zero elements per row of the ELL part (integer)
ell_val	array of `m times ell_width` elements containing the ELL part data (floating point).
ell_col_ind	array of `m times ell_width` elements containing the ELL part column indices (integer).
coo_val	array of `nnz` elements containing the COO part data (floating point).
coo_row_ind	array of `nnz` elements containing the COO part row indices (integer).
coo_col_ind	array of `nnz` elements containing the COO part column indices (integer).

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_.

2.9.10.7 Types

2.9.10.7.1 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()*.

2.9.10.7.2 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()*.

2.9.10.7.3 rocsparse_mat_info

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()*.

2.9.10.7.4 rocsparse_hyb_mat

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()*.

2.9.10.7.5 rocsparse_action

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:

rocsparse_action_symbolic = 0

Operate only on indices.

rocsparse_action_numeric = 1
Operate on data and indices.

2.9.10.7.6 rocsparse_hyb_partition

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:

rocsparse_hyb_partition_auto = 0
automatically decide on ELL nnz per row.

rocsparse_hyb_partition_user = 1
user given ELL nnz per row.

rocsparse_hyb_partition_max = 2
max ELL nnz per row, no COO part.

2.9.10.7.7 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:

rocsparse_index_base_zero = 0
zero based indexing.

rocsparse_index_base_one = 1
one based indexing.

2.9.10.7.8 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:

rocsparse_matrix_type_general = 0
general matrix type.

rocsparse_matrix_type_symmetric = 1
symmetric matrix type.

rocsparse_matrix_type_hermitian = 2
hermitian matrix type.

rocsparse_matrix_type_triangular = 3
triangular matrix type.

2.9.10.7.9 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:

rocsparse_fill_mode_lower = 0
lower triangular part is stored.

rocsparse_fill_mode_upper = 1
upper triangular part is stored.

2.9.10.7.10 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:

rocsparse_diag_type_non_unit = 0
diagonal entries are non-unity.

rocsparse_diag_type_unit = 1
diagonal entries are unity

2.9.10.7.11 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:

rocsparse_operation_none = 111
Operate with matrix.

rocsparse_operation_transpose = 112
Operate with transpose.

rocsparse_operation_conjugate_transpose = 113
Operate with conj. transpose.

2.9.10.7.12 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:

rocsparse_pointer_mode_host = 0

scalar pointers are in host memory.

rocsparse_pointer_mode_device = 1

scalar pointers are in device memory.

2.9.10.7.13 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:

rocsparse_analysis_policy_reuse = 0

try to re-use meta data.

rocsparse_analysis_policy_force = 1

force to re-build meta data.

2.9.10.7.14 rocsparse_solve_policy

enum rocsparse_solve_policy

Specify policy in triangular solvers and factorizations.

This is a placeholder.

Values:

rocsparse_solve_policy_auto = 0

automatically decide on level information.

2.9.10.7.15 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:

rocsparse_layer_mode_none = 0x0

layer is not active.

rocsparse_layer_mode_log_trace = 0x1
layer is in logging mode.

rocsparse_layer_mode_log_bench = 0x2
layer is in benchmarking mode.

2.9.10.7.16 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:

rocsparse_status_success = 0
success.

rocsparse_status_invalid_handle = 1
handle not initialized, invalid or null.

rocsparse_status_not_implemented = 2
function is not implemented.

rocsparse_status_invalid_pointer = 3
invalid pointer parameter.

rocsparse_status_invalid_size = 4
invalid size parameter.

rocsparse_status_memory_error = 5
failed memory allocation, copy, dealloc.

rocsparse_status_internal_error = 6
other internal library failure.

rocsparse_status_invalid_value = 7
invalid value parameter.

rocsparse_status_arch_mismatch = 8
device arch is not supported.

rocsparse_status_zero_pivot = 9
encountered zero pivot.

2.9.10.8 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:

`ROCSPARSE_LAYER` unset	logging is disabled.
`ROCSPARSE_LAYER` set to `1`	trace logging is enabled.
`ROCSPARSE_LAYER` set to `2`	bench logging is enabled.
`ROCSPARSE_LAYER` set to `3`	trace logging and bench logging is enabled.

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.

2.9.10.9 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.

2.9.10.9.1 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.

2.9.10.9.2 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.

2.9.10.9.3 rocsparse_set_stream()

rocsparse_status **rocsparse_set_stream**(*rocsparse_handle* *handle*, hipStream_t *stream*)

Specify user defined HIP stream.

`rocsparse_set_stream` specifies the stream to be used by the rocSPARSE library context and all subsequent function calls.

Example This example illustrates, how a user defined stream can be used in rocSPARSE.

```
// 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);
```

Parameters

- `[inout] handle`: the handle to the rocSPARSE library context.
- `[in] stream`: the stream to be used by the rocSPARSE library context.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: `handle` is invalid.

2.9.10.9.4 rocsparse_get_stream()

rocsparse_status **rocsparse_get_stream**(*rocsparse_handle* *handle*, hipStream_t **stream*)

Get current stream from library context.

`rocsparse_get_stream` gets the rocSPARSE library context stream which is currently used for all subsequent function calls.

Parameters

- `[in] handle`: the handle to the rocSPARSE library context.
- `[out] stream`: the stream currently used by the rocSPARSE library context.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_handle`: `handle` is invalid.

2.9.10.9.5 rocsparse_set_pointer_mode()

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.

2.9.10.9.6 rocsparse_get_pointer_mode()

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.

2.9.10.9.7 rocsparse_get_version()

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.

2.9.10.9.8 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.

2.9.10.9.9 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.

2.9.10.9.10 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

- `[in] descr`: the matrix descriptor.

Return Value

- `rocsparse_status_success`: the operation completed successfully.

- `rocsparse_status_invalid_pointer`: `descr` is invalid.

2.9.10.9.11 rocsparse_copy_mat_descr()

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.

2.9.10.9.12 rocsparse_set_mat_index_base()

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.

2.9.10.9.13 rocsparse_get_mat_index_base()

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

- `[in] descr`: the matrix descriptor.

2.9.10.9.14 rocsparse_set_mat_type()

rocsparse_status **rocsparse_set_mat_type**(*rocsparse_mat_descr descr*, *rocsparse_matrix_type type*)

Specify the matrix type of a matrix descriptor.

`rocsparse_set_mat_type` sets the matrix type of a matrix descriptor. Valid matrix types are *rocsparse_matrix_type_general*, *rocsparse_matrix_type_symmetric*, *rocsparse_matrix_type_hermitian* or *rocsparse_matrix_type_triangular*.

Parameters

- `[inout] descr`: the matrix descriptor.
- `[in] type`: *rocsparse_matrix_type_general*, *rocsparse_matrix_type_symmetric*, *rocsparse_matrix_type_hermitian* or *rocsparse_matrix_type_triangular*.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_pointer`: `descr` pointer is invalid.
- `rocsparse_status_invalid_value`: `type` is invalid.

2.9.10.9.15 rocsparse_get_mat_type()

rocsparse_matrix_type **rocsparse_get_mat_type**(**const** *rocsparse_mat_descr descr*)

Get the matrix type of a matrix descriptor.

`rocsparse_get_mat_type` returns the matrix type of a matrix descriptor.

Return *rocsparse_matrix_type_general*, *rocsparse_matrix_type_symmetric*, *rocsparse_matrix_type_hermitian* or *rocsparse_matrix_type_triangular*.

Parameters

- `[in] descr`: the matrix descriptor.

2.9.10.9.16 rocsparse_set_mat_fill_mode()

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.

`rocsparse_set_mat_fill_mode` sets the matrix fill mode of a matrix descriptor. Valid fill modes are *rocsparse_fill_mode_lower* or *rocsparse_fill_mode_upper*.

Parameters

- `[inout] descr`: the matrix descriptor.
- `[in] fill_mode`: *rocsparse_fill_mode_lower* or *rocsparse_fill_mode_upper*.

Return Value

- `rocsparse_status_success`: the operation completed successfully.
- `rocsparse_status_invalid_pointer`: `descr` pointer is invalid.
- `rocsparse_status_invalid_value`: `fill_mode` is invalid.

2.9.10.9.17 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.

2.9.10.9.18 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.

2.9.10.9.19 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.

2.9.10.9.20 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.

2.9.10.9.21 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.

2.9.10.9.22 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.

2.9.10.9.23 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

- [in] info: the info structure.

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.

2.9.10.10 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.

2.9.10.10.1 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 α and adds the result to the dense vector y, such that

$$y := y + \alpha \cdot x$$

```
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 α.
- `[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.

2.9.10.10.2 rocsparse_doti()

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

$$\text{result} := y^T x$$

```
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.

2.9.10.10.3 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.

`rocsparse_gthr` gathers the elements that are listed in `x_ind` from the dense vector y and stores them in the sparse vector x.

```
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.

2.9.10.10.4 rocsparse_gthrz()

rocsparse_status **rocsparse_sgthrz**(*rocsparse_handle handle*, rocsparse_int *nnz*, float **y*, float **x_val*, **const** rocsparse_int **x_ind*, *rocsparse_index_base idx_base*)

rocsparse_status **rocsparse_dgthrz**(*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.

`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.

```
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.

2.9.10.10.5 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.

2.9.10.10.6 rocsparse_sctr()

rocsparse_status **rocsparse_ssctr**(*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_dsctr**(*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.

```
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.

2.9.10.11 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.

2.9.10.11.1 rocsparse_coomv()

rocsparse_status **rocsparse_scoomv** (*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 **coo_val*, **const** rocsparse_int **coo_row_ind*, **const** rocsparse_int **coo_col_ind*, **const** float **x*, **const** float **beta*, float **y*)

rocsparse_status **rocsparse_dcoomv** (*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 **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.

`rocsparse_coomv` multiplies the scalar α 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 β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & if\ trans == rocsparse\_operation\_none \\ A^T, & if\ trans == rocsparse\_operation\_transpose \\ A^H, & if\ trans == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

The COO matrix has to be sorted by row indices. This can be achieved by using *rocsparse_coosort_by_row()*.

```
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] trans`: matrix operation type.
- `[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 α.
- `[in] descr`: descriptor of the sparse COO matrix. Currently, only *rocsparse_matrix_type_general* is supported.
- `[in] coo_val`: array of `nnz` elements 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.
- `[in] x`: array of `n` elements ($op(A) = A$) or `m` elements ($op(A) = A^T$ or $op(A) = A^H$).
- `[in] beta`: scalar β.
- `[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` != *rocsparse_operation_none* or *rocsparse_matrix_type* != *rocsparse_matrix_type_general*.

2.9.10.11.2 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*)

rocsparse_status **rocsparse_dcsrmv_analysis**(*rocsparse_handle* *handle*, *rocsparse_operation* *trans*, rocsparse_int *m*, rocsparse_int *n*, 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*)

Sparse matrix vector multiplication using CSR storage format.

`rocsparse_csrmv_analysis` performs the analysis step for *rocsparse_scsrmv()* and *rocsparse_dcsrmv()*. 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 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] 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

- `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*.

2.9.10.11.3 rocsparse_csrmv()

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*)

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 α 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 β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & if\ trans == rocsparse\_operation\_none \\ A^T, & if\ trans == rocsparse\_operation\_transpose \\ A^H, & if\ trans == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

The `info` parameter is optional and contains information collected by *rocsparse_scsrmv_analysis()* or *rocsparse_dcsrmv_analysis()*. If present, the information will be used to speed up the `csrmv` computation. If `info` == `NULL`, general `csrmv` routine will be used instead.

```
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.

```
// 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,
```

(continues on next page)

(continued from previous page)

```
                 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 α.
- `[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()* 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 β.
- `[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*.

2.9.10.11.4 rocsparse_csrmv_analysis_clear()

rocsparse_status **rocsparse_csrmv_clear**(*rocsparse_handle handle*, *rocsparse_mat_info info*)

Sparse matrix vector multiplication using CSR storage format.

`rocsparse_csrmv_clear` deallocates all memory that was allocated by *rocsparse_scsrmv_analysis()* or *rocsparse_dcsrmv_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_csrmv_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.

2.9.10.11.5 rocsparse_ellmv()

rocsparse_status **rocsparse_sellmv**(*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_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 α 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 β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & if\ trans == rocsparse\_operation\_none \\ A^T, & if\ trans == rocsparse\_operation\_transpose \\ A^H, & if\ trans == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

```
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` == *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 α.
- `[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 β.
- `[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*.

2.9.10.11.6 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 α 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 β, such that

$$y := \alpha \cdot op(A) \cdot x + \beta \cdot y,$$

with

$$op(A) = \begin{cases} A, & if\ trans == rocsparse\_operation\_none \\ A^T, & if\ trans == rocsparse\_operation\_transpose \\ A^H, & if\ trans == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

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 `trans` == *rocsparse_operation_none* is supported.

Parameters

- `[in] handle`: handle to the rocsparse library context queue.
- `[in] trans`: matrix operation type.
- `[in] alpha`: scalar α.
- `[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 β.
- `[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`: `trans` != *rocsparse_operation_none* or *rocsparse_matrix_type* != *rocsparse_matrix_type_general*.

2.9.10.11.7 rocsparse_csrsv_zero_pivot()

rocsparse_status **rocsparse_csrsv_zero_pivot**(*rocsparse_handle* *handle*, **const** *rocsparse_mat_descr* *descr*, *rocsparse_mat_info* *info*, rocsparse_int **position*)

Sparse triangular solve using CSR storage format.

`rocsparse_csrsv_zero_pivot` returns *rocsparse_status_zero_pivot*, if either a structural or numerical zero has been found during *rocsparse_scsrsv_solve()* or *rocsparse_dcsrsv_solve()* 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_csrsv_zero_pivot` is a blocking function. It might influence performance negatively.

Parameters

- `[in] handle`: handle to the rocsparse library context queue.
- `[in] descr`: descriptor of the sparse CSR matrix.
- `[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.

2.9.10.11.8 rocsparse_csrsv_buffer_size()

rocsparse_status **rocsparse_scsrsv_buffer_size**(*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*)

rocsparse_status **rocsparse_dcsrsv_buffer_size**(*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*)

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_scsrsv_solve()* and *rocsparse_dcsrsv_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()* and *rocsparse_dcsrilu0_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] 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] buffer_size`: number of bytes of the temporary storage buffer required by *rocsparse_scsrsv_analysis()*, *rocsparse_dcsrsv_analysis()*, *rocsparse_scsrsv_solve()* and *rocsparse_dcsrsv_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*.

2.9.10.11.9 rocsparse_csrsv_analysis()

rocsparse_status **`rocsparse_scsrsv_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()* and *rocsparse_dcsrsv_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()* and *rocsparse_dcsrilu0_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] 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_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*.

2.9.10.11.10 rocsparse_csrsv_solve()

rocsparse_status **rocsparse_scsrsv_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 α, such that

$$op(A) \cdot y = \alpha \cdot x,$$

with

$$op(A) = \begin{cases} A, & if\ trans == rocsparse\_operation\_none \\ A^T, & if\ trans == rocsparse\_operation\_transpose \\ A^H, & if\ trans == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

`rocsparse_csrsv_solve` requires a user allocated temporary buffer. Its size is returned by *rocsparse_scsrsv_buffer_size()* or *rocsparse_dcsrsv_buffer_size()*. Furthermore, analysis meta data is required. It can be obtained by *rocsparse_scsrsv_analysis()* or *rocsparse_dcsrsv_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 `trans` == *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 $L \cdot y = x$.

```
// 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);

```

(continues on next page)

(continued from previous page)

```
// 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

// Clean up
hipFree(temp_buffer);
rocsparse_destroy_mat_info(info);
rocsparse_destroy_mat_descr(descr);
```

(continues on next page)

(continued from previous page)

```
rocsparse_destroy_handle(handle);
```

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] alpha`: scalar α.
- `[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`: *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`, `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*.

2.9.10.11.11 rocsparse_csrsv_clear()

rocsparse_status **rocsparse_csrsv_clear**(*rocsparse_handle handle*, **const** *rocsparse_mat_descr 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()* or *rocsparse_dcsrsv_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.
- `[in] descr`: descriptor of the sparse CSR matrix.
- `[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.

2.9.10.12 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.

2.9.10.12.1 rocsparse_csrmm()

rocsparse_status **rocsparse_scsrmm**(*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*)

rocsparse_status **rocsparse_dcsrmm**(*rocsparse_handle handle*, *rocsparse_operation trans_A*, *rocsparse_operation trans_B*, rocsparse_int *m*, rocsparse_int *n*, rocsparse_int *k*, 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*, **const** double **B*, rocsparse_int *ldb*, **const** double **beta*, double **C*, rocsparse_int *ldc*)

Sparse matrix dense matrix multiplication using CSR storage format.

`rocsparse_csrmm` multiplies the scalar α 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 β, such that

$$C := \alpha \cdot op(A) \cdot op(B) + \beta \cdot C,$$

with

$$op(A) = \begin{cases} A, & if\ trans\_A == rocsparse\_operation\_none \\ A^T, & if\ trans\_A == rocsparse\_operation\_transpose \\ A^H, & if\ trans\_A == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

and

$$op(B) = \begin{cases} B, & if\ trans\_B == rocsparse\_operation\_none \\ B^T, & if\ trans\_B == rocsparse\_operation\_transpose \\ B^H, & if\ trans\_B == rocsparse\_operation\_conjugate\_transpose \end{cases}$$

```
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
```

(continues on next page)

(continued from previous page)

```
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 α.
- `[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 β.
- `[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*.

2.9.10.13 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.

2.9.10.13.1 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()* or *rocsparse_dcsrilu0()* 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.

2.9.10.13.2 rocsparse_csrilu0_buffer_size()

rocsparse_status **rocsparse_scsrilu0_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*)

rocsparse_status **rocsparse_dcsrilu0_buffer_size**(*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*, 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_scsrilu0()* and *rocsparse_dcsrilu0()*. 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()* and *rocsparse_dcsrsv_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_scsrilu0()* and *rocsparse_dcsrilu0()*.

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*.

2.9.10.13.3 rocsparse_csrilu0_analysis()

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*)

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()* and *rocsparse_dcsrilu0()*. 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()* and *rocsparse_dcsrsv_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_not_implemented`: `trans` != *rocsparse_operation_none* or *rocsparse_matrix_type* != *rocsparse_matrix_type_general*.

2.9.10.13.4 rocsparse_csrilu0()

rocsparse_status **rocsparse_scsrilu0**(*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*)

rocsparse_status **rocsparse_dcsrilu0**(*rocsparse_handle handle*, rocsparse_int *m*, rocsparse_int *nnz*, **const** *rocsparse_mat_descr descr*, double **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 $m \times m$ CSR matrix A, such that

$$A \approx LU$$

`rocsparse_csrilu0` requires a user allocated temporary buffer. Its size is returned by *rocsparse_scsrilu0_buffer_size()* or *rocsparse_dcsrilu0_buffer_size()*. Furthermore, analysis meta data is required. It can be obtained by *rocsparse_scsrilu0_analysis()* or *rocsparse_dcsrilu0_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 $m \times m$ matrix A, stored in CSR storage format. The following example computes the incomplete LU factorization $M \approx LU$ and solves the preconditioned system $My = x$.

```
// 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);
```

(continues on next page)

(continued from previous page)

```
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,
                            m,
```

(continues on next page)

(continued from previous page)

```
                            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,
```

(continues on next page)

(continued from previous page)

```
                                                               info,
                                                               &position))
{
    printf("U has structural and/or numerical zero at U(%d,%d)\n",
           position,
           position);
}

// Solve Lz = x
rocsparse_dcsrsv_solve(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_dcsrsv_solve(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.
- `[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*.

2.9.10.13.5 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()* or *rocsparse_dcsrilu0_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.

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.

2.9.10.14 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.

2.9.10.14.1 rocsparse_csr2coo()

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.

```
//      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

// 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
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,
```

(continues on next page)

(continued from previous page)

```
          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.

2.9.10.14.2 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.

```
//      1 2 0 3 0
// A =  0 4 5 0 0
//      6 0 0 7 8

```

(continues on next page)

(continued from previous page)

```
rocsparse_int m   = 3;
rocsparse_int n   = 5;
rocsparse_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
rocsparse_int* csr_row_ptr;
rocsparse_int* csr_col_ind;
float* csr_val;

hipMalloc((void**)&csr_row_ptr, sizeof(rocsparse_int) * (m + 1));
hipMalloc((void**)&csr_col_ind, sizeof(rocsparse_int) * nnz);
hipMalloc((void**)&csr_val, sizeof(float) * nnz);

// Convert the coo row indices into csr row offsets
rocsparse_coo2csr(handle,
                  coo_row_ind,
                  nnz,
                  m,
                  csr_row_ptr,
                  rocsparse_index_base_zero);

// Copy the column and value arrays
hipMemcpy(csr_col_ind,
          coo_col_ind,
          sizeof(rocsparse_int) * nnz,
          hipMemcpyDeviceToDevice);

hipMemcpy(csr_val,
          coo_val,
          sizeof(float) * nnz,
          hipMemcpyDeviceToDevice);
```

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

- `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.

2.9.10.14.3 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()* and *rocsparse_dcsr2csc()*. 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 sparse_csr2csc().

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.
- `rocsparse_status_internal_error`: an internal error occurred.

2.9.10.14.4 rocsparse_csr2csc()

rocsparse_status **rocsparse_scsr2csc**(*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,
```

(continues on next page)

(continued from previous page)

```
                        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_internal_error`: an internal error occurred.

2.9.10.14.5 rocsparse_csr2ell_width()

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_internal_error`: an internal error occurred.
- `rocsparse_status_not_implemented`: *rocsparse_matrix_type* != *rocsparse_matrix_type_general*.

2.9.10.14.6 rocsparse_csr2ell()

rocsparse_status **rocsparse_scsr2ell**(*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.

```
//     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

- `[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*.

2.9.10.14.7 rocsparse_ell2csr_nnz()

rocsparse_status **rocsparse_ell2csr_nnz**(*rocsparse_handle handle*, rocsparse_int *m*, rocsparse_int *n*, **const** *rocsparse_mat_descr ell_descr*, rocsparse_int *ell_width*, **const** rocsparse_int **ell_col_ind*, **const** *rocsparse_mat_descr csr_descr*, rocsparse_int **csr_row_ptr*, rocsparse_int **csr_nnz*)

Convert a sparse ELL matrix into a sparse CSR matrix.

`rocsparse_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 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_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_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

- `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`: `ell_descr`, `ell_col_ind`, `csr_descr`, `csr_row_ptr` or `csr_nnz` pointer is invalid.
- `rocsparse_status_not_implemented`: *rocsparse_matrix_type* != *rocsparse_matrix_type_general*.

2.9.10.14.8 rocsparse_ell2csr()

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()* and *rocsparse_dcsr2csc()*. 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 sparse_csr2csc().

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.
- `rocsparse_status_internal_error`: an internal error occurred.

2.9.10.14.9 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_dcsr2hyb**(*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.

```
// 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*.

2.9.10.14.10 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 $p = 0 : 1 : (n - 1)$.

```
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.

```
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

- `[in] handle`: handle to the rocsparse library context queue.
- `[in] n`: size of the map `p`.
- `[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`: `p` pointer is invalid.

2.9.10.14.11 rocsparse_csrsort_buffer_size()

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.

2.9.10.14.12 rocsparse_csrsort()

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 3×3 CSR matrix.

```
//      1 2 3
// A = 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);
```

(continues on next page)

(continued from previous page)

```
// 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*.

2.9.10.14.13 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.

`coosort_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.
- 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 buffer_size pointer is invalid.
- rocsparse_status_internal_error: an internal error occurred.

2.9.10.14.14 rocsparse_coosort_by_row()

rocsparse_status **rocsparse_coosort_by_row**(*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 row.

rocsparse_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 *rocsparse_create_identity_permutation()*.

rocsparse_coosort_by_row 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 3×3 COO matrix by row indices.

```
//      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, 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
```

(continues on next page)

(continued from previous page)

```
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_row(handle,
                         m,
                         n,
                         nnz,
                         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) * 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.

2.9.10.14.15 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 × 3 COO matrix by column indices.

```
//      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);
```

(continues on next page)

(continued from previous page)

```
// Sort the COO matrix
rocsparse_coosort_by_column(handle,
                            m,
                            n,
                            nnz,
                            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) * 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.

2.9.11 Deprecated Libraries

2.9.11.1 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

2.9.11.2 hipeigen

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

For more information, please refer HIPE

2.9.11.3 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

2.9.11.4 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

2.9.11.5 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

2.9.11.6 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

2.9.11.7 hcFFT

hcFFT has been **deprecated** and has been replaced by rocFFT

For more information, please refer HCF

2.10 ROCm Compiler SDK

2.10.1 GCN Native ISA LLVM Code Generator

- ROCm-Native-ISA

2.10.2 ROCm Code Object Format

- ROCm-Codeobj-format

2.10.3 ROCm Device Library

2.10.3.1 OVERVIEW

This repository contains the following libraries:

Name	Comments	Dependencies
irif	Interface to LLVM IR	
ocml	Open Compute Math library(ocml)	irif
oclc	Open Compute library controls (documentation)	
ockl	Open Compute Kernel library.	irif
opencl	OpenCL built-in library	ocml, ockl
hc	Heterogeneous Compute built-in library	ocml, ockl

All libraries are compiled to LLVM Bitcode which can be linked. Note that libraries use specific AMDGPU intrinsics.

2.10.3.2 BUILDING

To build it, use RadeonOpenCompute LLVM/LLD/Clang. Default branch on these repositories is “amd-common”, which may contain AMD-specific codes yet upstreamed.

```
git clone git@github.com:RadeonOpenCompute/llvm.git llvm_amd-common
cd llvm_amd-common/tools
git clone git@github.com:RadeonOpenCompute/lld.git lld
git clone git@github.com:RadeonOpenCompute/clang.git clang
cd ..
mkdir -p build
```

(continues on next page)

(continued from previous page)

```
cd build
cmake \
   -DCMAKE_BUILD_TYPE=Release \
   -DCMAKE_INSTALL_PREFIX=/opt/rocm/llvm \
   -DLLVM_TARGETS_TO_BUILD="AMDGPU;X86" \
   ..
```

Testing also requires amdhsacod utility from ROCm Runtime.

Use out-of-source CMake build and create separate directory to run CMake.

The following build steps are performed:

```
mkdir -p build
cd build
export LLVM_BUILD=... (path to LLVM build)
CC=$LLVM_BUILD/bin/clang cmake -DLLVM_DIR=$LLVM_BUILD -DAMDHSACOD=$HSA_DIR/bin/x86_64/
↪amdhsacod ..
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:

```
export LLVM_BUILD=... (path to LLVM build)
# Build prepare-builtins
cd utils
mkdir build
cd build
cmake -DLLVM_DIR=$LLVM_BUILD ..
make
# Build bitcode libraries
cd ../..
mkdir build
cd build
CC=$LLVM_BUILD/bin/clang 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`/
↪prepare-builtins ..
```

To install artifacts: make install

To run offline tests: make test

To create packages for the library: make package

2.10.3.3 USING BITCODE LIBRARIES

The bitcode libraries should be linked to user bitcode (obtained from source) before final code generation with llvm-link or -mlink-bitcode-file option of clang.

For OpenCL, the list of bitcode libraries includes opencl, its dependencies (ocml, ockl, irif) and oclc control libraries selected according to OpenCL compilation mode. Assuming that the build of this repository was done in /srv/git/ROCm-Device-Libs/build, the following command line shows how to compile simple OpenCL source test.cl into code object test.so:

```
clang -x cl -Xclang -finclude-default-header \
    -target amdgcn--amdhsa -mcpu=fiji \
```

(continues on next page)

(continued from previous page)

```
    -Xclang -mlink-bitcode-file -Xclang /srv/git/ROCm-Device-Libs/build/opencl/opencl.
↪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/ockl/ockl.
↪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_isa_
↪version_803.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/irif/irif.
↪amdgcn.bc \
    test.cl -o test.so
```

2.10.3.4 TESTING

Currently all tests are offline:

- OpenCL source is compiled to LLVM bitcode
- Test bitcode is linked to library bitcode with llvm-link
- Clang OpenCL compiler is run on resulting bitcode, producing code object.
- Resulting code object is passed to llvm-objdump and amdhsacod -test.

The output of tests (which includes AMDGPU disassembly) can be displayed by running ctest -VV in build directory.

Tests for OpenCL conformance kernels can be enabled by specifying -DOCL_CONFORMANCE_HOME= to CMake, for example, cmake … -DOCL_CONFORMANCE_HOME=/srv/hsa/drivers/opencl/tests/extra/hsa/ocl/conformance/1.2

2.10.4 ROCr Runtime

Github link of ROCr Runtime check Here

2.10.4.1 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)

2.10.4.2 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.

2.10.4.3 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/rocm/hsa and /opt/rocm by default. The core runtime package depends on the hsakmt-roct-dev package

Installation instructions can be found in the ROCm Documentation

2.10.4.4 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.

2.10.4.5 Sample

The simplest way to check if the kernel, runtime and base development environment are installed correctly is to run a simple sample. A modified version of the vector_copy sample was taken from the HSA-Runtime-AMD repository and added to the ROCR repository to facilitate this. Build the sample and run it, using this series of commands:

cd ROCR-Runtime/sample && make && ./vector_copy

If the sample runs without generating errors, the installation is complete.

2.10.4.6 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.

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.

2.11 ROCm System Management

2.11.1 ROCm-SMI

ROCm System Management Interface

This repository includes the rocm-smi tool. This tool exposes functionality for clock and temperature management of your 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 a quick excerpt:

usage: rocm-smi [-h] [-d DEVICE [DEVICE . . .]] [-i] [-v] [–showhw] [-t] [-c] [-g] [-f] [-p] [-P] [-o] [-m] [-M] [-l]
[-s] [-u] [–showmemuse] [-b] [–showreplaycount] [-S] [–showvoltage] [–showrasinfo BLOCK [BLOCK . . .]] [–showfwinfo [BLOCK [BLOCK . . .]]] [–showproductname] [-a] [–showmeminfo TYPE [TYPE . . .]] [–showdriverversion] [–showuniqueid] [–showserial] [–showpids] [–showxgmierr] [–alldevices] [-r] [–setsclk LEVEL [LEVEL . . .]] [–setmclk LEVEL [LEVEL . . .]] [–setpcie LEVEL [LEVEL . . .]] [–setslevel SCLKLEVEL SCLK SVOLT] [–setmlevel MCLKLEVEL MCLK MVOLT] [–resetfans] [–setfan LEVEL] [–setperflevel LEVEL] [–setoverdrive %] [–setmemoverdrive %] [–setpoweroverdrive WATTS] [–resetpoweroverdrive] [–setprofile SETPROFILE] [–resetprofile] [–rasenable BLOCK ERRTYPE] [–rasdisable BLOCK ERRTYPE] [–rasinject BLOCK] [–gpureset] [–resetxgmierr] [–load FILE | –save FILE] [–autorespond RESPONSE] [–loglevel ILEVEL] [–json]

AMD ROCm System Management Interface

optional arguments:

-h, –help	show this help message and exit
–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

-i, –showid	Show GPU ID
-v, –showvbios	Show VBIOS version
–showhw	Show Hardware details
-t, –showtemp	Show current temperature
-c, –showclocks	Show current clock frequencies
-g, –showgpuclocks	Show current GPU clock frequencies
-f, –showfan	Show current fan speed
-p, –showperflevel	Show current PowerPlay Performance Level
-P, –showpower	Show current GPU ASIC power consumption
-o, –showoverdrive	Show current OverDrive level
-m, –showmemoverdrive	Show current GPU Memory Clock OverDrive level
-M, –showmaxpower	Show maximum graphics package power this GPU will consume
-l, –showprofile	Show Compute Profile attributes
-s, –showclkfrq	Show supported GPU and Memory Clock
-u, –showuse	Show current GPU use

Table 1 – continued from previous page

–showmemuse	Show current GPU memory used
-b, –showbw	Show estimated PCIe use
–showreplaycount	Show PCIe Replay Count
-S, –showclkvolt	Show supported GPU and Memory Clocks and Voltages
–showvoltage	Show current GPU voltage
–showrasinfo BLOCK [BLOCK . . .]	Show RAS enablement information and error counts for the specified block(s)
–showfwinfo [BLOCK [BLOCK . . .]]	Show FW information
–showproductname	Show SKU/Vendor name
-a, –showallinfo	Show all SMI-supported values values
–showmeminfo TYPE [TYPE . . .]	Show Memory usage information for given block(s) TYPE
–showdriverversion	Show kernel driver version
–showuniqueid	Show GPU’s Unique ID
–showserial	Show GPU’s Serial Number
–showpids	Show current running KFD PIDs
–showxgmierr	Show XGMI error information since last read
–alldevices	Execute command on non-AMD devices as well as AMD devices
-r, –resetclocks	Reset clocks to OverDrive to default
–setsclk LEVEL [LEVEL . . .]	Set GPU Clock Frequency Level 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)
–setslevel 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
–resetfans	Reset fans to automatic (driver) control
–setfan LEVEL	Set GPU Fan Speed Level
–setperflevel LEVEL	Set PowerPlay Performance Level
–setoverdrive %	Set GPU OverDrive level
–setmemoverdrive %	Set GPU Memory Overclock OverDrive level (requires manuallhigh Perf level)
–setpoweroverdrive WATTS	Set the maximum GPU power using Power OverDrive in Watts
–resetpoweroverdrive	Set the maximum GPU power back to the device deafult state
–setprofile SETPROFILE	Specify Power Profile level (#) or a quoted string of CUSTOM Profile attributes “# # #
–resetprofile	Reset Power Profile back to default
–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)
–gpureset	Reset specified GPU (One GPU must be specified)
–autorespond RESPONSE	Response to automatically provide for all prompts (NOT RECOMMENDED)
–loglevel ILEVEL	How much output will be printed for what program is doing, one of debug/info/warning
–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
```

–setperflevel 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 –setslevel/–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 –showprofile 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 decresed (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 decresed (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

-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

OverDrive settings

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 0xffffbfff (11111111111111111011111111111111), then enabling the OverDrive bit would make it 0xffffffff (11111111111111111111111111111111).

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:

```
./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.

2.11.1.1 Programing ROCm-SMI

2.11.2 SYSFS Interface

2.11.2.1 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.

Note that motherboards vary widely in the connections to sensor chips.There is no standard that ensures connection of sensor to CPU,variation in external resistor value and conversion calculation we can not hard code this into the driver and have to be done in user space.

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*. 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

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.

2.11.2.1.1 Global attributes

name	The chip name.This should be a short, lowercase string, not containing whitespace, dashes, or the wildcard character '*'.This attribute represents the chip name. It is the only mandatory attribute.I2C devices get this attribute created automatically. RO
update_interval	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.

2.11.2.1.2 Voltages

in[0-*]_min	Voltage min value. Unit: millivolt RW
in[0-*]_lcrit	Voltage critical min value. Unit: millivolt RW If voltage drops to or below this limit, the system may take drastic action such as power down or reset. At the very least, it should report a fault.
in[0-*]_max	Voltage max value. Unit: millivolt RW
in[0-*]_crit	Voltage critical max value. Unit: millivolt RW If voltage reaches or exceeds this limit, the system may take drastic action such as power down or reset. At the very least, it should report a fault.
in[0-*]_input	Voltage input value. Unit: millivolt RO Voltage measured on the chip pin.Actual voltage depends on the scaling resistors on the motherboard, as recommended in the chip datasheet.This varies by chip and by motherboard. Because of this variation, values are generally NOT scaled by the chip driver, and must be done by the application.However, some 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.
in[0-*]_average	Average voltage Unit: millivolt RO
in[0-*]_lowest	Historical minimum voltage

Also see the Alarms section for status flags associated with voltages.

2.11.2.1.3 Fans

fan[1-*]_min	Fan minimum value Unit: revolution/min (RPM) RW
fan[1-*]_max	Fan maximum value Unit: revolution/min (RPM) Only rarely supported by the hardware. RW
fan[1-*]_input	Fan input value. Unit: revolution/min (RPM) RO
fan[1-*]_div	Fan divisor. Integer value in powers of two (1, 2, 4, 8, 16, 32, 64, 128). RW Some chips only support values 1, 2, 4 and 8. Note that this is actually an internal clock divisor, which affects the measurable speed range, not the read value.
fan[1-*]_pulses	Number of tachometer pulses per fan revolution. Integer value, typically between 1 and 4. RW 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.
fan[1-*]_target	Desired fan speed Unit: revolution/min (RPM) RW Only makes sense if the chip supports closed-loop fan speed control based on the measured fan speed.
fan[1-*]_label	Suggested fan channel label

Also see the Alarms section for status flags associated with fans.

2.11.2.1.4 PWM

pwm[1-*]	Pulse width modulation fan control. Integer value in the range 0 to 255 RW 255 is max or 100%.
pwm[1-*]_enable	Fan speed control method: 0: no fan speed control (i.e. fan at full speed) 1: manual fan speed control enabled (using pwm[1-*]) 2+: automatic fan speed control enabled Check individual chip documentation files for automatic mode details. RW
pwm[1-*]_mode	0: DC mode (direct current) 1: PWM mode (pulse-width modulation) RW
pwm[1-*]_freq	Base PWM frequency in Hz. Only possibly available when pwmN_mode is PWM, but not always present even then. RW
pwm[1-*]_auto_channels_temp	Select which temperature channels affect this PWM output in auto mode. Bitfield, 1 is temp1, 2 is temp2, 4 is temp3 etc. . . Which values are possible depend on the chip used. RW
pwm[1-*]_auto_point[1-*]_pwm pwm[1-*]_auto_point[1-*]_temp pwm[1-*]_auto_point[1-*]_temp_hyst	Define the PWM vs temperature curve. Number of trip points is chip-dependent.Use this for chips which associate trip points to PWM output channels. RW
temp[1-*]_auto_point[1-*]_pwm temp[1-*]_auto_point[1-*]_temp temp[1-*]_auto_point[1-*]_temp_hyst	Define the PWM vs temperature curve. Number of trip points is chip dependent. Use this for chips which associate trip points to temperature channels. RW

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.

2.11.2.1.5 Temperatures

temp[1-*]_type	Sensor type selection. Integers 1 to 6 RW 1: CPU embedded diode 2: 3904 transistor 3: thermal diode 4: thermistor 5: AMD AMDSI 6: Intel PECI Not all types are supported by all chips
temp[1-*]_max	Temperature max value. Unit: millidegree Celsius (or millivolt, see below) RW
temp[1-*]_min	Temperature min value. Unit: millidegree Celsius RW
temp[1-*]_max_hyst	Temperature hysteresis value for max limit. Unit: millidegree Celsius Must be reported as an absolute temperature, NOT a delta from the max value. RW
temp[1-*]_min_hyst	Temperature hysteresis value for min limit. Unit: millidegree Celsius Must be reported as an absolute temperature, NOT a delta from the min value. RW
temp[1-*]_input	Temperature input value. Unit: millidegree Celsius RO
temp[1-*]_crit	Temperature critical max value, typically greater than corresponding temp_max values. Unit: millidegree Celsius RW
temp[1-*]_crit_hyst	Temperature hysteresis value for critical limit.

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.

2.11.2.1.6 Currents

curr[1-*]_max	Current max value Unit: milliampere RW
curr[1-*]_min	Current min value. Unit: milliampere RW
curr[1-*]_lcrit	Current critical low value Unit: milliampere RW
curr[1-*]_crit	Current critical high value. Unit: milliampere RW
curr[1-*]_input	Current input value Unit: milliampere RO
curr[1-*]_average	Average current use Unit: milliampere RO
curr[1-*]_lowest	Historical minimum current Unit: milliampere RO
curr[1-*]_highest	Historical maximum current Unit: milliampere RO
curr[1-*]_reset_history	Reset currX_lowest and currX_highest WO
curr_reset_history	Reset currX_lowest and currX_highest for all sensors WO

Also see the Alarms section for status flags associated with currents.

2.11.2.1.7 Power

power[1-*]_average	Average power use Unit: microWatt RO
power[1-*]_average_interval	Power use averaging interval. A poll notification is sent to this file if the hardware changes the averaging interval. Unit: milliseconds RW
power[1-*]_average_interval_max	Maximum power use averaging interval Unit: milliseconds RO
power[1-*]_average_interval_min	Minimum power use averaging interval Unit: milliseconds RO
power[1-*]_average_highest	Historical average maximum power use Unit: microWatt RO
power[1-*]_average_lowest	Historical average minimum power use Unit: microWatt RO
power[1-*]_average_max	A poll notification is sent to power[1-*]_average when power use rises above this value. Unit: microWatt RW
power[1-*]_average_min	A poll notification is sent to power[1-*]_average when power use sinks below this value. Unit: microWatt RW
power[1-*]_input	Instantaneous power use Unit: microWatt

Also see the Alarms section for status flags associated with power readings.

2.11.2.1.8 Energy

energy[1-*]_input	Cumulative energy use Unit: microJoule RO

2.11.2.1.9 Humidity

humidity[1-*]_input	Humidity Unit: milli-percent (per cent mille, pcm) RO

2.11.2.1.10 Alarms

Each channel or limit may have an associated alarm file, containing a boolean value. 1 means than 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.

in[0-*]_alarm curr[1-*]_alarm power[1-*]_alarm fan[1-*]_alarm temp[1-*]_alarm	Channel alarm 0: no alarm 1: alarm RO

OR

in[0-*]_min_alarm in[0-*]_max_alarm in[0-*]_lcrit_alarm in[0-*]_crit_alarm curr[1-*]_min_alarm curr[1-*]_max_alarm curr[1-*]_lcrit_alarm curr[1-*]_crit_alarm power[1-*]_cap_alarm power[1-*]_max_alarm power[1-*]_crit_alarm fan[1-*]_min_alarm fan[1-*]_max_alarm temp[1-*]_min_alarm temp[1-*]_max_alarm temp[1-*]_lcrit_alarm temp[1-*]_crit_alarm temp[1-*]_emergency_alarm	Limit alarm 0: no alarm 1: alarm RO

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.

fan[1-*]_fault temp[1-*]_fault	Input fault condition 0: no fault occurred 1: fault condition RO

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

beep_enable	Master beep enable 0: no beeps 1: beeps RW
in[0-*]_beep curr[1-*]_beep fan[1-*]_beep temp[1-*]_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:

alarms	Alarm bitmask. RO Integer representation of one to four bytes. A '1' bit means an alarm. Chips should be programmed for 'comparator' mode so that the alarm will 'come back' after you read the register if it is still valid. 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. Bits are defined in kernel/include/sensors.h.
beep_mask	Bitmask for beep. Same format as 'alarms' with the same bit locations, use discouraged for the same reason. Use individual *_beep files instead. RW

2.11.2.1.11 Intrusion detection

intrusion[0-*]_alarm	Chassis intrusion detection 0: OK 1: intrusion detected RW 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.
intrusion[0-*]_beep	Chassis intrusion beep 0: disable 1: enable RW

2.11.2.1.11.1 sysfs attribute writes interpretation

hwmon sysfs attributes always contain numbers, so the first thing to do is to convert the input to a number, there are 2 ways todo this depending whether the number can be negative or not: unsigned long u = simple_strtoul(buf, NULL, 10); long s = simple_strtol(buf, NULL, 10);

With buf being the buffer with the user input being passed by the kernel. Notice that we do not use the second argument of strto[u]l, and thus cannot tell when 0 is returned, if this was really 0 or is caused by invalid input. This is done deliberately as checking this everywhere would add a lot of code to the kernel.

Notice that it is important to always store the converted value in an unsigned long or long, so that no wrap around can happen before any further checking.

After the input string is converted to an (unsigned) long, the value should be checked if its acceptable. Be careful with further conversions on the value before checking it for validity, as these conversions could still cause a wrap around before the check. For example do not multiply the result, and only add/subtract if it has been divided before the add/subtract.

What to do if a value is found to be invalid, depends on the type of the sysfs attribute that is being set. If it is a continuous setting like a tempX_max or inX_max attribute, then the value should be clamped to its limits using clamp_val(value, min_limit, max_limit). If it is not continuous like for example a tempX_type, then when an invalid value is written, -EINVAL should be returned.

Example1, temp1_max, register is a signed 8 bit value (-128 - 127 degrees):

```
long v = simple_strtol(buf, NULL, 10) / 1000;
v = clamp_val(v, -128, 127);
/* write v to register */
```

Example2, fan divider setting, valid values 2, 4 and 8:

```
unsigned long v = simple_strtoul(buf, NULL, 10);

switch (v) {
case 2: v = 1; break;
case 4: v = 2; break;
case 8: v = 3; break;
default:
        return -EINVAL;
}
/* write v to register */
```

2.11.2.1.12 Performance

The pcie_bw sysfs file will report the usage of the PCIe bus over the last second, as a string with 3 integers: “bytes-received bytes-sent mps” . As there is no efficient way to calculate the size of each packet transmitted to and from the GPU in real time, the maximum payload size (mps), or the largest size of a PCIe packet, is included. The estimated bandwidth can then be calculated using by “bytes-received*mps + bytes-sent*mps” sed and multiplied by the number of packets received and sent.

2.11.3 KFD Topology

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
/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.

/sys/devices/virtual/kfd/kfd/topology/nodes/0/
/sys/devices/virtual/kfd/kfd/topology/nodes/1/
.
.
/sys/devices/virtual/kfd/kfd/topology/nodes/N-1/

2.11.3.1 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.

2.11.3.2 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.

2.11.3.3 Memory

The memory bank information attached to this agent is populated in "mem_banks" subdirectory. /sys/devices/virtual/kfd/kfd/topology/nodes/N/mem_banks

2.11.3.4 Cache

The caches available for this agent is populated in "cache" subdirectory /sys/devices/virtual/kfd/kfd/topology/nodes/N/cache

2.11.3.5 IO-LINKS

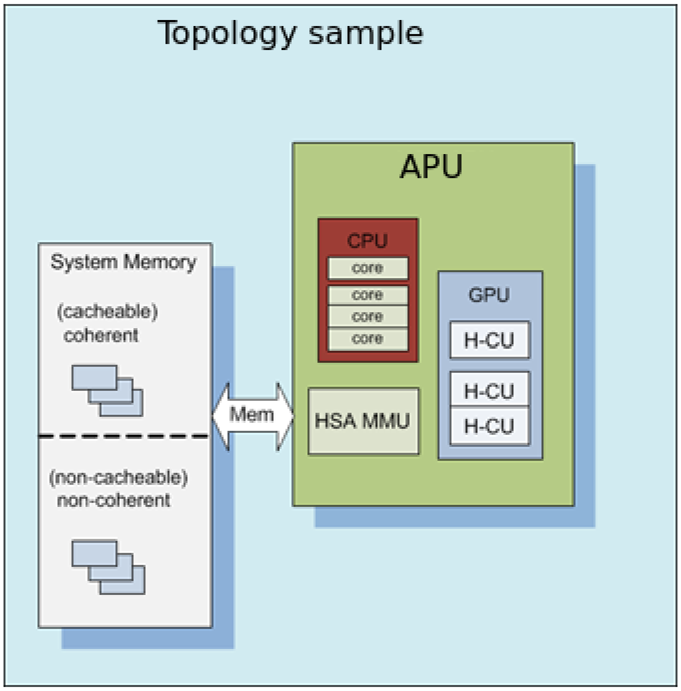
The IO links provides HSA agent interconnect information with latency (cost) between agents. This is useful for peer-to-peer transfers.

2.11.3.6 How to use topology information

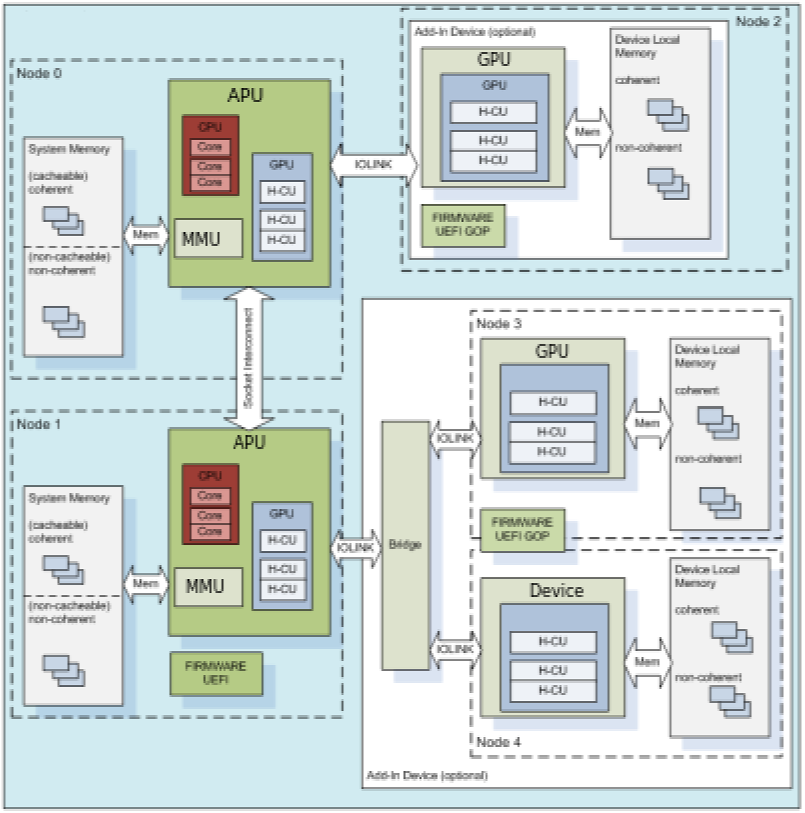
The information provided in sysfs should not be directly used by application software. Application software should always use Thunk library API (libhsakmt) to access topology information. Please refer to Thunk API for more information.

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.



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

2.12 ROCm Virtualization & Containers

2.12.1 PCIe Passthrough on KVM

The following KVM-based instructions assume a headless host with an input/output memory management unit (IOMMU) to pass peripheral devices such as a GPU to guest virtual machines. If you know your host supports IOMMU but the below command does not find "svm" or "vxm", you may need to enable IOMMU in your BIOS.

```
cat /proc/cpuinfo | grep -E "svm|vxm"
```

2.12.1.1 Ubuntu 16.04

Assume we use an intel system that support VT-d , with fresh ubuntu 16.04 installed

a. Install necessary packages and prepare for pass through device

1. sudo apt-get install qemu-kvm qemu-system bridge-utils virt-manager ubuntu-vm-builder libvirt-dev
2. **add following modules into /etc/modules**

   vfio
   vfio_iommu_type1
   vfio_pci
   kvm
   kvm_intel

   add intel_iommu=on in /etc/default/grub

   GRUB_CMDLINE_LINUX_DEFAULT="quiet splash intel_iommu=on"

   sudo update-grub
3. **Blacklist amdgpu by adding the following line to /etc/modprobe.d/blacklist.conf**

   blacklist amdgpu

b. Bind pass through device to vfio-pci

1. Create a script file (vfio-bind) under /usr/bin. The script file has the following content:

```
#!/bin/bash
modprobe vfio-pci
for dev in "$@"; do
        vendor=$(cat /sys/bus/pci/devices/$dev/vendor)
        device=$(cat /sys/bus/pci/devices/$dev/device)
```

(continues on next page)

(continued from previous page)

```
        if [ -e /sys/bus/pci/devices/$dev/driver ]; then
                echo $dev > /sys/bus/pci/devices/$dev/driver/unbind
        fi
        echo $vendor $device > /sys/bus/pci/drivers/vfio-pci/new_id
done
```

2. Make it executable by enter the command

```
chmod 755 vfio-bind
```

3. Bind the device to vfio by running the command for the three pass through devices

```
lspci -n -d 1002:
        83:00.0 0300: 1002:7300 (rev ca)
vfio.bind 0000:83:00.0
```

4. sudo reboot

c. Pass through device to guest VM

1. Start VMM by running "virt-manager" as root. Follow the on screen instruction to create one virtual machine(VM), make sure CPU copy host CPU configuration, network use bridge mode.
2. Add Hardware –> Select PCI Host device, select the appropriate device to pass through. ex:0000:83:00.0
3. sudo setpci -s 83:00.0 CAP_EXP+28.l=40
4. sudo reboot

After reboot, start virt-manager and then start the VM, inside the VM , lspci -d 1002: should shows the pass throughed device.

2.12.1.2 Fedora 27 or CentOS 7 (1708)

From a fresh install of Fedora 27 or CentOS 7 (1708)

a. Install necessary packages and prepare for pass through device

1. **Identity the vendor and device id(s) for the PCIe device(s) you wish to passthrough, e.g., 1002:6861 and 1002:aaf8 for an A**
   lspci -nnk
2. **Install virtualization packages** sudo dnf install @virtualization sudo usermod -G libvirt -a $(whoami) sudo usermod -G kvm -a $(whoami)
3. **Enable IOMMU in the GRUB_CMDLINE_LINUX variable for your target kernel**
   a. **For an AMD CPU** sudo sed 's/quiet/quiet amd_iommu=on iommu=pt/' /etc/sysconfig/grub
   b. **For an Intel CPU** sudo sed 's/quiet/quiet intel_iommu=on iommu=pt/' /etc/sysconfig/grub

b. Bind pass through device to vfio-pci

4. Preempt the host claiming the device by loading a stub driver

:: echo "options vfio-pci ids=1002:6861,1002:aaf8" | sudo tee -a /etc/modprobe.d/vfio.conf echo "options vfio-pci disable_vga=1" | sudo tee -a /etc/modprobe.d/vfio.conf sed 's/quiet/quiet rd.driver.pre=vfio-pci video=efifb:off/' /etc/sysconfig/grub

5. Update the kernel boot settings

:: sudo grub2-mkconfig -o /etc/grub2-efi.cfg echo 'add_drivers+="vfio vfio_iommu_type1 vfio_pci"' | sudo tee -a /etc/dracut.conf.d/vfio.conf sudo dracut -f –kver *uname -r*

6. Reboot and verify that vfio-pci driver has been loaded

:: lspci -nnk

c. Pass through device to guest VM

1. Within virt-manager the device should now appear in the list of available PCI devices

Note: To pass a device within a particular IOMMU group, all devices within that IOMMU group must also be passed. You may wish to refer here for more details, such as the following script that lists all IOMMU groups and the devices within them.

```
#!/bin/bash
shopt -s nullglob
for d in /sys/kernel/iommu_groups/*/devices/*; do
    n=${d#*/iommu_groups/*}; n=${n%%/*}
    printf 'IOMMU Group %s ' "$n"
    lspci -nns "${d##*/}"
done;
```

2.12.2 ROCm-Docker

Radeon Open Compute Platform for docker

Please refer ROCm-Docker

This repository contains a framework for building the software layers defined in the Radeon Open Compute Platform into portable docker images. The following are docker dependencies, which should be installed on the target machine.

- Docker on Ubuntu systems or Fedora systems
- Highly recommended: Docker-Compose to simplify container management

2.12.2.1 Docker Hub

Looking for an easy start with ROCm + Docker? The rocm/rocm-terminal image is hosted on Docker Hub . After the ROCm kernel is installed , pull the image from Docker Hub and create a new instance of a container.

```
sudo docker pull rocm/rocm-terminal
sudo docker run -it --device=/dev/kfd --device=/dev/dri --group-add video rocm/rocm-
↪terminal
```

2.12.2.2 ROCm-docker set up guide

Installation instructions and asciicasts demos are available to help users quickly get running with rocm-docker. Visit the set up guide to read more.

F.A.Q

When working with the ROCm containers, the following are common and useful docker commands:

- A new docker container typically does not house apt repository meta-data. Before trying to install new software using apt, make sure to run sudo apt update first
- A message like the following typically means your user **does not** have permissions to execute docker; use sudo or add your user to the docker group.
- Cannot connect to the Docker daemon. Is the docker daemon running on this host?

- Open another terminal into a running container
- sudo docker exec -it <CONTAINER-NAME> bash -l
- **Copy files from host machine into running docker container**
  - sudo docker cp HOST_PATH <CONTAINER-NAME>:/PATH
- **Copy files from running docker container onto host machine**
  - sudo docker cp <CONTAINER-NAME>:/PATH/TO/FILE HOST_PATH
- If receiving messages about no space left on device when pulling images, check the storage driver in use by the docker engine. If its ‘device mapper’, that means the image size limits imposed by the ‘device mapper’ storage driver are a problem Follow the documentation in the quickstart for a solution to change to the storage driver

Saving work in a container

Docker containers are typically ephemeral, and are discarded after closing the container with the **‘–rm’** flag to docker run. However, there are times when it is desirable to close a container that has arbitrary work in it, and serialize it back into a docker image. This may be to to create a checkpoint in a long and complicated series of instructions, or it may be desired to share the image with others through a docker registry, such as docker hub.

```
sudo docker ps -a  # Find container of interest
sudo docker commit <container-name> <new-image-name>
sudo docker images # Confirm existence of a new image
```

2.12.2.3 Details

Docker does not virtualize or package the linux kernel inside of an image or container. This is a design decision of docker to provide lightweight and fast containerization. The implication for this on the ROCm compute stack is that in order for the docker framework to function, **the ROCm kernel and corresponding modules must be installed on the host machine.** Containers share the host kernel, so the ROCm KFD component ROCK-Kernel-Driver1 functions outside of docker.

Installing ROCK on the host machine.

An apt-get repository is available to automate the installation of the required kernel and kernel modules.

2.12.2.4 Building images

There are two ways to install rocm components:

1.install from the rocm apt/rpm repository (repo.radeon.com)

2.build the components from source and run install scripts

The first method produces docker images with the smallest footprint and best building speed. The footprint is smaller because no developer tools need to be installed in the image, an the images build speed is fastest because typically downloading binaries is much faster than downloading source and then invoking a build process. Of course, building components allows much greater flexibility on install location and the ability to step through the source with debug builds. ROCm-docker supports making images either way, and depends on the flags passed to the setup script.

The setup script included in this repository is provides some flexibility to how docker containers are constructed. Unfortunately, Dockerfiles do not have a preprocessor or template language, so typically build instructions are hardcoded. However, the setup script allows us to write a primitive ‘template’, and after running it instantiates baked dockerfiles with environment variables substituted in. For instance, if you wish to build release images and debug images, first run the setup script to generate release dockerfiles and build the images. Then, run the setup script again and specify debug dockerfiles and build new images. The docker images should generate unique image names and not conflict with each other.

setup.sh

Currently, the setup.sh scripts checks to make sure that it is running on an **Ubuntu system**, as it makes a few assumptions about the availability of tools and file locations. If running rocm on a Fedora machine, inspect the source of setup.sh and issue the appropriate commands manually. There are a few parameters to setup.sh of a generic nature that affects all images built after running. If no parameters are given, built images will be based off of Ubuntu 16.04 with rocm components installed from debians downloaded from packages.amd.com. Supported parameters can be queried with ./setup –help.

setup.sh parameters	parameter [default]	description
–ubuntu	xx.yy [16.04]	Ubuntu version for to inherit base image
–install-docker-compose		helper to install the docker-compose tool

The following parameters are specific to building containers that compile rocm components from source.

setup.sh parameters	parameter [default]	description
–tag	string [‘master’]	string representing a git branch name
–branch	string [‘master’]	alias for tag
–debug		build code with debug flags

./setup generates finalized Dockerfiles from textual template files ending with the .template suffix. Each sub-directory of this repository corresponds to a docker ‘build context’ responsible for a software layer in the ROCm stack. After running the script, each directory contains generated dockerfiles for building images from debians and from source.

2.12.2.5 Docker compose

./setup prepares an environment to be controlled with Docker Compose. While docker-compose is not necessary for proper operation, it is highly recommended. setup.sh does provide a flag to simplify the installation of this tool. Docker-compose coordinates the relationships between the various ROCm software layers, and it remembers flags that should be passed to docker to expose devices and import volumes.

Example of using docker-compose

docker-compose.yml provides services that build and run containers. YAML is structured data, so it’s easy to modify and extend. The setup.sh script generates a .env file that docker-compose reads to satisfy the definitions of the variables in the .yml file.

- docker-compose run –rm rocm – Run container using rocm packages
- docker-compose run –rm rocm-from-src – Run container with rocm built from source

Docker-compose	description
docker-compose	docker compose executable
run	sub-command to bring up interactive container
–rm	when shutting the container down, delete it
rocm	application service defined in **docker-compose.yml**

rocm-user has root privileges by default

The dockerfile that serves as a ‘terminal’ creates a non-root user called **rocm-user**. This container is meant to serve as a development environment (therefore apt-get is likely needed), the user has been added to the linux sudo group. Since it is somewhat difficult to set and change passwords in a container (often requiring a rebuild), the password prompt has been disabled for the sudo group. While this is convenient for development to be able sudo apt-get install packages, it does imply lower security in the container.

To increase container security:

1.Eliminate the sudo-nopasswd COPY statement in the dockerfile and replace with

2.Your own password with RUN echo 'account:password' | chpasswd

The docker.ce release 18.02 has known defects working with rocm-user account insider docker image. Please upgrade docker package to the 18.04 build.

Footnotes:

[1] It can be installed into a container, it just doesn't do anything because containers do not go through the traditional boot process. We actually do provide a container for ROCK-Kernel-Driver, but it not used by the rest of the docker images. It does provide isolation and a reproducible environment for kernel development.

2.13 Remote Device Programming

2.13.1 ROCmRDMA

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;

2.13.1.1 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 manufacturses 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.

2.13.1.2 ROCmRDMA interface specification

The implementation of ROCmRDMA interface could be found in [amd_rdma.h] file.

2.13.1.3 Data structures

```
/**
 * Structure describing information needed to P2P access from another device
 * to specific location of GPU memory
 */
struct amd_p2p_info {
        uint64_t        va;             /**< Specify user virt. address
```

(continues on next page)

(continued from previous page)

```
                                          * which this page table described
                                          */

        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);
};
```

2.13.1.4 The function to query ROCmRDMA interface

```
/**
 * amdkfd_query_rdma_interface - Return interface (function pointers table) for
 *                             rdma interface
 *
 *
 *    \param interace     - OUT: Pointer to interface
 *    \return 0 if operation was successful.
 */
int amdkfd_query_rdma_interface(const struct amd_rdma_interface **rdma);
```

2.13.1.5 The function to query ROCmRDMA interface

```
/**
 * amdkfd_query_rdma_interface - Return interface (function pointers table) for rdma␣
↪interface
 * \param interace     - OUT: Pointer to interface
```

(continues on next page)

(continued from previous page)

```
 * \return 0 if operation was successful.
 */
 int amdkfd_query_rdma_interface(const struct amd_rdma_interface **rdma);
```

2.13.1.6 ROCmRDMA interface functions description

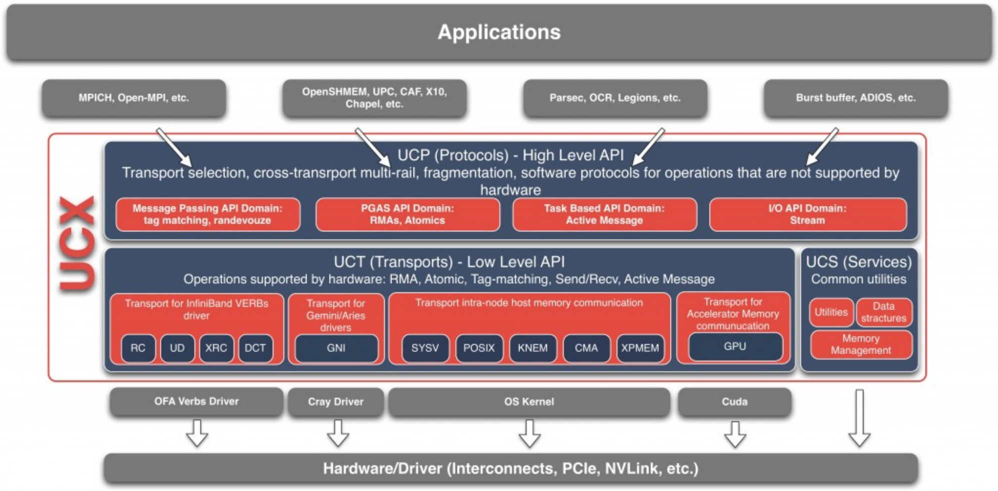
```
/**
 * This function makes the pages underlying a range of GPU virtual memory
 * accessible for DMA operations from another PCIe device
 *
 * \param   address       - The start address in the Unified Virtual Address
 *                          space in the specified process
 * \param   length        - 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.
 *                          Client should  immediately stop any transfer
 *                          operations and returned as soon as possible.
 *                          After return all resources associated with address
 *                          will be release and no access will be allowed.
 * \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);
```

```
/**
 * This function release resources previously allocated by get_pages() call.
 * \param   p_p2p_data - A pointer to pointer to amd_p2p_info entries
 *                     allocated by get_pages() call.
 * \return  0 if operation was successful
 */
 int put_pages(struct amd_p2p_info **p_p2p_data)
```

```
/**
  * Check if given address belongs to GPU address space.
  * \param   address - Address to check
  * \param   pid     - Process to which given address belongs.
  *                    Could be NULL if current one.
  * \return  0       - This is not GPU address managed by AMD driver
  *          1       - This is GPU address managed by AMD driver
  */
  int is_gpu_address(uint64_t address, struct pid *pid);
```

2.13.2 UCX

2.13.2.1 Introduction


Applications
MPICH, Open-MPI, etc.
OpenSHMEM, UPC, CAF, X10, Chapel, etc.
Parsec, OCR, Legions, etc.
Burst buffer, ADIOS, etc.
UCX
UCP (Protocols) - High Level API
Transport selection, cross-transrport multi-rail, fragmentation, software protocols for operations that are not supported by hardware
Message Passing API Domain: tag matching, randevouze
PGAS API Domain: RMAs, Atomics
Task Based API Domain: Active Message
I/O API Domain: Stream
UCT (Transports) - Low Level API
Operations supported by hardware: RMA, Atomic, Tag-matching, Send/Recv, Active Message
Transport for InfiniBand VERBs driver
RC
UD
XRC
DCT
Transport for Gemini/Aries drivers
GNI
Transport intra-node host memory communication
SYSV
POSIX
KNEM
CMA
XPMEM
Transport for Accelerator Memory communucation
GPU
UCS (Services)
Common utilities
Utilities
Data structures
Memory Management
OFA Verbs Driver
Cray Driver
OS Kernel
Cuda
Hardware/Driver (Interconnects, PCIe, NVLink, etc.)

2.13.2.2 UCX Quick start

Compiling UCX

```
% ./autogen.sh
% ./contrib/configure-release --prefix=$PWD/install
% make -j8 install
```

2.13.2.3 UCX API usage examples

https://github.com/openucx/ucx/tree/master/test/examples

2.13.2.4 Running UCX

2.13.2.4.1 UCX internal performance tests

This infrastructure provided a function which runs a performance test (in the current thread) on UCX communication APIs. The purpose is to allow a developer make optimizations to the code and immediately test their effects. The infrastructure provides both an API, and a standalone tool which uses that API - ucx_perftest. The API is also used for unit tests. Location: src/tools/perf

Features of the library:

- uct_perf_test_run() is the function which runs the test. (currently only UCT API is supported)
- No need to do any resource allocation - just pass the testing parameters to the API
- Requires running the function on 2 threads/processes/nodes - by passing RTE callbacks which are used to bootstrap the connections.
- Two testing modes - ping-pong and unidirectional stream (TBD bi-directional stream)
- Configurabe message size, and data layout (short/bcopy/zcopy)

- Supports: warmup cycles, unlimited iterations.
- UCT Active-messages stream is measured with simple flow-control.
- Tests driver is written in C++ (C linkage), to take advantage of templates.
- **Results are reported to callback function at the specified intervals, and also returned from the API call.**
    - Including: latency, message rate, bandwidth - iteration average, and overall average.

Features of ucx_perftest:

- Have pre-defined list of tests which are valid combinations of operation and testing mode.
- Can be run either as client-server application, as MPI application, or using libRTE.
- Supports: CSV output, numeric formatting.
- **Supports “batch mode” - write the lists of tests to run to a text file (see example in contrib/perf) and run them one after an**
    - “Cartesian” mode: if several batch files are specified, all possible combinations are executed!

```
$ ucx_perftest  -h
Usage: ucx_perftest [ server-hostname ] [ options ]

This test can be also launched as an MPI application
  Common options:

  Test options:
     -t <test>      Test to run.
                     am_lat : active message latency.
                    put_lat : put latency.
                    add_lat : atomic add latency.
                        get : get latency / bandwidth / message rate.
                       fadd : atomic fetch-and-add latency / message rate.
                       swap : atomic swap latency / message rate.
                      cswap : atomic compare-and-swap latency / message rate.
                      am_bw : active message bandwidth / message rate.
                     put_bw : put bandwidth / message rate.
                     add_mr : atomic add message rate.

     -D <layout>    Data layout.
                      short : Use short messages API (cannot used for get).
                      bcopy : Use copy-out API (cannot used for atomics).
                      zcopy : Use zero-copy API (cannot used for atomics).

     -d <device>    Device to use for testing.
     -x <tl>        Transport to use for testing.
     -c <cpu>       Set affinity to this CPU. (off)
     -n <iters>     Number of iterations to run. (1000000)
     -s <size>      Message size. (8)
     -H <size>      AM Header size. (8)
     -w <iters>     Number of warm-up iterations. (10000)
     -W <count>     Flow control window size, for active messages. (128)
     -O <count>     Maximal number of uncompleted outstanding sends. (1)
     -N             Use numeric formatting - thousands separator.
     -f             Print only final numbers.
     -v             Print CSV-formatted output.
     -p <port>      TCP port to use for data exchange. (13337)
     -b <batchfile> Batch mode. Read and execute tests from a file.
                     Every line of the file is a test to run. The first word is the
```

(continues on next page)

(continued from previous page)

```
                     test name, and the rest are command-line arguments for the test.
     -h             Show this help message.

  Server options:
     -l             Accept clients in an infinite loop
```

Example - using mpi as a launcher

When using mpi as the launcher to run ucx_perftest, please make sure that your ucx library was configured with mpi. Add the following to your configure line:

```
--with-mpi=/path/to/mpi/home
$salloc -N2 --ntasks-per-node=1 mpirun --bind-to core --display-map ucx_perftest -d␣
↪mlx5_1:1 \
                                     -x rc_mlx5 -t put_lat
salloc: Granted job allocation 6991
salloc: Waiting for resource configuration
salloc: Nodes clx-orion-[001-002] are ready for job
 Data for JOB [62403,1] offset 0

 ========================   JOB MAP   ========================

 Data for node: clx-orion-001   Num slots: 1    Max slots: 0    Num procs: 1
        Process OMPI jobid: [62403,1] App: 0 Process rank: 0

 Data for node: clx-orion-002   Num slots: 1    Max slots: 0    Num procs: 1
      Process OMPI jobid: [62403,1] App: 0 Process rank: 1

 =============================================================

+--------------+-----------------------------+---------------------+------------------
↪-----+
|              |       latency (usec)        |   bandwidth (MB/s)  |  message rate␣
↪(msg/s) |
+--------------+---------+---------+---------+----------+----------+-----------+------
↪-----+
| # iterations | typical | average | overall |  average |  overall |   average |  ␣
↪overall |
+--------------+---------+---------+---------+----------+----------+-----------+------
↪-----+
      586527     0.845     0.852     0.852       4.47       4.47      586527      ␣
↪586527
     1000000     0.844     0.848     0.851       4.50       4.48      589339      ␣
↪587686
```

2.13.2.4.2 OpenMPI and OpenSHMEM with UCX

UCX installation

Requirements: Autoconf 2.63 and above.

1. Get latest version of the UCX code

```
$ git clone https://github.com/openucx/ucx.git ucx
$ cd ucx
```

2. Run autogen:

```
$ ./autogen.sh
```

3. This step is only required for OpenPOWER platforms - Power 8 On Ubuntu platform the config.guess file is a bit outdated and does not have support for power. In order to resolve the issue you have to download an updated config.guess. From the root of the project:

```
$ wget https://github.com/shamisp/ucx/raw/topic/power8-config/config.guess
```

4. Configure:

```
$ mkdir build
$ cd build
$ ../configure --prefix=/your_install_path
```

Note: For best performance configuration, use ../contrib/configure-release. This will strip all debugging and profiling code.

5. Build and install:

```
$ make
$ make install
```

6. Running unit tests (using google test). This only work if gtest was installed and detected on your platform, and –enable-gtest was passed to configure:

   ```
   $ make -C test/gtest test
   ```

2.13.2.5 Interface to ROCm

- https://github.com/openucx/ucx/tree/master/src/uct/rocm

2.13.2.6 Documentation

- Slides
- API documentation (v1.5)

2.13.2.6.1 High Level Design

UCX code consists of 3 parts:

Protocol Layer - UCP
Transport Layer - UCT
Services - UCS

Protocol Layer

Supports all functionality described in the API, and does not require knowledge of particular hardware. It would try to provide best “out-of-box” performance, combining different hardware mechanisms and transports. It may emulate features which are not directly supported in hardware, such as one-sided operations. In addition, it would support common software protocols which are not implemented in hardware, such as tag matching and generic active messages. More details UCP-Design

Transport Layer

Provides direct access to hardware capabilities, without decision logic which would prefer one hardware mechanism over another. Some functionality may not be supported, due to hardware limitations. The capabilities are exposed in the interface. More details UCT-Design

Services

Collection of generic services, data structures, debug aids, etc.

Responsibilities of each layer

What	Where	Why
Tag matching	High level	Software protocol
RMA/AMO emulation	High level	Software protocol
Fragmentation	High level	Software protocol
Pending queue	High level	Stateful
Multi-transport/channel/rail	High level	OOB optimization
Select inline/bcopy/zcopy	High level	optimization logic
Reliability (e.g UD)	Low level	Transport specific
DMA buffer ownership	Low level	Transport specific
Memory registration cache	Low level	Transport dependent

See also:

- sideprogresscompletion
- DesignDiscuss

2.13.2.6.2 Infrastructure and Tools

Tools

- PrintUCXinfo
- findUCPendpoint
- Malloc hooks
- Performancemeasurement
- Testing
- UCXenv

Infrastructure library (UCS)

- Async
- Configuration parsing
- Memoryhooks
- **Data structures:**
  - Double linked list
  - Single linked queue
  - Fragment list - reordering
  - Memory pool

  - Index/Pointer array
  - SGLIB
- **Debugging:**
  - Resolving address to file name and line number
  - Handling faults
  - Attaching a debugger to self
  - logging
  - Assertions (compile-time and run-time)
  - Tracking memory used by different components
  - profiling
- statistic
- **Fast time measurement**
  - Read CPU timer
  - Convert time to sec/msec/usec/nsec
  - Timer queue
  - Timer wheel
- **Data types:**
  - Callback
  - Class infrastructure
  - Component infrastructure
  - Spinlock
  - Error codes
- **System services:**
  - Atomic operations
  - Fast bit operations (find first set bit, integer log2)
  - Get hostname
  - Generate UUID
  - Get CPU affinity
  - Read a whole file
  - Get page / huge page size
  - Allocate memory with SystemV
  - Get memory region access flags (from /proc/$$/maps)
  - Modify file flags with fcntl
  - Get process command line
  - Get CPU model, clock frequency
  - Get thread ID

2.13.2.7 FAQ

What is UCX ?

UCX is a framework (collection of libraries and interfaces) that provides efficient and relatively easy way to construct widely used HPC protocols: MPI tag matching, RMA operations, randevouze protocols, stream, fragmentation, remote atomic operations, etc.

How do I get in touch with UCX developers ?

Please join our mailing list: https://elist.ornl.gov/mailman/listinfo/ucx-group

What is UCP, UCT, UCS

- UCT is a transport layer that abstracts the differences across various hardware architectures and provides a low-level API that enables the implementation of communication protocols. The primary goal of the layer is to provide direct and efficient access to hardware network resources with minimal software overhead. For this purpose UCT relies on low-level drivers provided by vendors such as InfiniBand Verbs, Cray's uGNI, libfabrics, etc. In addition, the layer provides constructs for communication context management (thread-based and ap- plication level), and allocation and management of device- specific memories including those found in accelerators. In terms of communication APIs, UCT defines interfaces for immediate (short), buffered copy-and-send (bcopy), and zero- copy (zcopy) communication operations. The short operations are optimized for small messages that can be posted and completed in place. The bcopy operations are optimized for medium size messages that are typically sent through a so- called bouncing-buffer. Finally, the zcopy operations expose zero-copy memory-to-memory communication semantics.
- UCP implements higher-level protocols that are typically used by message passing (MPI) and PGAS programming models by using lower-level capabilities exposed through the UCT layer. UCP is responsible for the following functionality: initialization of the library, selection of transports for communication, message fragmentation, and multi-rail communication. Currently, the API has the following classes of interfaces: Initialization, Remote Memory Access (RMA) communication, Atomic Memory Operations (AMO), Active Message, Tag-Matching, and Collectives.
- UCS is a service layer that provides the necessary func- tionality for implementing portable and efficient utilities.

What are the key features of UCX ?

- Open source framework supported by vendors The UCX framework is maintained and supported by hardware vendors in addition to the open source community. Every pull-request is tested and multiple hardware platforms supported by vendors community.
- Performance, performance, performance… The framework design, data structures, and components are design to provide highly optimized access to the network hardware.
- High level API for a broad range HPC programming models. UCX provides a high level API implemented in software 'UCP' to fill in the gaps across interconnects. This allows to use a single set of APIs in a library to implement multiple interconnects. This reduces the level of complexities when implementing libraries such as Open MPI or OpenSHMEM. Because of this, UCX performance portable because a single implementation (in Open MPI or OpenSHMEM) will work efficiently on multiple interconnects. (e.g. uGNI, Verbs, libfabrics, etc).
- Support for interaction between multiple transports (or providers) to deliver messages. For example, UCX has the logic (in UCP) to make 'GPUDirect', IB' and share memory work together efficiently to deliver the data where is needed without the user dealing with this.
- Cross-transport multi-rail capabilities

What protocols are supported by UCX ?

UCP implements RMA put/get, send/receive with tag matching, Active messages, atomic operations. In near future we plan to add support for commonly used collective operations.

Is UCX replacement for GASNET ?

No. GASNET exposes high level API for PGAS programming management that provides symmetric memory management capabilities and build in runtime environments. These capabilities are out of scope of UCX project. Instead, GASNET can leverage UCX framework for fast end efficient implementation of GASNET for the network technologies support by UCX.

What is the relation between UCX and network drivers ?

UCX framework does not provide drivers, instead it relies on the drivers provided by vendors. Currently we use: OFA VERBs, Cray's UGNI, NVIDIA CUDA.

What is the relation between UCX and OFA Verbs or Libfabrics ?

UCX, is a middleware communication layer that relies on vendors provided user level drivers including OFA Verbs or libfabrics (or any other drivers provided by another communities or vendors) to implement high-level protocols which can be used to close functionality gaps between various vendors drivers including various libfabrics providers: coordination across various drivers, multi-rail capabilities, software based RMA, AMOs, tag-matching for transports and drivers that do not support such capabilities natively.

Is UCX a user level driver ?

No. Typically, Drivers aim to expose fine-grain access to the network architecture specific features. UCX abstracts the differences across various drivers and fill-in the gaps using software protocols for some of the architectures that don't provide hardware level support for all the operations.

Does UCX depend on an external runtime environment ?

UCX does not depend on an external runtime environment.

ucx_perftest (UCX based application/benchmark) can be linked with an external runtime environment that can be used for remote ucx_perftest launch, but this an optional configuration which is only used for environments that do not provide direct access to compute nodes. By default this option is disabled.

How to install UCX and OpenMPI ?

See How to install UCX and OpenMPI

How can I contribute ?

1.Fork

2.Fix bug or implement a new feature

3.Open Pull Request

2.13.3 MPI

OpenMPI and OpenSHMEM installation

1. Get latest-and-gratest 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/066370202dcad8e302f2baf8921e9efd0f1f7dfc leverages more efficient timer mechanism and there fore 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

MPI/OpenSHMEM	project
OpenMPI/OSHMEM	2.1.0
MPICH	Latest

2.13.4 IPC

2.13.4.1 Introduction

New datatypes

```
hsa_amd_ipc_memory_handle_t

/** IPC memory handle to by 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 by 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.

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,
```

(continues on next page)

(continued from previous page)

```
    /** 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

2.14 Deep Learning on ROCm

2.14.1 TensorFlow

2.14.1.1 ROCm Tensorflow v2.0 Release

We are excited to announce the release of ROCm enabled TensorFlow v2.0 for AMD GPUs. This includes the enhanced compilation toolchain and a set of bug fixes to support TensorFlow 2.0 features natively.

2.14.1.2 Tensorflow Installation

First, you'll need to install the open-source ROCm 2.7 stack. Details can be found here

Then, install these other relevant ROCm packages:

```
sudo apt update
sudo apt install rocm-libs miopen-hip cxlactivitylogger
```

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
```

Now that Tensorflow v2.0 is installed!

2.14.1.3 Tensorflow More Resources

Tensorflow docker images are also publicly available, more details can be found here

The official github repository is here

2.14.2 MIOpen

2.14.2.1 ROCm MIOpen v2.0 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 several new features including an immediate mode for selecting convolutions, bfloat16 support, new layers, modes, and algorithmns.
- 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.
- Source code
- **Documentation**
  - MIOpen
  - MIOpenGemm

2.14.2.2 Porting from cuDNN to MIOpen

The porting guide highlights the key differences between the current cuDNN and MIOpen APIs.

2.14.2.3 The ROCm 2.7 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.

2.14.3 PyTorch

2.14.3.1 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 2.7 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
```

2.14.3.2 Recommended:Install using published PyTorch ROCm docker image:

2. Obtain docker image:

```
docker pull rocm/pytorch:rocm2.7_ubuntu16.04_py2.7
```

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:rocm2.7_ubuntu16.04_py2.7
```

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'
```

2.14.3.3 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'
```

2.14.3.4 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
```

UseMAX_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'
```

2.14.3.5 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

```
cd ../imagenet
```

Follow instructions in README.md.

2.14.4 Caffe2

2.14.4.1 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-2.7s 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.

2.14.4.2 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

```
docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video␣
↪rocm/pytorch:rocm2.7_ubuntu16.04_py2.7_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.

2.14.4.3 Option 2: Install using Caffe2 ROCm docker image:

1. Clone PyTorch repository on the host:

```
cd ~
git clone --recurse-submodules https://github.com/pytorch/pytorch.git
cd pytorch
git submodule update --init --recursive
```

2. Launch the docker container

```
docker pull rocm/pytorch:rocm2.7_ubuntu16.04_py2.7
docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video -
↪v $PWD:/pytorch rocm/pytorch:rocm2.7_ubuntu16.04_py2.7
```

3. Build Caffe2 from source

```
cd /pytorch
```

If running on gfx900/vega10-type GPU(MI25, Vega56, Vega64,. . .)

```
.jenkins/caffe2/build.sh
```

If running on gfx906/vega20-type GPU(MI50, MI60)

```
HCC_AMDGPU_TARGET=gfx906 .jenkins/caffe2/build.sh
```

2.14.4.4 Test the Caffe2 Installation

To validate Caffe2 installation, run

1. Test Command

```
cd ~ && python -c 'from caffe2.python import core' 2>/dev/null && echo "Success" ||␣
↪echo "Failure"
```

2. Running unit tests in Caffe2

```
cd /pytorch
.jenkins/caffe2/test.sh
```

2.14.4.5 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.

```
python caffe2/python/convnet_benchmarks.py --batch_size 64 --model <name_of_the_
↪network> --engine MIOPEN
```

2.14.4.6 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.

2.14.4.7 Building own docker images

After cloning the pytorch repository, you can build your own Caffe2 ROCm docker image. Navigate to pytorch repo and run

```
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.

2.14.5 Deep Learning Framework support for ROCm

Framework	Status	MIOpen Enabled	Upstreamed	Current Repository
Caffe	Public	Yes		https://github.com/ROCmSoftwarePlatform/hipCaffe
Tensorflow	Development	Yes	CLA inProgress	Notes: Working on NCCL and XLA enablement, Running
Caffe2	Upstreaming	Yes	CLA inProgress	
Torch	HIP	Upstreaming	Development inProgress	https://github.com/ROCmSoftwarePlatform/cutorch_hip
PyTorch	Development	Development		
MxNet	Development	Development		https://github.com/ROCmSoftwarePlatform/mxnet
CNTK	Development	Development		

2.14.6 Tutorials

hipCaffe

- caffe

MXNet

- mxnet

2.15 System Level Debug

2.15.1 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

2.15.1.1 ROCr Error Code

- 2 Invalid Dimension
- 4 Invalid Group Memory
- 8 Invalid (or Null) Code
- 32 Invalid Format </li>
- 64 Group is too large
- 128 Out of VGPR’s
- 0x80000000 Debug Trap

2.15.1.2 Command to dump firmware version and get Linux Kernel version

- sudo cat /sys/kernel/debug/dri/1/amdgpu_firmware_info
- uname -a

2.15.1.3 Debug Flags

Debug messages when developing/debugging base ROCm dirver. 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.

2.15.1.4 ROCr level env variable for debug

- HSA_ENABLE_SDMA=0
- HSA_ENABLE_INTERRUPT=0
- HSA_SVM_GUARD_PAGES=0
- HSA_DISABLE_CACHE=1

2.15.1.5 Turn Off Page Retry on GFX9/Vega devices

- sudo –s
- echo 1 > /sys/module/amdkfd/parameters/noretry

2.15.1.6 HCC Debug Enviroment Varibles

HCC_PRINT_ENV=1	will print usage and current values for the HCC and HIP env variables.
HCC_PRINT_ENV = 1	Print values of HCC environment variables
HCC_SERIALIZE_KERNEL= 0	0x1=pre-serialize before each kernel launch, 0x2=post-serialize after each kernel launch,} 0x3=both
HCC_SERIALIZE_COPY= 0	0x1=pre-serialize before each data copy, 0x2=post-serialize after each data copy, 0x3=both
HCC_DB = 0	Enable HCC trace debug
HCC_OPT_FLUSH = 1	Perform system-scope acquire/release only at CPU sync boundaries (rather than after each kernel)
HCC_MAX_QUEUES= 20	Set max number of HSA queues this process will use. accelerator_views will share the allotted queues and steal from each other as necessary
HCC_UNPINNED_COPY_MODE = 2	Select algorithm for unpinned copies. 0=ChooseBest(see thresholds), 1=PinInPlace, 2=StagingBuffer,3=Memcpy
HCC_CHECK_COPY = 0	Check dst == src after each copy operation. Only works on large-bar systems.
HCC_H2D_STAGING_THRESHOLD = 64	Min size (in KB) to use staging buffer algorithm for H2D copy if ChooseBest algorithm selected
HCC_H2D_PININPLACE_THRESHOLD = 4096	Min size (in KB) to use pin-in-place algorithm for H2D copy if ChooseBest algorithm selected
HCC_D2H_PININPLACE_THRESHOLD = 1024	Min size (in KB) to use pin-in-place for D2H copy if ChooseBest algorithm selected
HCC_PROFILE = 0	Enable HCC kernel and data profiling. 1=summary, 2=trace
HCC_PROFILE_VERBOSE = 31	Bitmark to control profile verbosity and format. 0x1=default, 0x2=show begin/end, 0x4=show barrier

2.15.1.7 HIP Environment Varibles

HIP_PRINT_ENV=1	Print HIP environment variables.
HIP_LAUNCH_BLOCKING=0	Make HIP kernel launches 'host-synchronous', so they block until any kernel launches. Alias: CUDA_LAUNCH_BLOCKING
HIP_LAUNCH_BLOCKING_KERNELS=	Comma-separated list of kernel names to make host-synchronous, so they block until completed.
HIP_API_BLOCKING= 0	Make HIP APIs 'host-synchronous', so they block until completed. Impacts hipMemcpyAsync, hipMemsetAsync
HIP_HIDDEN_FREE_MEM= 256	Amount of memory to hide from the free memory reported by hipMemGetInfo, specified in MB.Impacts hipMemGetInfo
HIP_DB = 0	Print debug info. Bitmask (HIP_DB=0xff) or flags separated by '+' (HIP_DB=api+sync+mem+copy)
HIP_TRACE_API=0	Trace each HIP API call. Print function name and return code to stderr as program executes.
HIP_TRACE_API_COLOR= green	Color to use for HIP_API. None/Red/Green/Yellow/Blue/Magenta/Cyan/White
HIP_PROFILE_API = 0	Add HIP API markers to ATP file generated with CodeXL. 0x1=short API name, 0x2=full API name including args
HIP_DB_START_API =	Comma-separated list of tid.api_seq_num for when to start debug and profiling.
HIP_DB_STOP_API =	Comma-separated list of tid.api_seq_num for when to stop debug and profiling.
HIP_VISIBLE_DEVICES = 0	Only devices whose index is present in the sequence are visible to HIP applications and they are enumerated in the order of sequence
HIP_WAIT_MODE = 0	Force synchronization mode. 1= force yield, 2=force spin, 0=defaults specified in application
HIP_FORCE_P2P_HOST = 0	Force use of host/staging copy for peer-to-peer copies.1=always use copies, 2=always return false for hipDeviceCanAccessPeer
HIP_FORCE_SYNC_COPY = 0	Force all copies (even hipMemcpyAsync) to use sync copies
HIP_FAIL_SOC = 0	
	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

2.15.1.8 OpenCL Debug Flags

- AMD_OCL_WAIT_COMMAND=1 (0 = OFF, 1 = On)

2.15.1.9 PCIe-Debug

Refer here for PCIe-Debug

There's some more information here on how to debug and profile HIP applications

- HIP-Debugging
- HIP-Profiling

2.16 Tutorial

- caffe How use Caffe on ROCm
- Vector-Add example ussing the HIP Programing 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.
- rocncloc 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

2.17 ROCm Glossary

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.

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.

PCIe 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 ofeach. 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 Queueing 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.

INDEX

Symbols

_ROCBLAS_AUXILIARY_H_ (*C macro*), 122
_ROCBLAS_FUNCTIONS_H_ (*C macro*), 123
_ROCBLAS_H_ (*C macro*), 141
_ROCBLAS_TYPES_H_ (*C macro*), 138

G

get_rocblas_status_for_hip_status (*C++ function*), 144

M

MAT_BUFF_MAX_BYTES (*C++ member*), 143
MATRIX_DIM_X (*C++ member*), 144
MATRIX_DIM_Y (*C++ member*), 144

N

NB_X (*C++ member*), 143

O

open_log_stream (*C++ function*), 142

R

rocalution::_rocalution_sync (*C++ function*), 176
rocalution::AIChebyshev (*C++ class*), 224
rocalution::AIChebyshev::Set (*C++ function*), 224
rocalution::allocate_host (*C++ function*), 173
rocalution::AS (*C++ class*), 225
rocalution::AS::Set (*C++ function*), 226
rocalution::BaseAMG (*C++ class*), 220
rocalution::BaseAMG::BuildHierarchy (*C++ function*), 220
rocalution::BaseAMG::BuildSmoothers (*C++ function*), 220
rocalution::BaseAMG::ClearLocal (*C++ function*), 220
rocalution::BaseAMG::GetNumLevels (*C++ function*), 221
rocalution::BaseAMG::SetCoarsestLevel (*C++ function*), 220
rocalution::BaseAMG::SetDefaultSmootherFormat (*C++ function*), 220
rocalution::BaseAMG::SetManualSmoothers (*C++ function*), 220
rocalution::BaseAMG::SetManualSolver (*C++ function*), 220
rocalution::BaseAMG::SetOperatorFormat (*C++ function*), 221
rocalution::BaseMultiGrid (*C++ class*), 219
rocalution::BaseMultiGrid::InitLevels (*C++ function*), 219
rocalution::BaseMultiGrid::SetCycle (*C++ function*), 219
rocalution::BaseMultiGrid::SetHostLevels (*C++ function*), 219
rocalution::BaseMultiGrid::SetKcycleFull (*C++ function*), 219
rocalution::BaseMultiGrid::SetOperatorHierarchy (*C++ function*), 219
rocalution::BaseMultiGrid::SetProlongOperator (*C++ function*), 219
rocalution::BaseMultiGrid::SetRestrictOperator (*C++ function*), 219
rocalution::BaseMultiGrid::SetScaling (*C++ function*), 219
rocalution::BaseMultiGrid::SetSmoother (*C++ function*), 219
rocalution::BaseMultiGrid::SetSmootherPostIter (*C++ function*), 219
rocalution::BaseMultiGrid::SetSmootherPreIter (*C++ function*), 219
rocalution::BaseMultiGrid::SetSolver (*C++ function*), 219
rocalution::BaseRocalution (*C++ class*), 176
rocalution::BaseRocalution::Clear (*C++ function*), 177
rocalution::BaseRocalution::CloneBackend (*C++ function*), 177
rocalution::BaseRocalution::Info (*C++ function*), 177
rocalution::BaseRocalution::MoveToAccelerator (*C++ function*), 177

rocalution::BaseRocalution::MoveToAcceleratorAsync (*C++ function*), 177
rocalution::BaseRocalution::MoveToHost (*C++ function*), 177
rocalution::BaseRocalution::MoveToHostAsync (*C++ function*), 177
rocalution::BaseRocalution::Sync (*C++ function*), 177
rocalution::BiCGStab (*C++ class*), 216
rocalution::BiCGStabl (*C++ class*), 216
rocalution::BiCGStabl::SetOrder (*C++ function*), 216
rocalution::BlockJacobi (*C++ class*), 226
rocalution::BlockJacobi::Set (*C++ function*), 226
rocalution::BlockPreconditioner (*C++ class*), 226
rocalution::BlockPreconditioner::Set (*C++ function*), 227
rocalution::BlockPreconditioner::SetDiagonalSolver (*C++ function*), 227
rocalution::BlockPreconditioner::SetExternalLastMatrix (*C++ function*), 227
rocalution::BlockPreconditioner::SetLSolver (*C++ function*), 227
rocalution::BlockPreconditioner::SetPermutation (*C++ function*), 227
rocalution::CG (*C++ class*), 216
rocalution::Chebyshev (*C++ class*), 215
rocalution::Chebyshev::Set (*C++ function*), 216
rocalution::CR (*C++ class*), 217
rocalution::DiagJacobiSaddlePointPrecond (*C++ class*), 231
rocalution::DiagJacobiSaddlePointPrecond::Set (*C++ function*), 232
rocalution::DirectLinearSolver (*C++ class*), 222
rocalution::disable_accelerator_rocalution (*C++ function*), 176
rocalution::FCG (*C++ class*), 217
rocalution::FGMRES (*C++ class*), 218
rocalution::FGMRES::SetBasisSize (*C++ function*), 218
rocalution::FixedPoint (*C++ class*), 215
rocalution::FixedPoint::SetRelaxation (*C++ function*), 215
rocalution::free_host (*C++ function*), 174
rocalution::FSAI (*C++ class*), 224
rocalution::FSAI::Set (*C++ function*), 225
rocalution::FSAI::SetPrecondMatrixFormat (*C++ function*), 225
rocalution::GlobalMatrix (*C++ class*), 204
rocalution::GlobalMatrix::AllocateCOO (*C++ function*), 204
rocalution::GlobalMatrix::AllocateCSR (*C++ function*), 204
rocalution::GlobalMatrix::Check (*C++ function*), 204
rocalution::GlobalMatrix::CloneFrom (*C++ function*), 205
rocalution::GlobalMatrix::CoarsenOperator (*C++ function*), 206
rocalution::GlobalMatrix::ConvertTo (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToBCSR (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToCOO (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToCSR (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToDENSE (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToDIA (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToELL (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToHYB (*C++ function*), 205
rocalution::GlobalMatrix::ConvertToMCSR (*C++ function*), 205
rocalution::GlobalMatrix::CopyFrom (*C++ function*), 205
rocalution::GlobalMatrix::ExtractInverseDiagonal (*C++ function*), 206
rocalution::GlobalMatrix::FurtherPairwiseAggregatio (*C++ function*), 206
rocalution::GlobalMatrix::GlobalMatrix (*C++ function*), 204
rocalution::GlobalMatrix::InitialPairwiseAggregatio (*C++ function*), 206
rocalution::GlobalMatrix::LeaveDataPtrCOO (*C++ function*), 205
rocalution::GlobalMatrix::LeaveDataPtrCSR (*C++ function*), 204
rocalution::GlobalMatrix::LeaveGhostDataPtrCOO (*C++ function*), 205
rocalution::GlobalMatrix::LeaveGhostDataPtrCSR (*C++ function*), 205
rocalution::GlobalMatrix::LeaveLocalDataPtrCOO (*C++ function*), 205
rocalution::GlobalMatrix::LeaveLocalDataPtrCSR (*C++ function*), 205
rocalution::GlobalMatrix::ReadFileCSR (*C++ function*), 205
rocalution::GlobalMatrix::ReadFileMTX (*C++ function*), 205
rocalution::GlobalMatrix::Scale (*C++*

function), 206
rocalution::GlobalMatrix::SetDataPtrCOO (*C++ function*), 204
rocalution::GlobalMatrix::SetDataPtrCSR (*C++ function*), 204
rocalution::GlobalMatrix::SetGhostDataPtrCOO (*C++ function*), 204
rocalution::GlobalMatrix::SetGhostDataPtrCSR (*C++ function*), 204
rocalution::GlobalMatrix::SetLocalDataPtrCOO (*C++ function*), 204
rocalution::GlobalMatrix::SetLocalDataPtrCSR (*C++ function*), 204
rocalution::GlobalMatrix::SetParallelManager (*C++ function*), 204
rocalution::GlobalMatrix::Sort (*C++ function*), 206
rocalution::GlobalMatrix::WriteFileCSR (*C++ function*), 205
rocalution::GlobalMatrix::WriteFileMTX (*C++ function*), 205
rocalution::GlobalPairwiseAMG (*C++ class*), 222
rocalution::GlobalPairwiseAMG::SetBeta (*C++ function*), 222
rocalution::GlobalPairwiseAMG::SetCoarseningFactor (*C++ function*), 222
rocalution::GlobalPairwiseAMG::SetOrdering (*C++ function*), 222
rocalution::GlobalVector (*C++ class*), 209
rocalution::GlobalVector::Allocate (*C++ function*), 210
rocalution::GlobalVector::GlobalVector (*C++ function*), 210
rocalution::GlobalVector::LeaveDataPtr (*C++ function*), 210
rocalution::GlobalVector::operator[] (*C++ function*), 210
rocalution::GlobalVector::Prolongation (*C++ function*), 210
rocalution::GlobalVector::Restriction (*C++ function*), 210
rocalution::GlobalVector::SetDataPtr (*C++ function*), 210
rocalution::GlobalVector::SetParallelManager (*C++ function*), 210
rocalution::GMRES (*C++ class*), 217
rocalution::GMRES::SetBasisSize (*C++ function*), 217
rocalution::GS (*C++ class*), 227
rocalution::IC (*C++ class*), 229
rocalution::IDR (*C++ class*), 218
rocalution::IDR::SetRandomSeed (*C++ function*), 218
rocalution::IDR::SetShadowSpace (*C++ function*), 218
rocalution::ILU (*C++ class*), 228
rocalution::ILU::Set (*C++ function*), 228
rocalution::ILUT (*C++ class*), 228
rocalution::ILUT::Set (*C++ function*), 229
rocalution::info_rocalution (*C++ function*), 176
rocalution::init_rocalution (*C++ function*), 174
rocalution::Inversion (*C++ class*), 223
rocalution::IterativeLinearSolver (*C++ class*), 213
rocalution::IterativeLinearSolver::GetAmaxResidualI (*C++ function*), 214
rocalution::IterativeLinearSolver::GetCurrentResidu (*C++ function*), 214
rocalution::IterativeLinearSolver::GetIterationCoun (*C++ function*), 214
rocalution::IterativeLinearSolver::GetSolverStatus (*C++ function*), 214
rocalution::IterativeLinearSolver::Init (*C++ function*), 214
rocalution::IterativeLinearSolver::InitMaxIter (*C++ function*), 214
rocalution::IterativeLinearSolver::InitMinIter (*C++ function*), 214
rocalution::IterativeLinearSolver::InitTol (*C++ function*), 214
rocalution::IterativeLinearSolver::RecordHistory (*C++ function*), 214
rocalution::IterativeLinearSolver::RecordResidualHi (*C++ function*), 214
rocalution::IterativeLinearSolver::SetPreconditione (*C++ function*), 214
rocalution::IterativeLinearSolver::SetResidualNorm (*C++ function*), 214
rocalution::IterativeLinearSolver::Solve (*C++ function*), 214
rocalution::IterativeLinearSolver::Verbose (*C++ function*), 214
rocalution::Jacobi (*C++ class*), 227
rocalution::LocalMatrix (*C++ class*), 185
rocalution::LocalMatrix::AddScalar (*C++ function*), 193
rocalution::LocalMatrix::AddScalarDiagonal (*C++ function*), 193
rocalution::LocalMatrix::AddScalarOffDiagonal (*C++ function*), 193
rocalution::LocalMatrix::AllocateBCSR (*C++ function*), 185
rocalution::LocalMatrix::AllocateCOO (*C++ function*), 186
rocalution::LocalMatrix::AllocateCSR

(*C++ function*), 185
rocalution::LocalMatrix::AllocateDENSE (*C++ function*), 187
rocalution::LocalMatrix::AllocateDIA (*C++ function*), 186
rocalution::LocalMatrix::AllocateELL (*C++ function*), 187
rocalution::LocalMatrix::AllocateHYB (*C++ function*), 187
rocalution::LocalMatrix::AllocateMCSR (*C++ function*), 186
rocalution::LocalMatrix::AMGAggregate (*C++ function*), 202
rocalution::LocalMatrix::AMGAggregation (*C++ function*), 202
rocalution::LocalMatrix::AMGConnect (*C++ function*), 202
rocalution::LocalMatrix::AMGSmoothedAggregation (*C++ function*), 202
rocalution::LocalMatrix::Check (*C++ function*), 185
rocalution::LocalMatrix::CloneFrom (*C++ function*), 199
rocalution::LocalMatrix::CMK (*C++ function*), 194
rocalution::LocalMatrix::CoarsenOperator (*C++ function*), 203
rocalution::LocalMatrix::Compress (*C++ function*), 201
rocalution::LocalMatrix::ConnectivityOrder (*C++ function*), 195
rocalution::LocalMatrix::ConvertTo (*C++ function*), 201
rocalution::LocalMatrix::ConvertToBCSR (*C++ function*), 200
rocalution::LocalMatrix::ConvertToCOO (*C++ function*), 200
rocalution::LocalMatrix::ConvertToCSR (*C++ function*), 200
rocalution::LocalMatrix::ConvertToDENSE (*C++ function*), 201
rocalution::LocalMatrix::ConvertToDIA (*C++ function*), 201
rocalution::LocalMatrix::ConvertToELL (*C++ function*), 200
rocalution::LocalMatrix::ConvertToHYB (*C++ function*), 201
rocalution::LocalMatrix::ConvertToMCSR (*C++ function*), 200
rocalution::LocalMatrix::CopyFrom (*C++ function*), 199
rocalution::LocalMatrix::CopyFromAsync (*C++ function*), 199
rocalution::LocalMatrix::CopyFromCOO (*C++ function*), 200
rocalution::LocalMatrix::CopyFromCSR (*C++ function*), 200
rocalution::LocalMatrix::CopyFromHostCSR (*C++ function*), 200
rocalution::LocalMatrix::CopyToCOO (*C++ function*), 200
rocalution::LocalMatrix::CopyToCSR (*C++ function*), 200
rocalution::LocalMatrix::CreateFromMap (*C++ function*), 200
rocalution::LocalMatrix::DiagonalMatrixMult (*C++ function*), 201
rocalution::LocalMatrix::DiagonalMatrixMultL (*C++ function*), 201
rocalution::LocalMatrix::DiagonalMatrixMultR (*C++ function*), 201
rocalution::LocalMatrix::ExtractColumnVector (*C++ function*), 202
rocalution::LocalMatrix::ExtractDiagonal (*C++ function*), 194
rocalution::LocalMatrix::ExtractInverseDiagonal (*C++ function*), 194
rocalution::LocalMatrix::ExtractL (*C++ function*), 194
rocalution::LocalMatrix::ExtractRowVector (*C++ function*), 202
rocalution::LocalMatrix::ExtractSubMatrices (*C++ function*), 194
rocalution::LocalMatrix::ExtractSubMatrix (*C++ function*), 193
rocalution::LocalMatrix::ExtractU (*C++ function*), 194
rocalution::LocalMatrix::FSAI (*C++ function*), 202
rocalution::LocalMatrix::FurtherPairwiseAggregatio
(*C++ function*), 203
rocalution::LocalMatrix::Gershgorin (*C++ function*), 201
rocalution::LocalMatrix::GetFormat (*C++ function*), 185
rocalution::LocalMatrix::Householder (*C++ function*), 197
rocalution::LocalMatrix::ICFactorize (*C++ function*), 196
rocalution::LocalMatrix::ILU0Factorize (*C++ function*), 196
rocalution::LocalMatrix::ILUpFactorize (*C++ function*), 196
rocalution::LocalMatrix::ILUTFactorize (*C++ function*), 196
rocalution::LocalMatrix::InitialPairwiseAggregatio
(*C++ function*), 202, 203
rocalution::LocalMatrix::Invert (*C++*

function), 197

rocalution::LocalMatrix::Key (*C++ function*), 201

rocalution::LocalMatrix::LAnalyse (*C++ function*), 196

rocalution::LocalMatrix::LAnalyseClear (*C++ function*), 197

rocalution::LocalMatrix::LeaveDataPtrCOO (*C++ function*), 190

rocalution::LocalMatrix::LeaveDataPtrCSR (*C++ function*), 191

rocalution::LocalMatrix::LeaveDataPtrDENSE (*C++ function*), 193

rocalution::LocalMatrix::LeaveDataPtrDIA (*C++ function*), 192

rocalution::LocalMatrix::LeaveDataPtrELL (*C++ function*), 192

rocalution::LocalMatrix::LeaveDataPtrMCSR (*C++ function*), 191

rocalution::LocalMatrix::LLAnalyse (*C++ function*), 196

rocalution::LocalMatrix::LLAnalyseClear (*C++ function*), 196

rocalution::LocalMatrix::LLSolve (*C++ function*), 196

rocalution::LocalMatrix::LSolve (*C++ function*), 197

rocalution::LocalMatrix::LUAnalyse (*C++ function*), 196

rocalution::LocalMatrix::LUAnalyseClear (*C++ function*), 196

rocalution::LocalMatrix::LUFactorize (*C++ function*), 196

rocalution::LocalMatrix::LUSolve (*C++ function*), 196

rocalution::LocalMatrix::MatrixAdd (*C++ function*), 201

rocalution::LocalMatrix::MatrixMult (*C++ function*), 201

rocalution::LocalMatrix::MaximalIndependentSet (*C++ function*), 195

rocalution::LocalMatrix::MultiColoring (*C++ function*), 195

rocalution::LocalMatrix::Permute (*C++ function*), 194

rocalution::LocalMatrix::PermuteBackward (*C++ function*), 194

rocalution::LocalMatrix::QRDecompose (*C++ function*), 197

rocalution::LocalMatrix::QRSolve (*C++ function*), 197

rocalution::LocalMatrix::RCMK (*C++ function*), 194

rocalution::LocalMatrix::ReadFileCSR (*C++ function*), 198

rocalution::LocalMatrix::ReadFileMTX (*C++ function*), 197

rocalution::LocalMatrix::ReplaceColumnVector (*C++ function*), 202

rocalution::LocalMatrix::ReplaceRowVector (*C++ function*), 202

rocalution::LocalMatrix::RugeStueben (*C++ function*), 202

rocalution::LocalMatrix::Scale (*C++ function*), 193

rocalution::LocalMatrix::ScaleDiagonal (*C++ function*), 193

rocalution::LocalMatrix::ScaleOffDiagonal (*C++ function*), 193

rocalution::LocalMatrix::SetDataPtrCOO (*C++ function*), 188

rocalution::LocalMatrix::SetDataPtrCSR (*C++ function*), 188

rocalution::LocalMatrix::SetDataPtrDENSE (*C++ function*), 190

rocalution::LocalMatrix::SetDataPtrDIA (*C++ function*), 190

rocalution::LocalMatrix::SetDataPtrELL (*C++ function*), 189

rocalution::LocalMatrix::SetDataPtrMCSR (*C++ function*), 189

rocalution::LocalMatrix::Sort (*C++ function*), 201

rocalution::LocalMatrix::SPAI (*C++ function*), 202

rocalution::LocalMatrix::SymbolicPower (*C++ function*), 201

rocalution::LocalMatrix::Transpose (*C++ function*), 201

rocalution::LocalMatrix::UAnalyse (*C++ function*), 197

rocalution::LocalMatrix::UAnalyseClear (*C++ function*), 197

rocalution::LocalMatrix::UpdateValuesCSR (*C++ function*), 199

rocalution::LocalMatrix::USolve (*C++ function*), 197

rocalution::LocalMatrix::WriteFileCSR (*C++ function*), 198

rocalution::LocalMatrix::WriteFileMTX (*C++ function*), 197

rocalution::LocalMatrix::ZeroBlockPermutation (*C++ function*), 196

rocalution::LocalMatrix::Zeros (*C++ function*), 193

rocalution::LocalStencil (*C++ class*), 203

rocalution::LocalStencil::GetNDim (*C++ function*), 203 rocalution::LocalStencil::LocalStencil (*C++ function*), 203

rocalution::LocalStencil::SetGrid (*C++ function*), 203

rocalution::LocalVector (*C++ class*), 206

rocalution::LocalVector::Allocate (*C++ function*), 206

rocalution::LocalVector::CopyFromData (*C++ function*), 208

rocalution::LocalVector::CopyFromPermute (*C++ function*), 208

rocalution::LocalVector::CopyFromPermuteBackward (*C++ function*), 208

rocalution::LocalVector::CopyToData (*C++ function*), 209

rocalution::LocalVector::ExtractCoarseBoundary (*C++ function*), 209

rocalution::LocalVector::ExtractCoarseMapping (*C++ function*), 209

rocalution::LocalVector::GetContinuousValues (*C++ function*), 209

rocalution::LocalVector::GetIndexValues (*C++ function*), 209

rocalution::LocalVector::LeaveDataPtr (*C++ function*), 207

rocalution::LocalVector::operator[] (*C++ function*), 207, 208

rocalution::LocalVector::Permute (*C++ function*), 209

rocalution::LocalVector::PermuteBackward (*C++ function*), 209

rocalution::LocalVector::Prolongation (*C++ function*), 209

rocalution::LocalVector::Restriction (*C++ function*), 209

rocalution::LocalVector::SetContinuousValues (*C++ function*), 209

rocalution::LocalVector::SetDataPtr (*C++ function*), 207

rocalution::LocalVector::SetIndexArray (*C++ function*), 209

rocalution::LocalVector::SetIndexValues (*C++ function*), 209

rocalution::LU (*C++ class*), 223

rocalution::MixedPrecisionDC (*C++ class*), 215

rocalution::MixedPrecisionDC::Set (*C++ function*), 215

rocalution::MultiColored (*C++ class*), 229

rocalution::MultiColored::SetDecomposition (*C++ function*), 230

rocalution::MultiColored::SetPrecondMatrixFormat (*C++ function*), 230

rocalution::MultiColoredGS (*C++ class*), 230

rocalution::MultiColoredILU (*C++ class*), 230

rocalution::MultiColoredILU::Set (*C++ function*), 230, 231

rocalution::MultiColoredSGS (*C++ class*), 230

rocalution::MultiColoredSGS::SetRelaxation (*C++ function*), 230

rocalution::MultiElimination (*C++ class*), 231

rocalution::MultiElimination::GetLevel (*C++ function*), 231

rocalution::MultiElimination::GetSizeDiagBlock (*C++ function*), 231

rocalution::MultiElimination::Set (*C++ function*), 231

rocalution::MultiElimination::SetPrecondMatrixForma (*C++ function*), 231

rocalution::MultiGrid (*C++ class*), 219

rocalution::Operator (*C++ class*), 178

rocalution::Operator::Apply (*C++ function*), 178

rocalution::Operator::ApplyAdd (*C++ function*), 178

rocalution::Operator::GetGhostM (*C++ function*), 178

rocalution::Operator::GetGhostN (*C++ function*), 178

rocalution::Operator::GetGhostNnz (*C++ function*), 178

rocalution::Operator::GetLocalM (*C++ function*), 178

rocalution::Operator::GetLocalN (*C++ function*), 178

rocalution::Operator::GetLocalNnz (*C++ function*), 178

rocalution::Operator::GetM (*C++ function*), 178

rocalution::Operator::GetN (*C++ function*), 178

rocalution::Operator::GetNnz (*C++ function*), 178

rocalution::PairwiseAMG (*C++ class*), 222

rocalution::PairwiseAMG::SetBeta (*C++ function*), 222

rocalution::PairwiseAMG::SetCoarseningFactor (*C++ function*), 222

rocalution::PairwiseAMG::SetOrdering (*C++ function*), 222

rocalution::ParallelManager (*C++ class*), 210

rocalution::ParallelManager::Clear (*C++ function*), 210

rocalution::ParallelManager::GetGlobalSize

(*C++ function*), 210
rocalution::ParallelManager::GetLocalSize (*C++ function*), 210
rocalution::ParallelManager::GetNumProcs (*C++ function*), 211
rocalution::ParallelManager::GetNumReceivers (*C++ function*), 211
rocalution::ParallelManager::GetNumSenders (*C++ function*), 211
rocalution::ParallelManager::GlobalToLocal (*C++ function*), 211
rocalution::ParallelManager::LocalToGlobal (*C++ function*), 211
rocalution::ParallelManager::ReadFileASCII (*C++ function*), 211
rocalution::ParallelManager::SetBoundaryIndex (*C++ function*), 211
rocalution::ParallelManager::SetGlobalSize (*C++ function*), 211
rocalution::ParallelManager::SetLocalSize (*C++ function*), 211
rocalution::ParallelManager::SetMPICommunicator (*C++ function*), 210
rocalution::ParallelManager::SetReceivers (*C++ function*), 211
rocalution::ParallelManager::SetSenders (*C++ function*), 211
rocalution::ParallelManager::Status (*C++ function*), 211
rocalution::ParallelManager::WriteFileASCII (*C++ function*), 211
rocalution::Preconditioner (*C++ class*), 224
rocalution::QMRCGStab (*C++ class*), 218
rocalution::QR (*C++ class*), 223
rocalution::RAS (*C++ class*), 226
rocalution::rocalution_time (*C++ function*), 174
rocalution::RugeStuebenAMG (*C++ class*), 221
rocalution::RugeStuebenAMG::SetCouplingStrength (*C++ function*), 222
rocalution::SAAMG (*C++ class*), 221
rocalution::SAAMG::SetCouplingStrength (*C++ function*), 221
rocalution::SAAMG::SetInterpRelax (*C++ function*), 221
rocalution::set_device_rocalution (*C++ function*), 175
rocalution::set_omp_affinity_rocalution (*C++ function*), 176
rocalution::set_omp_threads_rocalution (*C++ function*), 175
rocalution::set_omp_threshold_rocalution (*C++ function*), 176
rocalution::set_to_zero_host (*C++ function*), 174
rocalution::SGS (*C++ class*), 228
rocalution::Solver (*C++ class*), 211
rocalution::Solver::Build (*C++ function*), 212
rocalution::Solver::BuildMoveToAcceleratorAsync (*C++ function*), 212
rocalution::Solver::Clear (*C++ function*), 212
rocalution::Solver::MoveToAccelerator (*C++ function*), 212
rocalution::Solver::MoveToHost (*C++ function*), 212
rocalution::Solver::Print (*C++ function*), 212
rocalution::Solver::ReBuildNumeric (*C++ function*), 212
rocalution::Solver::ResetOperator (*C++ function*), 212
rocalution::Solver::SetOperator (*C++ function*), 212
rocalution::Solver::Solve (*C++ function*), 212
rocalution::Solver::SolveZeroSol (*C++ function*), 212
rocalution::Solver::Sync (*C++ function*), 212
rocalution::Solver::Verbose (*C++ function*), 212
rocalution::SPAI (*C++ class*), 225
rocalution::SPAI::SetPrecondMatrixFormat (*C++ function*), 225
rocalution::stop_rocalution (*C++ function*), 175
rocalution::TNS (*C++ class*), 225
rocalution::TNS::Set (*C++ function*), 225
rocalution::TNS::SetPrecondMatrixFormat (*C++ function*), 225
rocalution::UAAMG (*C++ class*), 221
rocalution::UAAMG::SetCouplingStrength (*C++ function*), 221
rocalution::UAAMG::SetOverInterp (*C++ function*), 221
rocalution::VariablePreconditioner (*C++ class*), 229
rocalution::VariablePreconditioner::SetPreconditio
(*C++ function*), 229
rocalution::Vector (*C++ class*), 179
rocalution::Vector::AddScale (*C++ function*), 183
rocalution::Vector::Amax (*C++ function*), 184
rocalution::Vector::Asum (*C++ function*), 184
rocalution::Vector::Check (*C++ function*), 179
rocalution::Vector::CloneFrom (*C++

function), 182, 183
rocalution::Vector::CopyFrom (*C++ function*), 181, 182
rocalution::Vector::CopyFromAsync (*C++ function*), 182
rocalution::Vector::CopyFromDouble (*C++ function*), 182
rocalution::Vector::CopyFromFloat (*C++ function*), 182
rocalution::Vector::Dot (*C++ function*), 184
rocalution::Vector::DotNonConj (*C++ function*), 184
rocalution::Vector::GetGhostSize (*C++ function*), 179
rocalution::Vector::GetLocalSize (*C++ function*), 179
rocalution::Vector::GetSize (*C++ function*), 179
rocalution::Vector::Norm (*C++ function*), 184
rocalution::Vector::Ones (*C++ function*), 179
rocalution::Vector::PointWiseMult (*C++ function*), 184
rocalution::Vector::Power (*C++ function*), 184
rocalution::Vector::ReadFileASCII (*C++ function*), 179
rocalution::Vector::ReadFileBinary (*C++ function*), 180
rocalution::Vector::Reduce (*C++ function*), 184
rocalution::Vector::Scale (*C++ function*), 184
rocalution::Vector::ScaleAdd (*C++ function*), 183
rocalution::Vector::ScaleAdd2 (*C++ function*), 184
rocalution::Vector::ScaleAddScale (*C++ function*), 183
rocalution::Vector::SetRandomNormal (*C++ function*), 179
rocalution::Vector::SetRandomUniform (*C++ function*), 179
rocalution::Vector::SetValues (*C++ function*), 179
rocalution::Vector::WriteFileASCII (*C++ function*), 180
rocalution::Vector::WriteFileBinary (*C++ function*), 180
rocalution::Vector::Zeros (*C++ function*), 179
rocblas (*C++ type*), 122
rocblas::reinit_logs (*C++ function*), 122
rocblas_add_stream (*C++ function*), 121, 123
rocblas_create_handle (*C++ function*), 121, 123, 143
rocblas_dasum (*C++ function*), 108, 126
rocblas_datatype (*C++ enum*), 104, 140
rocblas_datatype_f16_c (*C++ enumerator*), 104, 140
rocblas_datatype_f16_r (*C++ enumerator*), 104, 140
rocblas_datatype_f32_c (*C++ enumerator*), 104, 140
rocblas_datatype_f32_r (*C++ enumerator*), 104, 140
rocblas_datatype_f64_c (*C++ enumerator*), 104, 140
rocblas_datatype_f64_r (*C++ enumerator*), 104, 140
rocblas_datatype_i32_c (*C++ enumerator*), 105, 140
rocblas_datatype_i32_r (*C++ enumerator*), 104, 140
rocblas_datatype_i8_c (*C++ enumerator*), 105, 140
rocblas_datatype_i8_r (*C++ enumerator*), 104, 140
rocblas_datatype_u32_c (*C++ enumerator*), 105, 140
rocblas_datatype_u32_r (*C++ enumerator*), 104, 140
rocblas_datatype_u8_c (*C++ enumerator*), 105, 140
rocblas_datatype_u8_r (*C++ enumerator*), 104, 140
rocblas_daxpy (*C++ function*), 107, 125
rocblas_dcopy (*C++ function*), 106, 124
rocblas_ddot (*C++ function*), 106, 124
rocblas_destroy_handle (*C++ function*), 121, 123, 143
rocblas_dgeam (*C++ function*), 117, 135
rocblas_dgemm (*C++ function*), 114, 132
rocblas_dgemm_kernel_name (*C++ function*), 117, 134
rocblas_dgemm_strided_batched (*C++ function*), 115, 134
rocblas_dgemv (*C++ function*), 109, 127
rocblas_dger (*C++ function*), 111, 129
rocblas_diagonal (*C++ enum*), 103, 139
rocblas_diagonal_non_unit (*C++ enumerator*), 103, 139
rocblas_diagonal_unit (*C++ enumerator*), 103, 139
rocblas_dnrm2 (*C++ function*), 108, 126
rocblas_double_complex (*C++ type*), 102, 138
rocblas_dscal (*C++ function*), 105, 124
rocblas_dswap (*C++ function*), 107, 125
rocblas_dsyr (*C++ function*), 112, 129

rocblas_dtrsm (*C++ function*), 113, 131
rocblas_dtrsv (*C++ function*), 110, 128
rocblas_dtrtri (*C++ function*), 130
rocblas_dtrtri_batched (*C++ function*), 113, 130
rocblas_fill (*C++ enum*), 103, 139
rocblas_fill_full (*C++ enumerator*), 103, 139
rocblas_fill_lower (*C++ enumerator*), 103, 139
rocblas_fill_upper (*C++ enumerator*), 103, 139
rocblas_float_complex (*C++ type*), 102, 138
rocblas_gemm_algo (*C++ enum*), 105, 141
rocblas_gemm_algo_standard (*C++ enumerator*), 105, 141
rocblas_gemm_ex (*C++ function*), 118, 135
rocblas_gemm_strided_batched_ex (*C++ function*), 118, 135
rocblas_get_matrix (*C++ function*), 122, 123, 143
rocblas_get_pointer_mode (*C++ function*), 122, 123, 143
rocblas_get_stream (*C++ function*), 121, 123, 143
rocblas_get_vector (*C++ function*), 122, 123, 143
rocblas_get_version_string (*C++ function*), 120, 137, 141
rocblas_half (*C++ type*), 102, 138
rocblas_half_complex (*C++ type*), 102, 138
rocblas_handle (*C++ type*), 102, 138
rocblas_haxpy (*C++ function*), 107, 125
rocblas_hgemm (*C++ function*), 114, 131
rocblas_hgemm_kernel_name (*C++ function*), 117, 134
rocblas_hgemm_strided_batched (*C++ function*), 115, 132
rocblas_idamax (*C++ function*), 109, 126
rocblas_idamin (*C++ function*), 109, 127
rocblas_int (*C++ type*), 102, 138
rocblas_isamax (*C++ function*), 109, 126
rocblas_isamin (*C++ function*), 109, 126
rocblas_layer_mode (*C++ enum*), 105, 140
rocblas_layer_mode_log_bench (*C++ enumerator*), 105, 140
rocblas_layer_mode_log_profile (*C++ enumerator*), 105, 141
rocblas_layer_mode_log_trace (*C++ enumerator*), 105, 140
rocblas_layer_mode_none (*C++ enumerator*), 105, 140
rocblas_long (*C++ type*), 102, 138
rocblas_operation (*C++ enum*), 103, 139
rocblas_operation_conjugate_transpose (*C++ enumerator*), 103, 139
rocblas_operation_none (*C++ enumerator*), 103, 139
rocblas_operation_transpose (*C++ enumerator*), 103, 139
rocblas_pointer_mode (*C++ enum*), 105, 140
rocblas_pointer_mode_device (*C++ enumerator*), 105, 140
rocblas_pointer_mode_host (*C++ enumerator*), 105, 140
rocblas_pointer_to_mode (*C++ function*), 121, 123, 143
rocblas_sasum (*C++ function*), 108, 125
rocblas_saxpy (*C++ function*), 107, 125
rocblas_scopy (*C++ function*), 106, 124
rocblas_sdot (*C++ function*), 106, 124
rocblas_set_matrix (*C++ function*), 122, 123, 143
rocblas_set_pointer_mode (*C++ function*), 122, 123, 143
rocblas_set_stream (*C++ function*), 121, 123, 143
rocblas_set_vector (*C++ function*), 122, 123, 143
rocblas_sgeam (*C++ function*), 117, 134
rocblas_sgemm (*C++ function*), 114, 132
rocblas_sgemm_kernel_name (*C++ function*), 117, 134
rocblas_sgemm_strided_batched (*C++ function*), 115, 134
rocblas_sgemv (*C++ function*), 109, 127
rocblas_sger (*C++ function*), 111, 128
rocblas_side (*C++ enum*), 103, 139
rocblas_side_both (*C++ enumerator*), 104, 139
rocblas_side_left (*C++ enumerator*), 104, 139
rocblas_side_right (*C++ enumerator*), 104, 139
rocblas_snrm2 (*C++ function*), 108, 126
rocblas_sscal (*C++ function*), 105, 123
rocblas_sswap (*C++ function*), 107, 124
rocblas_ssyr (*C++ function*), 112, 129
rocblas_status (*C++ enum*), 104, 139
rocblas_status_internal_error (*C++ enumerator*), 104, 140
rocblas_status_invalid_handle (*C++ enumerator*), 104, 139
rocblas_status_invalid_pointer (*C++ enumerator*), 104, 140
rocblas_status_invalid_size (*C++ enumerator*), 104, 140
rocblas_status_memory_error (*C++ enumerator*), 104, 140
rocblas_status_not_implemented (*C++ enumerator*), 104, 140
rocblas_status_success (*C++ enumerator*), 104, 139
rocblas_strsm (*C++ function*), 113, 131

rocblas_strsv (*C++ function*), 110, 127
rocblas_strtri (*C++ function*), 129
rocblas_strtri_batched (*C++ function*), 113, 130
rocfft_array_type (*C++ enum*), 100
rocfft_array_type_complex_interleaved (*C++ enumerator*), 100
rocfft_array_type_complex_planar (*C++ enumerator*), 100
rocfft_array_type_hermitian_interleaved (*C++ enumerator*), 100
rocfft_array_type_hermitian_planar (*C++ enumerator*), 100
rocfft_array_type_real (*C++ enumerator*), 100
rocfft_cleanup (*C++ function*), 95
rocfft_exec_mode_blocking (*C++ enumerator*), 100
rocfft_exec_mode_nonblocking (*C++ enumerator*), 100
rocfft_exec_mode_nonblocking_with_flush (*C++ enumerator*), 100
rocfft_execute (*C++ function*), 97
rocfft_execution_info (*C++ type*), 95
rocfft_execution_info_create (*C++ function*), 98
rocfft_execution_info_destroy (*C++ function*), 98
rocfft_execution_info_set_stream (*C++ function*), 98
rocfft_execution_info_set_work_buffer (*C++ function*), 98
rocfft_execution_mode (*C++ enum*), 100
rocfft_placement_inplace (*C++ enumerator*), 99
rocfft_placement_notinplace (*C++ enumerator*), 100
rocfft_plan (*C++ type*), 94
rocfft_plan_create (*C++ function*), 95
rocfft_plan_description (*C++ type*), 95
rocfft_plan_description_create (*C++ function*), 96
rocfft_plan_description_destroy (*C++ function*), 96
rocfft_plan_description_set_data_layout (*C++ function*), 96
rocfft_plan_destroy (*C++ function*), 96
rocfft_plan_get_print (*C++ function*), 96
rocfft_plan_get_work_buffer_size (*C++ function*), 96
rocfft_precision (*C++ enum*), 99
rocfft_precision_double (*C++ enumerator*), 99
rocfft_precision_single (*C++ enumerator*), 99
rocfft_result_placement (*C++ enum*), 99
rocfft_setup (*C++ function*), 95
rocfft_status (*C++ enum*), 99
rocfft_status_failure (*C++ enumerator*), 99
rocfft_status_invalid_arg_value (*C++ enumerator*), 99
rocfft_status_invalid_array_type (*C++ enumerator*), 99
rocfft_status_invalid_dimensions (*C++ enumerator*), 99
rocfft_status_invalid_distance (*C++ enumerator*), 99
rocfft_status_invalid_offset (*C++ enumerator*), 99
rocfft_status_invalid_strides (*C++ enumerator*), 99
rocfft_status_success (*C++ enumerator*), 99
rocfft_transform_type (*C++ enum*), 99
rocfft_transform_type_complex_forward (*C++ enumerator*), 99
rocfft_transform_type_complex_inverse (*C++ enumerator*), 99
rocfft_transform_type_real_forward (*C++ enumerator*), 99
rocfft_transform_type_real_inverse (*C++ enumerator*), 99
rocsparse_action (*C++ enum*), 238
rocsparse_action_numeric (*C++ enumerator*), 238
rocsparse_action_symbolic (*C++ enumerator*), 238
rocsparse_analysis_policy (*C++ enum*), 241
rocsparse_analysis_policy_force (*C++ enumerator*), 241
rocsparse_analysis_policy_reuse (*C++ enumerator*), 241
rocsparse_coo2csr (*C++ function*), 282
rocsparse_coosort_buffer_size (*C++ function*), 295
rocsparse_coosort_by_column (*C++ function*), 298
rocsparse_coosort_by_row (*C++ function*), 296
rocsparse_copy_mat_descr (*C++ function*), 247
rocsparse_create_handle (*C++ function*), 243
rocsparse_create_hyb_mat (*C++ function*), 249
rocsparse_create_identity_permutation (*C++ function*), 292
rocsparse_create_mat_descr (*C++ function*), 246
rocsparse_create_mat_info (*C++ function*), 250
rocsparse_csr2coo (*C++ function*), 281
rocsparse_csr2csc_buffer_size (*C++

function), 284, 290
rocsparse_csr2ell_width (*C++ function*), 287
rocsparse_csrilu0_clear (*C++ function*), 280
rocsparse_csrilu0_zero_pivot (*C++ function*), 273
rocsparse_csrmv_clear (*C++ function*), 261
rocsparse_csrsort (*C++ function*), 294
rocsparse_csrsort_buffer_size (*C++ function*), 293
rocsparse_csrsv_clear (*C++ function*), 269
rocsparse_csrsv_zero_pivot (*C++ function*), 264
rocsparse_daxpyi (*C++ function*), 251
rocsparse_dcoomv (*C++ function*), 256
rocsparse_dcsr2csc (*C++ function*), 284
rocsparse_dcsr2ell (*C++ function*), 287
rocsparse_dcsr2hyb (*C++ function*), 291
rocsparse_dcsrilu0 (*C++ function*), 276
rocsparse_dcsrilu0_analysis (*C++ function*), 275
rocsparse_dcsrilu0_buffer_size (*C++ function*), 274
rocsparse_dcsrmm (*C++ function*), 270
rocsparse_dcsrmv (*C++ function*), 258
rocsparse_dcsrmv_analysis (*C++ function*), 257
rocsparse_dcsrsv_analysis (*C++ function*), 265
rocsparse_dcsrsv_buffer_size (*C++ function*), 264
rocsparse_dcsrsv_solve (*C++ function*), 267
rocsparse_ddoti (*C++ function*), 252
rocsparse_dellmv (*C++ function*), 261
rocsparse_destroy_handle (*C++ function*), 243
rocsparse_destroy_hyb_mat (*C++ function*), 250
rocsparse_destroy_mat_descr (*C++ function*), 246
rocsparse_destroy_mat_info (*C++ function*), 250
rocsparse_dgthr (*C++ function*), 252
rocsparse_dgthrz (*C++ function*), 253
rocsparse_dhybmv (*C++ function*), 263
rocsparse_diag_type (*C++ enum*), 240
rocsparse_diag_type_non_unit (*C++ enumerator*), 240
rocsparse_diag_type_unit (*C++ enumerator*), 240
rocsparse_droti (*C++ function*), 254
rocsparse_dsctr (*C++ function*), 255
rocsparse_ell2csr_nnz (*C++ function*), 289
rocsparse_fill_mode (*C++ enum*), 240
rocsparse_fill_mode_lower (*C++ enumerator*), 240
rocsparse_fill_mode_upper (*C++ enumerator*), 240
rocsparse_get_git_rev (*C++ function*), 246
rocsparse_get_mat_diag_type (*C++ function*), 249
rocsparse_get_mat_fill_mode (*C++ function*), 249
rocsparse_get_mat_index_base (*C++ function*), 247
rocsparse_get_mat_type (*C++ function*), 248
rocsparse_get_pointer_mode (*C++ function*), 245
rocsparse_get_stream (*C++ function*), 244
rocsparse_get_version (*C++ function*), 245
rocsparse_handle (*C++ type*), 238
rocsparse_hyb_mat (*C++ type*), 238
rocsparse_hyb_partition (*C++ enum*), 239
rocsparse_hyb_partition_auto (*C++ enumerator*), 239
rocsparse_hyb_partition_max (*C++ enumerator*), 239
rocsparse_hyb_partition_user (*C++ enumerator*), 239
rocsparse_index_base (*C++ enum*), 239
rocsparse_index_base_one (*C++ enumerator*), 239
rocsparse_index_base_zero (*C++ enumerator*), 239
rocsparse_layer_mode (*C++ enum*), 241
rocsparse_layer_mode_log_bench (*C++ enumerator*), 242
rocsparse_layer_mode_log_trace (*C++ enumerator*), 241
rocsparse_layer_mode_none (*C++ enumerator*), 241
rocsparse_mat_descr (*C++ type*), 238
rocsparse_mat_info (*C++ type*), 238
rocsparse_matrix_type (*C++ enum*), 239
rocsparse_matrix_type_general (*C++ enumerator*), 239
rocsparse_matrix_type_hermitian (*C++ enumerator*), 239
rocsparse_matrix_type_symmetric (*C++ enumerator*), 239
rocsparse_matrix_type_triangular (*C++ enumerator*), 239
rocsparse_operation (*C++ enum*), 240
rocsparse_operation_conjugate_transpose (*C++ enumerator*), 240
rocsparse_operation_none (*C++ enumerator*), 240
rocsparse_operation_transpose (*C++ enumerator*), 240
rocsparse_pointer_mode (*C++ enum*), 241

rocsparse_pointer_mode_device (*C++ enumerator*), 241
rocsparse_pointer_mode_host (*C++ enumerator*), 241
rocsparse_saxpyi (*C++ function*), 251
rocsparse_scoomv (*C++ function*), 256
rocsparse_scsr2csc (*C++ function*), 284
rocsparse_scsr2ell (*C++ function*), 287
rocsparse_scsr2hyb (*C++ function*), 291
rocsparse_scsrilu0 (*C++ function*), 276
rocsparse_scsrilu0_analysis (*C++ function*), 275
rocsparse_scsrilu0_buffer_size (*C++ function*), 274
rocsparse_scsrmm (*C++ function*), 270
rocsparse_scsrmv (*C++ function*), 258
rocsparse_scsrmv_analysis (*C++ function*), 257
rocsparse_scsrsv_analysis (*C++ function*), 265
rocsparse_scsrsv_buffer_size (*C++ function*), 264
rocsparse_scsrsv_solve (*C++ function*), 267
rocsparse_sdoti (*C++ function*), 252
rocsparse_sellmv (*C++ function*), 261
rocsparse_set_mat_diag_type (*C++ function*), 249
rocsparse_set_mat_fill_mode (*C++ function*), 248
rocsparse_set_mat_index_base (*C++ function*), 247
rocsparse_set_mat_type (*C++ function*), 248
rocsparse_set_pointer_mode (*C++ function*), 245
rocsparse_set_stream (*C++ function*), 244
rocsparse_sgthr (*C++ function*), 252
rocsparse_sgthrz (*C++ function*), 253
rocsparse_shybmv (*C++ function*), 263
rocsparse_solve_policy (*C++ enum*), 241
rocsparse_solve_policy_auto (*C++ enumerator*), 241
rocsparse_sroti (*C++ function*), 254
rocsparse_ssctr (*C++ function*), 255
rocsparse_status (*C++ enum*), 242
rocsparse_status_arch_mismatch (*C++ enumerator*), 242
rocsparse_status_internal_error (*C++ enumerator*), 242
rocsparse_status_invalid_handle (*C++ enumerator*), 242
rocsparse_status_invalid_pointer (*C++ enumerator*), 242
rocsparse_status_invalid_size (*C++ enumerator*), 242
rocsparse_status_invalid_value (*C++ enumerator*), 242
rocsparse_status_memory_error (*C++ enumerator*), 242
rocsparse_status_not_implemented (*C++ enumerator*), 242
rocsparse_status_success (*C++ enumerator*), 242
rocsparse_status_zero_pivot (*C++ enumerator*), 242

T

TO_STR (*C macro*), 141
TO_STR2 (*C macro*), 141

V

VEC_BUFF_MAX_BYTES (*C++ member*), 143
VERSION_STRING (*C macro*), 141