ReadTheDocs-Breathe Documentation

Release 1.0.0

Thomas Edvalson

Feb 06, 2019

Contents

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.

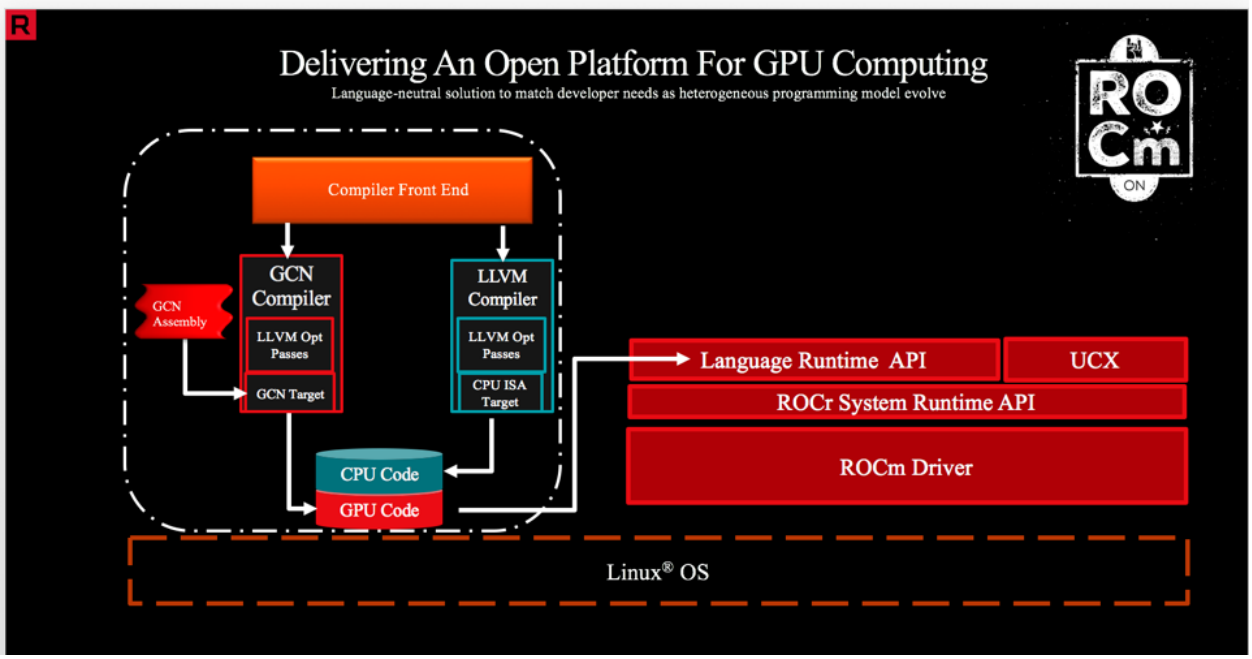
ROCm is built for scale; it supports multi-GPU computing in and out of server-node communication through RDMA. It also simplifies the stack when the driver directly incorporates RDMA peer-sync support.

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

CHAPTER 1

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

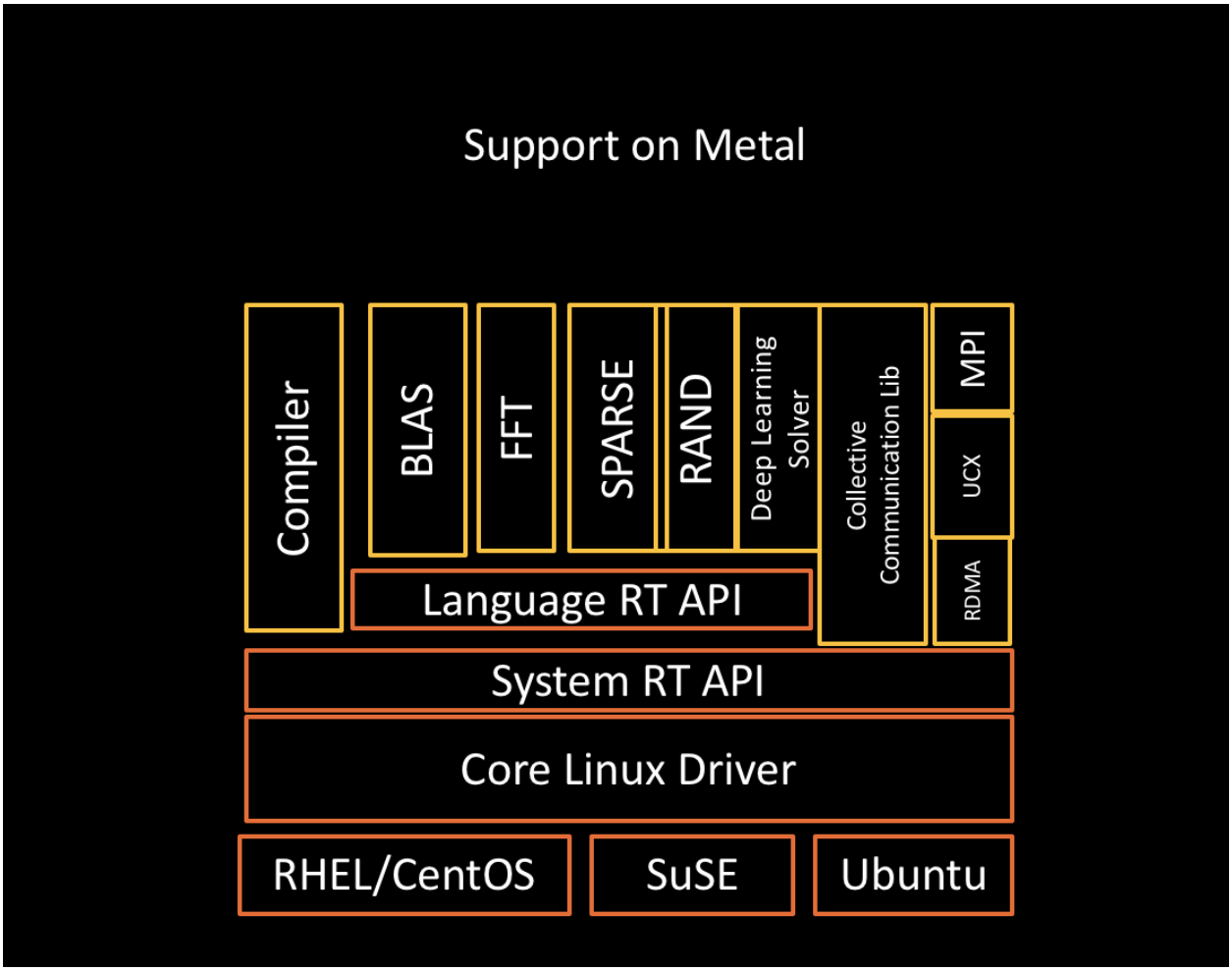
The ROCr System Runtime is language independent and makes heavy use of the Heterogeneous System Architecture (HSA) Runtime API. This approach provides a rich foundation to exectute programming languages such as HCC C++ and HIP, the Khronos Group's OpenCL, and Continuum's Anaconda Python.



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 2

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 modifying the open source code base and rebuilding 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 regarding the installation steps and issues that can be faced during installation.

2.1.3 Programming Guide

This guide provides a detailed discussion of the ROCm programming model and programming interface. It then 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 which can be found in this section.

HC Programing Guide

This guide provides a detailed discussion of The Heterogeneous Compute programming installation requirements methods to install on various platfroms 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 variou concepts of HIP Poring, 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 Perfromance 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 and 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 we have MIOpen API and MIOpenGEMM API listed.

2.1.7 ROCm Tools

HCC

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

GCN Assembler and Disassembler

This Section provieds details regarding GCN in-detail.

GCN Assembler Tools

In this Section there are useful items related to AMDGPU ISA assembler has been documented.

ROCm-GDB

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

ROCm-Profiler

This section gives Details on Radeon Compute Profiler is a performance analysis tool, here we have details on how to clone and use it.

CodeXL

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

GPUperfAPI

This section provieds details related to GPU Performance API. 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 and also be compiled with the CUDA compiler using HIP tools for running on Nvidia GPU devices.

|rocBLAS | This section provieds 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 :ref:'rocblas'and cuBLAS as backends.

hcRNG

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

clFFT

This section provieds 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 provieds 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 provieds 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..

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

Here we have instruction related to ROCm Device Library overview,Building and Testing related information with respect to Device Library.

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 where all detailes related to file structure related to system are captured in sysfs.

KFD Topology

KFD Kernel Topology is the system file structure wich 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 New features and enhancements in ROCm 2.1

2.2.1.1 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

2.2.1.2 Improvements to ROCM-SMI tool -

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

2.2.1.3 DGEMM Optimizations -

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

2.2.2 New features and enhancements in ROCm 2.0

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

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

2.2.2.2 Adds support for Vega 7nm, Polaris 12 GPUs

2.2.2.3 Introduces MIVisionX

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

2.2.2.4 Improvements to ROCm Libraries

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

2.2.2.5 MIOpen

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

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

- TensorFlow v1.12 is enabled with fp16 support

2.2.2.7 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

2.2.2.8 Improvements to ROCProfiler tool

- Support for Vega 7nm

2.2.2.9 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.

2.2.2.10 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

2.2.2.11 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

2.2.2.12 Kubernetes support

2.2.2.13 Removed features

- HCC: removed support for C++AMP

2.2.3 New features and enhancements in ROCm 1.9.2

2.2.3.1 RDMA(MPI) support on Vega 7nm

- Support ROCnRDMA based on Mellanox InfiniBand.

2.2.3.2 Improvements to HCC

- Improved link time optimization.

2.2.3.3 Improvements to ROCProfiler tool

- General bug fixes and implemented versioning APIs.

2.2.3.4 Critical bug fixes

2.2.4 New features and enhancements in ROCm 1.9.1

2.2.4.1 Added DPM support to Vega 7nm

Dynamic Power Management feature is enabled on Vega 7nm.

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

2.2.5 New features and enhancements in ROCm 1.9.0

2.2.5.1 Preview for Vega 7nm

- Enables developer preview support for Vega 7nm

2.2.5.2 System Management Interface

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

2.2.5.3 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

2.2.5.4 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

2.2.5.5 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

2.2.5.6 New distribution support

- Binary package support for Ubuntu 18.04

2.2.5.7 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 Current ROCm Version: 2.1

2.3.2 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.

2.3.2.1 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

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:

- 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.

At this time, 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.

2.3.2.2 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.

2.3.2.3 Not supported or limited support under ROCm

2.3.2.3.1 Limited support

- ROCm 2.1.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.

2.3.2.3.2 Not supported

- "Tonga", "Iceland", "Vega M", and "Vega 12" GPUs are not supported in ROCm 2.1.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.

2.3.3 The latest ROCm platform - ROCm 2.1

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
      - ROCm Clang
      - ROCm lld
      - ROCm Device Libraries
  - ROCM Clang-OCL Kernel Compiler
  - Asynchronous Task and Memory Interface (ATMI)
  - 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
  - HIP Thrust
  - ROCm SMI Lib
  - RCCL
  - MIVisionX
  - CUB HIP

2.3.3.1 Supported Operating Systems - New operating systems available

The ROCm 2.1.x platform supports the following operating systems:

- Ubuntu 16.04.x and 18.04.x (Version 16.04.3 and newer or kernels 4.13 and newer)
- CentOS 7.4, 7.5, and 7.6 (Using devtoolset-7 runtime support)
- RHEL 7.4, 7.5, and 7.6 (Using devtoolset-7 runtime support)

2.3.3.2 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

2.3.4 Installing from AMD ROCm repositories

AMD hosts both Debian and RPM repositories for the ROCm 2.1.x packages at this time.

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

2.3.4.1 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)
  - HIP Thrust: `thrust` (on RHEL/CentOS), `hip-thrust` (on Ubuntu)
  - ROCm SMI Lib: `rocm_smi_lib64`
  - RCCL: `rccl`
  - MIVisionX: `mivisionx`
  - CUB HIP: `cub-hip`

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
      |--rocm-device-libs
      |--rocm-utils
          |-- rocminfo
          |-- rocm-cmake
          \-- rocm-clang-ocl # This will cause OpenCL to be installed
```

(continues on next page)

(continued from previous page)

```
      |--hcc
      |--hip_base
      |--hip_doc
      |--hip_hcc
      |--hip_samples
      |--rocm-smi
      |--hsakmt-roct
      |--hsakmt-roct-dev
      |--hsa-amd-aqlprofile
      |--comgr
      \--rocr_debug_agent

rocm-libs
 |-- rocblas
 |-- rocfft
 |-- rocrand
 \-- hipblas
```

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.

2.3.4.2 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.

2.3.4.2.1 First make sure your system is up to date

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

2.3.4.2.2 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
```

2.3.4.2.3 Install

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

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

2.3.4.2.4 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.

2.3.4.2.5 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.

2.3.4.2.6 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
```

2.3.4.2.7 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
```

2.3.4.2.8 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

2.3.4.2.9 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
```

2.3.4.3 CentOS/RHEL 7 (7.4, 7.5, 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.

2.3.4.3.1 Preparing RHEL 7 (7.4, 7.5, 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
```

2.3.4.3.2 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.

2.3.4.3.3 Prepare CentOS/RHEL (7.4, 7.5, 7.6) for DKMS Install

Installing kernel drivers on CentOS/RHEL 7.4/7.5/7.6 requires dkms tool being installed:

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

2.3.4.3.4 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.

2.3.4.3.5 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 7.4, 7.5, 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.

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.

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
```

2.3.4.3.6 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
```

2.3.4.3.7 How to uninstall ROCm from CentOS/RHEL 7.4, 7.5 and 7.6

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

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

2.3.4.3.8 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 yum install rocm-dev
```

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

2.3.4.3.9 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
```

2.3.5 Known issues / workarounds

PyTorch : observing "test_gamma_gpu_sample" subtest failure and test_cuda : test_gather_dim test failure in few configs Tensor flow : observed memory access fault while running SAGAN tensor flow model in Polaris based ASIC Caffe2 : Observed segmentation fault (core dumped) while running Caffe2 mGPU Resnet/ResNext Training using 4 GPU's

2.3.5.1 HipCaffe is supported on single GPU configurations

2.3.5.2 The ROCm SMI library calls to rsmi_dev_power_cap_set() and rsmi_dev_power_profile_set() will not work for all but the first gpu in multi-gpu set ups.

2.3.6 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

2.3.7 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.

2.3.7.1 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.

2.3.7.2 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-2.1.0
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.

2.3.7.3 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.

2.3.8 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 HCC: Heterogeneous Compute Compiler

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 source code can be in the same source file and use the same C++ language;templates and classes work naturally across the host/device boundary
- HCC generates both host and device code from the same compiler, so it benefits from a consistent view of the source code using the same Clang-based language parser

Full control over the machine

- Access AMD scratchpad memories ("LDS")
- Fully control data movement, prefetch and discard
- Fully control asynchronous kernel launch and completion
- Get device-side dependency resolution for kernel and data commands (without host involvement)
- Obtain HSA agents, queues and signals for low-level control of the architecture using the HSA Runtime API
- Use [direct-to-ISA](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 GPGPU languages will find HIP very easy to learn and use. AMD platforms implement this language using the HC dialect described above, providing similar low-level control over the machine.

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_MEMFENCE
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 HC Programming Guide

What is the Heterogeneous Compute (HC) API ?

It's a C++ dialect with extensions to launch kernels and manage accelerator memory. It closely tracks the evolution of C++ and will incorporate parallelism and concurrency features as the C++ standard does. For example, HC includes early support for the C++17 Parallel STL. At the recent ISO C++ meetings in Kona and Jacksonville, the committee was excited about enabling the language to express all forms of parallelism, including 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 source code can be in the same source file and use the same C++ language; templates and classes work naturally across the host/device boundary
- HCC generates both host and device code from the same compiler, so it benefits from a consistent view of the source code using the same Clang-based language parser

Full control over the machine

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

2.4.2.1 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.

2.4.2.2 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.

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.

2.4.2.3 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.

2.4.2.4 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.

2.4.2.5 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
```

2.4.2.6 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 \
..
make
```

To install it, use the following steps

```
sudo make install
```

2.4.2.7 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

2.4.2.8 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
```

2.4.2.9 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 `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: `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"
```

(continues on next page)

(continued from previous page)

```
.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
// 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
```

(continues on next page)

(continued from previous page)

```
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 Thrust API Documentation

- HIP-thrust

2.7.6 Math Library API's

- hcRNG
- clBLAS
- clSPARSE_API

2.7.7 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

HW specific low-level performance analysis API, 'rocprofiler' library and 'rocprof' tool for profiling of GPU compute applications. The profiling includes HW performance counters with complex performance metrics and HW traces. Supports GFX8/GFX9.

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

- Output results in CSV format

2.8.4.1 Download

To clone ROC Profiler from GitHub use the folowing command:

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

The library source tree:

- **bin**
  - rpl_run.sh - 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/lib:/opt/rocm/include/hsa
cmake -DCMAKE_INSTALL_PREFIX=/opt/rocm ..
make
sudo make install
```

To test the built library:

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

2.8.4.3 Profiling Tool 'rocprof' Usage

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

```
rpl_run.sh [-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/sqtt 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>

-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]
```

(continues on next page)

(continued from previous page)

```
--ctx-limit <max number> - maximum number of outstanding contexts [0 - unlimited]
--heartbeat <rate sec> - to print progress heartbeats [0 - disabled]
--sqtt-size <byte size> - to set SQTT buffer size, aggregate for all SE [0x2000000]
    Can be set in KB (1024B) or MB (1048576) units, examples 20K or 20M respectively.
--sqtt-local <on|off> - to allocate SQTT buffer in local GPU memory [on]

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', 'sqtt-size',
↪ 'sqtt-local'.
An example of 'rpl_rc.xml':
  <defaults
    basenames=off
    timestamp=off
    ctx-limit=0
    heartbeat=0
    sqtt-size=0x20M
    sqtt-local=on
  ></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 ROCm2.0.

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 2.0:

```
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

    .
    .
    .

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
```

(continues on next page)

(continued from previous page)

```
     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 ROCm-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.

RCP was formerly delivered as part of CodeXL with the executable name "CodeXLGpuProfiler". Prior to its inclusion in CodeXL, it was known as "sprofile" and was part of the AMD APP Profiler product.

A subset of RCP is (the portion that supports ROCm) is automatically installed with ROCm. Once ROCm is installed, the profiler will appear in the /opt/rocm/profiler directory.

- Measure the execution time of an OpenCL™ or ROCm/HSA kernel.
- Query the hardware performance counters on an AMD Radeon graphics card.

- Use the CXLActivityLogger API to trace and measure the execution of segments in the program.
- Display the IL/HSAIL and ISA (hardware disassembly) code of OpenCL™ kernels.
- Calculate kernel occupancy information, which estimates the number of in-flight wavefronts on a compute unit as a percentage of the theoretical maximum number of wavefronts that the compute unit can support.
- When used with CodeXL, all profiler data can be visualized in a user-friendly graphical user interface.
- **Version 5.5 (8/22/18)**
  - Adds support for additional GPUs, including Vega series GPUs
  - ROCm/HSA: Support for ROCm 2.0
  - Improves display of pointer parameters for some HSA APIs in the ATP file
  - Fixes an issue with parsing an ATP file which has non-ascii characters (affected Summary page generation and display within CodeXL)
  - ROCm/HSA: Fixes several issues with incorrect or missing data transfer timestamps.
- An AMD Radeon GCN-based GPU or APU
- **Radeon Software Adrenaline Edition 18.8.1 or later (Driver Packaging Version 18.30 or later).**
  - For Vega support, a driver with Driver Packaging Version 17.20 or later is required
- ROCm 2.0 See system requirements for ROCm: https://rocm-documentation.readthedocs.io/en/latest/Installation_Guide/Installation-Guide.html and https://rocm.github.io/hardware.html.
- **Windows 7, 8.1, and 10**
  - For Windows, the Visual C++ Redistributable for Visual Studio 2015 is required. It can be downloaded from https://www.microsoft.com/en-us/download/details.aspx?id=48145
- Ubuntu (16.04 and later) and RHEL (7 and later) distributions

To clone the RCP repository, execute the following git commands

- git clone https://github.com/GPUOpen-Tools/RCP.git

After cloning the repository, please run the following python script to retrieve the required dependencies (see BUILD.md for more information):

- python Scripts/UpdateCommon.py

UpdateCommon.py has replaced the use of git submodules in the CodeXL repository Source Code Directory Layout

- Build – contains both Linux and Windows build-related files
- Scripts– scripts to use to clone/update dependent repositories
- Src/CLCommon – contains source code shared by the various OpenCL™ agents
- Src/CLOccupancyAgent – contains source code for the OpenCL™ agent which collects kernel occupancy information
- Src/CLProfileAgent – contains source code for the OpenCL™ agent which collects hardware performance counters
- Src/CLTraceAgent – contains source code for the OpenCL™ agent which collects application trace information
- Src/Common – contains source code shared by all of RCP
- Src/DeviceInfo – builds a lib containing the Common/Src/DeviceInfo code (Linux only)
- Src/HSAFdnCommon – contains source code shared by the various ROCm agents

- Src/HSAFdnPMC – contains source code for the ROCm agent which collects hardware performance counters
- Src/HSAFdnTrace – contains source code for the ROCm agent which collects application trace information
- Src/HSAUtils – builds a lib containing the Common ROCm code (Linux only)
- Src/MicroDLL – contains source code for API interception (Windows only)
- Src/PreloadXInitThreads – contains source code for a library that call XInitThreads (Linux only)
- Src/ProfileDataParser – contains source code for a library can be used to parse profiler output data files
- Src/VersionInfo– contains version info resource files
- Src/sanalyze – contains source code used to analyze and summarize profiler data
- Src/sprofile – contains source code for the main profiler executable

Although the Radeon Compute Profiler is a newly-branded tool, the technology contained in it has been around for several years. RCP has its roots in the AMD APP Profiler product, which progressed from version 1.x to 3.x. Then the profiler was included in CodeXL, and the codebase was labelled as version 4.x. Now that RCP is being pulled out of CodeXL and into its own codebase again, we've bumped the version number up to 5.x.

ROCm Profiler blog post

- **For the OpenCL™ Profiler**
  - Collecting Performance Counters for an OpenCL™ application is not currently working for Vega GPUs on Windows when using a 17.20-based driver. This is due to missing driver support in the 17.20 driver. Future driver versions should provide the support needed.
  - Collecting Performance Counters using –perfcounter for an OpenCL™ application when running OpenCL-on-ROCm is not suported currently. The workaround is to profile using the ROCm profiler (using the –hsapmc command-line switch).
- **For the ROCm Profiler**
  - API Trace and Perf Counter data may be truncated or missing if the application being profiled does not call hsa_shut_down
  - Kernel occupancy information will only be written to disk if the application being profiled calls hsa_shut_down
  - When collecting a trace for an application that performs memory transfers using hsa_amd_memory_async_copy, if the application asks for the data transfer timestamps directly, it will not get correct timestamps. The profiler will show the correct timestamps, however.
  - When collecting an aql packet trace, if the application asks for the kernel dispatch timestamps directly, it will not get correct timestamps. The profiler will show the correct timestamps, however.
  - When the rocm-profiler package (.deb or .rpm) is installed along with rocm, it may not be able to generate the default single-pass counter files. If you do not see counter files in /opt/rocm/profiler/counterfiles, you can generate them manually with this command: "sudo /opt/rocm/profiler/bin/CodeXLGpuProfiler –list –outputfile /opt/rocm/profiler/ counterfiles/counters –maxpassperfile 1"

2.8.8 CodeXL

CodeXL is a comprehensive tool suite that enables developers to harness the benefits of CPUs, GPUs and APUs. It includes powerful GPU debugging, comprehensive GPU and CPU profiling, DirectX12® Frame Analysis, static OpenCL™, OpenGL®, Vulkan® and DirectX® kernel/shader analysis capabilities, and APU/CPU/GPU power profiling, enhancing accessibility for software developers to enter the era of heterogeneous computing. CodeXL is available both as a Visual Studio® extension and a standalone user interface application for Windows® and Linux®.

2.8.8.1 Motivation

CodeXL, previously a tool developed as closed-source by Advanced Micro Devices, Inc., is now released as Open Source. AMD believes that adopting the open-source model and sharing the CodeXL source base with the world can help developers make better use of CodeXL and make CodeXL a better tool.

To encourage 3rd party contribution and adoption, CodeXL is no longer branded as an AMD product. AMD will still continue development of this tool and upload new versions and features to GPUOpen.

2.8.8.2 Installation and Build

Windows: To install CodeXL, use the provided executable file CodeXL_*.exe Linux: To install CodeXL, use the provided RPM file, Debian file, or simply extract the compressed archive onto your hard drive. Refer to BUILD.md for information on building CodeXL from source.

2.8.8.3 Contributors

CodeXL's GitHub repository (http://github.com/GPUOpen-Tools/CodeXL) is moderated by Advanced Micro Devices, Inc. as part of the GPUOpen initiative.

AMD encourages any and all contributors to submit changes, features, and bug fixes via Git pull requests to this repository.

Users are also encouraged to submit issues and feature requests via the repository's issue tracker.

2.8.8.4 License

CodeXL is part of the GPUOpen.com initiative. CodeXL source code and binaries are released under the following MIT license:

MIT LICENSE: Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.8.8.5 Attribution and Copyrights

Component licenses can be found under the CodeXL GitHub repository source root, in the /Setup/Legal/ folder.

OpenCL is a trademark of Apple Inc. used by permission by Khronos. OpenGL is a registered trademark of Silicon Graphics, Inc. in the United States and/or other countries worldwide. Microsoft, Windows, DirectX and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other jurisdictions. Vulkan is

a registered trademark of Khronos Group Inc. in the United States and/or other jurisdictions. Linux is the registered trademark of Linus Torvalds in the United States and/or other jurisdictions.

LGPL (Copyright ©1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA). Use of the Qt library is governed by the GNU Lesser General Public License version 2.1 (LGPL v 2.1). CodeXL uses QT 5.5.1. Source code for QT is available here: http://qt-project.org/downloads. The QT source code has not been tempered with and the built binaries are identical to what any user that downloads the source code from the web and builds them will produce.

Boost is Copyright © Beman Dawes, 2003. [CR]LunarG, Inc. is Copyright © 2015 LunarG, Inc. jqPlot is copyright © 2009-2011 Chris Leonello. glew - The OpenGL Extension Wrangler Library is Copyright © 2002-2007, Milan Ikits <milan ikits[]ieee org>, Copyright © 2002-2007, Marcelo E. Magallon <mmagallo[]debian org>, Copyright © 2002, Lev Povalahev, All rights reserved. lgplib is Copyright © 1994-1998, Thomas G. Lane., Copyright © 1991-2013, Thomas G. Lane, Guido Vollbeding. LibDwarf (BSD) is Copyright © 2007 John Birrell (jb@freebsd.org), Copyright © 2010 Kai Wang, All rights reserved. libpng is Copyright © 1998-2014 Glenn Randers-Pehrson, (Version 0.96 Copyright © 1996, 1997 Andreas Dilger) (Version 0.88 Copyright © 1995, 1996 Guy Eric Schalnat, Group 42, Inc.). QScintilla is Copyright © 2005 by Riverbank Computing Limited info@riverbankcomputing.co.uk. TinyXML is released under the zlib license © 2000-2007, Lee Thomason, © 2002-2004, Yves Berquin © 2005, Tyge Lovset. UTF8cpp is Copyright © 2006 Nemanja Trifunovic. zlib is Copyright © 1995-2010 Jean-loup Gailly and Mark Adler, Copyright © 2003 Chris Anderson christop@charm.net, Copyright © 1998-2010 Gilles Vollant (minizip) (http://www.winimage.com/zLibDll/minizip.html), Copyright © 2009-2010 Mathias Svensson (http://result42.com), Copyright © 2007-2008 Even Rouault. QCustomPlot, an easy to use, modern plotting widget for Qt, Copyright (C) 2011-2015 Emanuel Eichhammer

2.8.9 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 both CodeXL and GPU PerfStudio.

2.8.9.1 Major Features

- Provides a standard API for accessing GPU Performance counters for both graphics and compute workloads across multiple GPU APIs.
- Supports DirectX11, OpenGL, OpenGLES, OpenCL™, and ROCm/HSA
- Developer Preview for DirectX12 (no hardware-based performance counter support yet)
- Supports all current GCN-based Radeon graphics cards and APUs.
- Supports both Windows and Linux
- Provides derived "public" counters based on raw HW counters
- "Internal" version provides access to some raw hardware counters. See "Public" vs "Internal" Versions for more information.

2.8.9.2 What's New

Version 2.23 (6/27/17)

- Add support for additional GPUs, including Vega series GPUs
- Allow unit tests to be built and run on Linux

2.8.9.3 System Requirements

- An AMD Radeon GCN-based GPU or APU
- **Radeon Software Crimson ReLive Edition 17.4.3 or later (Driver Packaging Version 17.10 or later).**
  - For Vega support, a driver with Driver Packaging Version 17.20 or later is required
- Pre-GCN-based GPUs or APUs are no longer supported by GPUPerfAPI. Please use an older version (2.17) with older hardware.
- Windows 7, 8.1, and 10
- Ubuntu (16.04 and later) and RHEL (7 and later) distributions

2.8.9.4 Cloning the Repository

To clone the GPA repository, execute the following git commands

- git clone https://github.com/GPUOpen-Tools/GPA.git After cloning the repository, please run the following python script to retrieve the required dependencies (see BUILD.md for more information):
- python Scripts/UpdateCommon.py UpdateCommon has replaced the use of git submodules in the GPA repository

2.8.9.5 Source Code Directory Layout

- Build – contains both Linux and Windows build-related files
- Common – Common libs, header and source code not found in other repositories
- Doc – contains User Guide and Doxygen configuration files
- Src/DeviceInfo – builds a lib containing the Common/Src/DeviceInfo code (Linux only)
- Src/GPUPerfAPI-Common– contains source code for a Common library shared by all versions of GPUPerfAPI
- Src/GPUPerfAPICL - contains the source for the OpenCL™ version of GPUPerfAPI
- Src/GPUPerfAPICounterGenerator - contains the source code for a Common library providing all counter data
- Src/GPUPerfAPICounters - contains the source code for a library that can be used to query counters without an active GPUPerfAPI context
- Src/GPUPerfAPIDX - contains source code shared by the DirectX versions of GPUPerfAPI
- Src/GPUPerfAPIDX11 - contains the source for the DirectX11 version of GPUPerfAPI
- Src/GPUPerfAPIDX12 - contains the source for the DirectX12 version of GPUPerfAPI (Developer Preview)
- Src/GPUPerfAPIGL - contains the source for the OpenGL version of GPUPerfAPI
- Src/GPUPerfAPIGLES - contains the source for the OpenGLES version of GPUPerfAPI
- Src/GPUPerfAPIHSA - contains the source for the ROCm/HSA version of GPUPerfAPI
- Src/GPUPerfAPIUnitTests- contains a small set of unit tests for GPUPerfAPI
- Src/PublicCounterCompiler - source code for a tool to generate C++ code for public counters from text files defining the counters.
- Src/PublicCounterCompilerInputFiles - input files that can be fed as input to the PublicCounterCompiler tool
- Scripts – scripts to use to clone/update dependent repositories

2.8.9.6 Public" vs "Internal" Versions

This open source release supports building both the "Public" and "Internal" versions of GPUPerfAPI. By default the Visual Studio solution and the Linux build scripts will produce what is referred to as the "Public" version of GPUPerfAPI. This version exposes "Public", or "Derived", counters. These are counters that are computed using a set of hardware counters. Until now, only the Public the version of GPUPerfAPI was available on the AMD Developer website. As part of the open-source effort, we are also providing the ability to build the "Internal" versions of GPUPerfAPI. In addition to exposing the same counters as the Public version, the Internal version also exposes some of the hardware Counters available in the GPU/APU. It's important to note that not all hardware counters receive the same validation as other parts of the hardware on all GPUs, so in some cases accuracy of counter data cannot be guaranteed. The usage of the Internal version is identical to the Public version. The only difference will be in the name of the library an application loads at runtime and the list of counters exposed by the library. See the Build Instructions for more information on how to build and use the Internal version. In the future, we see there being only a single version of GPUPerfAPI, with perhaps a change in the API to allow users of GPA to indicate whether the library exposes just the Derived counters or both the Derived and the Hardware counters. We realize using the term "Internal" for something which is no longer actually Internal-to-AMD can be a bit confusing, and we will aim to change this in the future.

2.8.9.7 Known Issues

- The OpenCL™ version of GPUPerfAPI requires at least Driver Version 17.30.1071 for Vega GPUs on Windows. Earlier driver versions have either missing or incomplete support for collecting OpenCL performance counters

2.8.10 ROCm Binary Utilities

Documentation need to be updated.

2.8.11 MIVisionX

MIVisionX toolkit is a comprehensive computer vision and machine intelligence libraries, utilities and applications bundled into a single toolkit

2.8.11.1 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.

The amd_openvx project consists of the following components:

- **OpenVX**: AMD OpenVX library

The OpenVX framework provides a mechanism to add new vision functions to OpenVX by 3rd party vendors. Look into github

- **vx_nn**: OpenVX neural network module that was built on top of MIOpen
- **vx_loomsl**: Radeon LOOM stitching library for live 360 degree video applications
- **vx_opencv**: OpenVX module that implemented a mechanism to access OpenCV functionality as OpenVX kernels

2.8.11.1.1 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.

2.8.11.1.2 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 libraries**
  - 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.

2.8.11.1.3 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.

2.8.11.1.4 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.

2.8.11.1.5 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.

2.8.11.2 AMD OpenVX Extensions (amd_openvx_extensions)

The OpenVX framework provides a mechanism to add new vision functions to OpenVX by 3rd party vendors. This project has below OpenVX modules and utilities to extend AMD OpenVX (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

2.8.11.2.1 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} };
```

(continues on next page)

(continued from previous page)

```
    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");
```

(continues on next page)

(continued from previous page)

```
    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;
}
```

2.8.11.2.2 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 | | Con-
cat|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 | | Lo-
cal Response Normalization|vxNormalizationLayer|org.khronos.nn_extension.normalization_layer
| | Pooling|vxPoolingLayer|org.khronos.nn_extension.pooling_layer | | ROI Pool-
ing|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 | | Ten-
sor Convert to Image|vxConvertTensorToImageNode|com.amd.nn_extension.convert_tensor_to_image
| | Tensor Multiply|vxTensorMultiplyNode|org.khronos.openvx.tensor_multiply | | Tensor Sub-
tract|vxTensorSubtractNode|org.khronos.openvx.tensor_subtract | | Upsample Nearest Neighbor-
hood|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
```

(continues on next page)

(continued from previous page)

```
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
```

2.8.11.2.3 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
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
```

(continues on next page)

(continued from previous page)

```
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
```

2.8.11.2.3.1 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

2.8.11.3 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.

2.8.11.3.1 Cloud Inference Application (cloud_inference)

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

2.8.11.3.2 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 . . .

2.8.11.3.3 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

2.8.11.4 Samples

MIVisionX samples using OpenVX and OpenVX extension libraries

2.8.11.4.1 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
```

2.8.11.5 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.

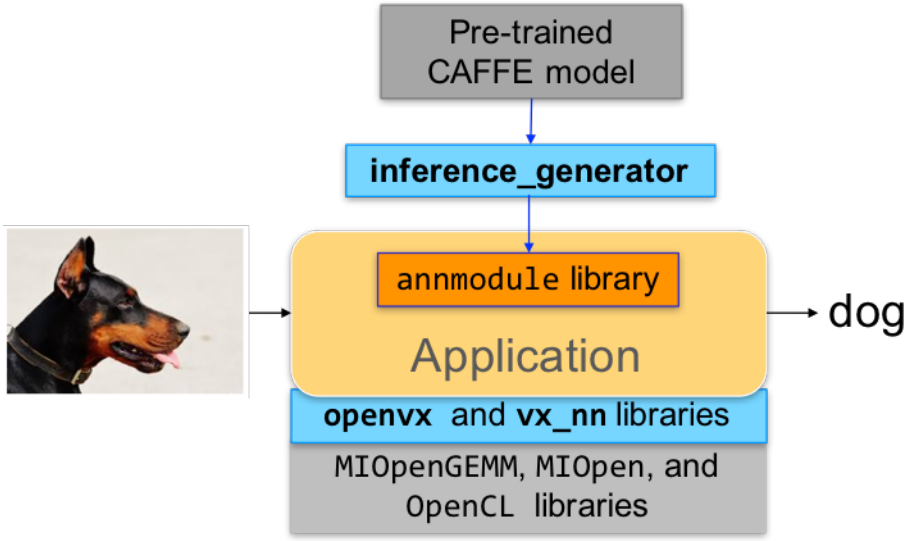
MIVisionX provides you with tools for accomplishing your tasks throughout the whole neural net life-cycle, from creating a model to deploying them for your target platforms.

2.8.11.6 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.



2.8.11.7 Pre-requisites

- CPU: SSE4.1 or above CPU, 64-bit
- **GPU: Radeon Instinct or Vega Family of Products (16GB recommended)**
  - Linux: install ROCm with OpenCL development kit
  - Windows: install the latest drivers and OpenCL SDK download
- CMake 2.8 or newer download
- Qt Creator for annInferenceApp
- **protobuf for inference_generator**

  - install libprotobuf-dev and protobuf-compiler needed for vx_nn

- **OpenCV 3 (optional) download for vx_opencv**
  - Set OpenCV_DIR environment variable to OpenCV/build folder

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

2.8.11.7.2 Prerequisites for running the scripts

- ubuntu 16.04/18.04
- ROCm supported hardware
- ROCm

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.

usage:

```
python MIVisionX-setup.py -s [sudo password - required] -d [setup directory -␣
↪optional (default:~/)] -m [MIOpen Version - optional (default:1.6.0)]
```

2.8.11.8 Build MIVisionX

2.8.11.8.1 Build using CMake on Linux (Ubuntu 16.04 64-bit) 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
- git clone this project using –recursive option so that correct branch of the deps project is cloned automatically.
- **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)

2.8.11.8.2 Build annInferenceApp using Qt Creator

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

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

- Use loom.sln to build x64 platform

2.8.11.9 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.

2.8.11.9.1 MIVisionX Docker

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

2.8.11.9.2 Docker Workflow Sample on Ubuntu 16.04

Prerequisites

- Ubuntu 16.04
- rocm supported hardware

2.8.11.9.3 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
apt-cache policy docker-ce
sudo apt-get install -y docker-ce
sudo systemctl status docker
```

Step 3 - Get Docker Image

```
sudo docker pull kiritigowda/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 kiritigowda/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␣
↪kiritigowda/mivisionx-ubuntu-16.04
```

2.8.11.10 Release Notes

2.8.11.10.1 Supported Neural Net Layers

```
Layer name
Activation
Argmax
Batch Normalization
Concat
Convolution
Deconvolution
Fully Connected
Local Response Normalization (LRN)
Pooling
Scale
Slice
Softmax
Tensor Add
Tensor Convert Depth
Tensor Convert from Image
Tensor Convert to Image
Tensor Multiply
Tensor Subtract
Upsample Nearest Neighborhood
```

2.8.11.10.2 Known issues

- ROCm - 1.8.151 performance degradation

2.8.11.10.3 Tested configurations

- Linux: Ubuntu - 16.04/18.04 & CentOS - 7.5/7.6
- ROCm: rocm-dkms - 1.9.307
- rocm-cmake - github master:ac45c6e
- MIOpenGEMM - 1.1.5
- MIOpen - 1.6.0
- Protobuf - V3.5.2
- OpenCV - 3.3.0
- 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.

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.

Library and API Documentation Please refer to the Library documentation for current documentation.

2.9.1.4 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;
        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
```

(continues on next page)

(continued from previous page)

```
        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;
}
```

2.9.2 rocBLAS

- 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 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
```

2.9.2.2 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)
```

2.9.2.3 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

2.9.2.4 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

2.9.2.5 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);
```

2.9.2.6 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.

2.9.2.7 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.

2.9.2.8 API

This section provides details of the library API

2.9.2.8.1 Types

2.9.2.8.1.1 Definitions

2.9.2.8.1.2 rocblas_int

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

2.9.2.8.1.3 rocblas_long

typedef int64_t **rocblas_long**

2.9.2.8.1.4 rocblas_float_complex

typedef float2 **rocblas_float_complex**

2.9.2.8.1.5 rocblas_double_complex

typedef double2 **rocblas_double_complex**

2.9.2.8.1.6 rocblas_half

typedef uint16_t **rocblas_half**

2.9.2.8.1.7 rocblas_half_complex

typedef float2 **rocblas_half_complex**

2.9.2.8.1.8 rocblas_handle

Warning: doxygentypedef: Cannot find typedef "rocblas_handle" in doxygen xml output for project "ReadTheDocs-Breathe" from directory: xml/

2.9.2.8.1.9 Enums

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

2.9.2.8.1.10 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.

113]Operate with the conjugate transpose of the matrix.

2.9.2.8.1.11 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

2.9.2.8.1.12 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.

2.9.2.8.1.13 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

2.9.2.8.1.14 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

2.9.2.8.1.15 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

2.9.2.8.1.16 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

2.9.2.8.1.17 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

2.9.2.8.1.18 rocblas_gemm_algo

enum rocblas_gemm_algo
Indicates if layer is active with bitmask.

Values:

rocblas_gemm_algo_standard = 0b0000000000

2.9.2.8.2 Functions

2.9.2.8.2.1 Level 1 BLAS

2.9.2.8.2.2 rocblas_<type>scal()

ROCBLAS_EXPORT rocblas_status rocblas_dscal(rocblas_handle handle, rocblas_int n, const dou

ROCBLAS_EXPORT rocblas_status rocblas_sscal(rocblas_handle handle, rocblas_int n, const flo
BLAS Level 1 API.

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

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

Parameters

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `n`: rocblas_int.
- `alpha`: specifies the scalar alpha.

- `x`: pointer storing vector x on the GPU.
- `incx`: specifies the increment for the elements of x.

2.9.2.8.2.3 rocblas_<type>copy()

ROCBLAS_EXPORT rocblas_status rocblas_dcopy(rocblas_handle handle, rocblas_int n, const dou

ROCBLAS_EXPORT rocblas_status rocblas_scopy(rocblas_handle handle, rocblas_int n, const flo

BLAS Level 1 API.

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

```
y := x,
```

Parameters

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

2.9.2.8.2.4 rocblas_<type>dot()

ROCBLAS_EXPORT rocblas_status rocblas_ddot(rocblas_handle handle, rocblas_int n, const doub

ROCBLAS_EXPORT rocblas_status rocblas_sdot(rocblas_handle handle, rocblas_int n, const floa

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

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

2.9.2.8.2.5 rocblas_<type>swap()

ROCBLAS_EXPORT rocblas_status rocblas_sswap(rocblas_handle handle, rocblas_int n, float * x

BLAS Level 1 API.

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

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

Parameters

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

ROCBLAS_EXPORT rocblas_status rocblas_dswap(rocblas_handle handle, rocblas_int n, double *

2.9.2.8.2.6 rocblas_<type>axpy()

ROCBLAS_EXPORT rocblas_status rocblas_daxpy(rocblas_handle handle, rocblas_int n, const dou

ROCBLAS_EXPORT rocblas_status rocblas_saxpy(rocblas_handle handle, rocblas_int n, const flo

ROCBLAS_EXPORT rocblas_status rocblas_haxpy(rocblas_handle handle, rocblas_int n, const roc

BLAS Level 1 API.

axpy compute y := alpha * x + y

Parameters

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

2.9.2.8.2.7 rocblas_<type>asum()

ROCBLAS_EXPORT rocblas_status rocblas_dasum(rocblas_handle handle, rocblas_int n, const dou

ROCBLAS_EXPORT rocblas_status rocblas_sasum(rocblas_handle handle, rocblas_int n, const flo

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

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

2.9.2.8.2.8 rocblas_<type>nrm2()

ROCBLAS_EXPORT rocblas_status rocblas_dnrm2(rocblas_handle handle, rocblas_int n, const dou

ROCBLAS_EXPORT rocblas_status rocblas_snrm2(rocblas_handle handle, rocblas_int n, const flo

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

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

2.9.2.8.2.9 rocblas_i<type>amax()

ROCBLAS_EXPORT rocblas_status rocblas_idamax(rocblas_handle handle, rocblas_int n, const do

ROCBLAS_EXPORT rocblas_status rocblas_isamax(rocblas_handle handle, rocblas_int n, const fl

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

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

2.9.2.8.2.10 rocblas_i<type>amin()

ROCBLAS_EXPORT rocblas_status rocblas_idamin(rocblas_handle handle, rocblas_int n, const do

ROCBLAS_EXPORT rocblas_status rocblas_isamin(rocblas_handle handle, rocblas_int n, const fl
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

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

2.9.2.8.2.11 Level 2 BLAS

2.9.2.8.2.12 rocblas_<type>gemv()

ROCBLAS_EXPORT rocblas_status rocblas_dgemv(rocblas_handle handle, rocblas_operation trans,

ROCBLAS_EXPORT rocblas_status rocblas_sgemv(rocblas_handle handle, rocblas_operation trans,
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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `trans`: rocblas_operation
- `m`: rocblas_int
- `n`: rocblas_int
- `alpha`: specifies the scalar alpha.
- `A`: pointer storing matrix A on the GPU.
- `lda`: rocblas_int specifies the leading dimension of A.
- `x`: pointer storing vector x on the GPU.
- `incx`: specifies the increment for the elements of x.
- `beta`: specifies the scalar beta.
- `y`: pointer storing vector y on the GPU.

- incy: rocblas_int specifies the increment for the elements of y.

2.9.2.8.2.13 rocblas_<type>ger()

ROCBLAS_EXPORT rocblas_status rocblas_dger(rocblas_handle handle, rocblas_int m, rocblas_in

ROCBLAS_EXPORT rocblas_status rocblas_sger(rocblas_handle handle, rocblas_int m, rocblas_in

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

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

- handle: rocblas_handle. handle to the rocblas library context queue.
- m: rocblas_int
- n: rocblas_int
- alpha: specifies the scalar alpha.
- x: pointer storing vector x on the GPU.
- incx: rocblas_int specifies the increment for the elements of x.
- y: pointer storing vector y on the GPU.
- incy: rocblas_int specifies the increment for the elements of y.
- A: pointer storing matrix A on the GPU.

- `lda`: rocblas_int specifies the leading dimension of A.

2.9.2.8.2.14 rocblas_<type>syr()

ROCBLAS_EXPORT rocblas_status rocblas_dsyr(rocblas_handle handle, rocblas_fill uplo, rocbla

ROCBLAS_EXPORT rocblas_status rocblas_ssyr(rocblas_handle handle, rocblas_fill uplo, rocbla

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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `n`: rocblas_int
- `alpha`: specifies the scalar alpha.
- `x`: pointer storing vector x on the GPU.
- `incx`: rocblas_int specifies the increment for the elements of x.
- `A`: pointer storing matrix A on the GPU.
- `lda`: rocblas_int specifies the leading dimension of A.

2.9.2.8.2.15 Level 3 BLAS

2.9.2.8.2.16 rocblas_<type>trtri_batched()

ROCBLAS_EXPORT rocblas_status rocblas_dtrtri_batched(rocblas_handle handle, rocblas_fill up

ROCBLAS_EXPORT rocblas_status rocblas_strtri_batched(rocblas_handle handle, rocblas_fill up

BLAS Level 3 API.

trtri compute the inverse of a matrix A

```
inv(A);
```

Parameters

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `uplo`: rocblas_fill. specifies whether the upper 'rocblas_fill_upper' or lower 'rocblas_fill_lower'
- `diag`: rocblas_diagonal. = 'rocblas_diagonal_non_unit', A is non-unit triangular; = 'rocblas_diagonal_unit', A is unit triangular;
- `n`: rocblas_int.
- `A`: pointer storing matrix A on the GPU.
- `lda`: rocblas_int specifies the leading dimension of A.
- `bsa`: rocblas_int "batch stride a": stride from the start of one "A" matrix to the next

-

2.9.2.8.2.17 rocblas_<type>trsm()

ROCBLAS_EXPORT rocblas_status rocblas_dtrsm(rocblas_handle handle, rocblas_side side, rocbl

ROCBLAS_EXPORT rocblas_status rocblas_strsm(rocblas_handle handle, rocblas_side side, rocbl

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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `side`: rocblas_side. rocblas_side_left: op(A)*X = alpha*B. rocblas_side_right: X*op(A) = alpha*B.
- `uplo`: rocblas_fill. rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
- `transA`: rocblas_operation. transB: op(A) = A. rocblas_operation_transpose: op(A) = A^T. rocblas_operation_conjugate_transpose: op(A) = A^H.
- `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.
- `m`: rocblas_int. m specifies the number of rows of B. m >= 0.
- `n`: rocblas_int. n specifies the number of columns of B. n >= 0.
- `alpha`: alpha specifies the scalar alpha. When alpha is &zero then A is not referenced and B need not be set before entry.
- `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.
- `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).
-

2.9.2.8.2.18 rocblas_<type>gemm()

ROCBLAS_EXPORT rocblas_status rocblas_dgemm(rocblas_handle handle, rocblas_operation transa

ROCBLAS_EXPORT rocblas_status rocblas_sgemm(rocblas_handle handle, rocblas_operation transa

ROCBLAS_EXPORT rocblas_status rocblas_hgemm(rocblas_handle handle, rocblas_operation transa

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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `transA`: rocblas_operation specifies the form of op(A)
- `transB`: rocblas_operation specifies the form of op(B)
- `m`: rocblas_int.
- `n`: rocblas_int.
- `k`: rocblas_int.
- `alpha`: specifies the scalar alpha.
- `A`: pointer storing matrix A on the GPU.
- `lda`: rocblas_int specifies the leading dimension of A.
- `B`: pointer storing matrix B on the GPU.
- `ldb`: rocblas_int specifies the leading dimension of B.
- `beta`: specifies the scalar beta.
- `C`: pointer storing matrix C on the GPU.
- `ldc`: rocblas_int specifies the leading dimension of C.

2.9.2.8.2.19 rocblas_<type>gemm_strided_batched()

```
ROCBLAS_EXPORT rocblas_status rocblas_dgemm_strided_batched(rocblas_handle handle, rocblas_
ROCBLAS_EXPORT rocblas_status rocblas_sgemm_strided_batched(rocblas_handle handle, rocblas_
ROCBLAS_EXPORT rocblas_status rocblas_hgemm_strided_batched(rocblas_handle handle, rocblas_
```

2.9.2.8.2.20 rocblas_<type>gemm_kernel_name()

```
ROCBLAS_EXPORT rocblas_status rocblas_dgemm_kernel_name(rocblas_handle handle, rocblas_oper
ROCBLAS_EXPORT rocblas_status rocblas_sgemm_kernel_name(rocblas_handle handle, rocblas_oper
ROCBLAS_EXPORT rocblas_status rocblas_hgemm_kernel_name(rocblas_handle handle, rocblas_oper
```

2.9.2.8.2.21 rocblas_<type>geam()

ROCBLAS_EXPORT rocblas_status rocblas_dgeam(rocblas_handle handle, rocblas_operation transa

ROCBLAS_EXPORT rocblas_status rocblas_sgeam(rocblas_handle handle, rocblas_operation transa

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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `transA`: rocblas_operation specifies the form of op(A)
- `transB`: rocblas_operation specifies the form of op(B)
- `m`: rocblas_int.
- `n`: rocblas_int.
- `alpha`: specifies the scalar alpha.
- `A`: pointer storing matrix A on the GPU.
- `lda`: rocblas_int specifies the leading dimension of A.
- `beta`: specifies the scalar beta.
- `B`: pointer storing matrix B on the GPU.
- `ldb`: rocblas_int specifies the leading dimension of B.
- `C`: pointer storing matrix C on the GPU.
- `ldc`: rocblas_int specifies the leading dimension of C.

2.9.2.8.2.22 BLAS Extensions

2.9.2.8.2.23 rocblas_gemm_ex()

ROCBLAS_EXPORT rocblas_status rocblas_gemm_ex(rocblas_handle handle, rocblas_operation tra

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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `transA`: rocblas_operation specifies the form of op(A)
- `transB`: rocblas_operation specifies the form of op(B)
- `m`: rocblas_int. matrix dimension m
- `n`: rocblas_int. matrix dimension n
- `k`: rocblas_int. matrix dimension k
- `alpha`: const void * specifies the scalar alpha. Same datatype as compute_type.
- `a`: void * pointer storing matrix A on the GPU.
- `a_type`: rocblas_datatype specifies the datatype of matrix A
- `lda`: rocblas_int specifies the leading dimension of A.
- `b`: void * pointer storing matrix B on the GPU.
- `b_type`: rocblas_datatype specifies the datatype of matrix B
- `ldb`: rocblas_int specifies the leading dimension of B.
- `beta`: const void * specifies the scalar beta. Same datatype as compute_type.
- `c`: void * pointer storing matrix C on the GPU.
- `c_type`: rocblas_datatype specifies the datatype of matrix C
- `ldc`: rocblas_int specifies the leading dimension of C.
- `d`: void * pointer storing matrix D on the GPU.
- `d_type`: rocblas_datatype specifies the datatype of matrix D
- `ldd`: rocblas_int specifies the leading dimension of D.
- `compute_type`: rocblas_datatype specifies the datatype of computation
- `algo`: rocblas_gemm_algo enumerant specifying the algorithm type.
- `solution_index`: int32_t reserved for future use
- `flags`: uint32_t reserved for future use
-

2.9.2.8.2.24 rocblas_gemm_strided_batched_ex()

ROCBLAS_EXPORT rocblas_status rocblas_strsm(rocblas_handle handle, rocblas_side side, rocbl

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

- `handle`: rocblas_handle. handle to the rocblas library context queue.
- `side`: rocblas_side. rocblas_side_left: op(A)*X = alpha*B. rocblas_side_right: X*op(A) = alpha*B.
- `uplo`: rocblas_fill. rocblas_fill_upper: A is an upper triangular matrix. rocblas_fill_lower: A is a lower triangular matrix.
- `transA`: rocblas_operation. transB: op(A) = A. rocblas_operation_transpose: op(A) = A^T. rocblas_operation_conjugate_transpose: op(A) = A^H.
- `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.
- `m`: rocblas_int. m specifies the number of rows of B. m >= 0.
- `n`: rocblas_int. n specifies the number of columns of B. n >= 0.
- `alpha`: alpha specifies the scalar alpha. When alpha is &zero then A is not referenced and B need not be set before entry.
- `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.
- `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).
-

2.9.2.8.2.25 Build Information

2.9.2.8.2.26 rocblas_get_version_string()

ROCBLAS_EXPORT rocblas_status rocblas_get_version_string(char * buf, size_t len)
loads char* buf with the rocblas library version. size_t len is the maximum length of char* buf.

Parameters

- `buf`: pointer to buffer for version string
- `len`: length of buf

2.9.2.8.2.27 Auxiliary

2.9.2.8.2.28 rocblas_pointer_to_mode()

ROCBLAS_EXPORT 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

2.9.2.8.2.29 rocblas_create_handle()

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

2.9.2.8.2.30 rocblas_destroy_handle()

```
ROCBLAS_EXPORT rocblas_status rocblas_destroy_handle(rocblas_handle handle)
```

2.9.2.8.2.31 rocblas_add_stream()

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

2.9.2.8.2.32 rocblas_set_stream()

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

2.9.2.8.2.33 rocblas_get_stream()

```
ROCBLAS_EXPORT rocblas_status rocblas_get_stream(rocblas_handle handle, hipStream_t * strea
```

2.9.2.8.2.34 rocblas_set_pointer_mode()

```
ROCBLAS_EXPORT rocblas_status rocblas_set_pointer_mode(rocblas_handle handle, rocblas_point
```

2.9.2.8.2.35 rocblas_get_pointer_mode()

```
ROCBLAS_EXPORT rocblas_status rocblas_get_pointer_mode(rocblas_handle handle, rocblas_point
```

2.9.2.8.2.36 rocblas_set_vector()

```
ROCBLAS_EXPORT rocblas_status rocblas_set_vector(rocblas_int n, rocblas_int elem_size, cons
```

2.9.2.8.2.37 rocblas_get_vector()

```
ROCBLAS_EXPORT rocblas_status rocblas_get_vector(rocblas_int n, rocblas_int elem_size, cons
```

2.9.2.8.2.38 rocblas_set_matrix()

```
ROCBLAS_EXPORT rocblas_status rocblas_set_matrix(rocblas_int rows, rocblas_int cols, rocbla
```

2.9.2.8.2.39 rocblas_get_matrix()

```
ROCBLAS_EXPORT rocblas_status rocblas_get_matrix(rocblas_int rows, rocblas_int cols, rocbla
```

2.9.3 hipBLAS

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 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
```

2.9.3.2 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 wiki has helpful information on how to configure cmake and manually build.

Functions supported

A list of exported functions from hipblas can be found on the wiki

2.9.3.3 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

2.9.3.4 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 );
```

2.9.3.5 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 hcRNG

2.9.4.1 Introduction

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.

The following list enumerates the current set of RNG generators that are supported so far.

- MRG31k3p
- MRG32k3a
- LFSR113
- Philox-4x32-10

2.9.4.2 Examples

Random number generator Mrg31k3p example:

file: Randomarray.cpp

#!c++

```
//This example is a simple random array generation and it compares host output with␣
↪device output
//Random number generator Mrg31k3p
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <assert.h>
#include <hcRNG/mrg31k3p.h>
#include <hcRNG/hcRNG.h>
#include <hc.hpp>
#include <hc_am.hpp>
using namespace hc;
```

(continues on next page)

(continued from previous page)

```
int main()
{
      hcrngStatus status = HCRNG_SUCCESS;
      bool ispassed = 1;
      size_t streamBufferSize;
      // Number oi streams
      size_t streamCount = 10;
      //Number of random numbers to be generated
      //numberCount must be a multiple of streamCount
      size_t numberCount = 100;
      //Enumerate the list of accelerators
      std::vector<hc::accelerator>acc = hc::accelerator::get_all();
      accelerator_view accl_view = (acc[1].create_view());
      //Allocate memory for host pointers
      float *Random1 = (float*) malloc(sizeof(float) * numberCount);
      float *Random2 = (float*) malloc(sizeof(float) * numberCount);
      float *outBufferDevice = hc::am_alloc(sizeof(float) * numberCount, acc[1], 0);

      //Create streams
      hcrngMrg31k3pStream *streams = hcrngMrg31k3pCreateStreams(NULL, streamCount, &
↪streamBufferSize, NULL);
      hcrngMrg31k3pStream *streams_buffer = hc::am_alloc(sizeof(hcrngMrg31k3pStream)␣
↪* streamCount, acc[1], 0);
      accl_view.copy(streams, streams_buffer, streamCount*␣
↪sizeof(hcrngMrg31k3pStream));

      //Invoke random number generators in device (here strean_length and streams_per_
↪thread arguments are default)
      status = hcrngMrg31k3pDeviceRandomU01Array_single(accl_view, streamCount,␣
↪streams_buffer, numberCount, outBufferDevice);

      if(status) std::cout << "TEST FAILED" << std::endl;
      accl_view.copy(outBufferDevice, Random1, numberCount * sizeof(float));

      //Invoke random number generators in host
      for (size_t i = 0; i < numberCount; i++)
        Random2[i] = hcrngMrg31k3pRandomU01(&streams[i % streamCount]);
      // Compare host and device outputs
      for(int i =0; i < numberCount; i++) {
          if (Random1[i] != Random2[i]) {
              ispassed = 0;
              std::cout <<" RANDDEVICE[" << i<< "] " << Random1[i] << "and RANDHOST["
↪<< i <<"] mismatches"<< Random2[i] <<                    std::endl;
              break;
          }
          else
              continue;
      }
      if(!ispassed) std::cout << "TEST FAILED" << std::endl;

      //Free host resources
      free(Random1);
      free(Random2);
      //Release device resources
      hc::am_free(outBufferDevice);
      hc::am_free(streams_buffer);
      return 0;
```

(continues on next page)

(continued from previous page)

```
}
```

- Compiling the example code:

  /opt/hcc/bin/clang++ *`/opt/hcc/bin/hcc-config –cxxflags –ldflags`* -lhc_am -lhcrng Randomarray.cpp

2.9.4.3 Installation

Installation steps

The following are the steps to use the library

- ROCM 2.0 Kernel, Driver and Compiler Installation (if not done until now)
- Library installation.

ROCM 2.0 Installation

To Know more about ROCM refer https://rocm-documentation.readthedocs.io/en/latest/Current_Release_Notes/Current-Release-Notes.html

a. Installing Debian ROCM repositories

Before proceeding, make sure to completely uninstall any pre-release ROCm packages.

Refer Here for instructions to remove pre-release ROCM packages

Follow Steps to install rocm package

```
wget -qO - http://packages.amd.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
sudo sh -c 'echo deb [arch=amd64] http://packages.amd.com/rocm/apt/debian/ xenial␣
↪main > /etc/apt/sources.list.d/rocm.list'
sudo apt-get update
sudo apt-get install rocm
```

Then, make the ROCm kernel your default kernel. If using grub2 as your bootloader, you can edit the GRUB_DEFAULT variable in the following file:

```
sudo vi /etc/default/grub
sudo update-grub
```

and **Reboot the system**

b. Verifying the Installation

Once Reboot, to verify that the ROCm stack completed successfully you can execute HSA vector_copy sample application:

```
cd /opt/rocm/hsa/sample
make
./vector_copy
```

Library Installation

a. Install using Prebuilt debian

```
wget https://github.com/ROCmSoftwarePlatform/hcRNG/blob/master/pre-builds/hcrng-
↪master-184472e-Linux.deb
sudo dpkg -i hcrng-master-184472e-Linux.deb
```

b. Build debian from source

```
git clone https://github.com/ROCmSoftwarePlatform/hcRNG.git && cd hcRNG
chmod +x build.sh && ./build.sh
```

build.sh execution builds the library and generates a debian under build directory.

2.9.4.4 Key Features

- Support for 4 commonly used uniform random number generators.
- Single and Double precision.
- Multiple streams, created on the host and generates random numbers either on the host or on computing devices.

Prerequisites

This section lists the known set of hardware and software requirements to build this library

Hardware

- CPU: mainstream brand, Better if with >=4 Cores Intel Haswell based CPU
- System Memory >= 4GB (Better if >10GB for NN application over multiple GPUs)
- Hard Drive > 200GB (Better if SSD or NVMe driver for NN application over multiple GPUs)
- Minimum GPU Memory (Global) > 2GB

GPU cards supported

- dGPU: AMD R9 Fury X, R9 Fury, R9 Nano
- APU: AMD Kaveri or Carrizo

AMD Driver and Runtime

- Radeon Open Compute Kernel (ROCK) driver : https://github.com/RadeonOpenCompute/ROCK-Kernel-Driver
- HSA runtime API and runtime for Boltzmann: https://github.com/RadeonOpenCompute/ROCR-Runtime

System software

- Ubuntu 14.04 trusty and later
- GCC 4.6 and later
- CPP 4.6 and later (come with GCC package)
- python 2.7 and later
- python-pip
- BeautifulSoup4 (installed using python-pip)
- HCC 0.9 from here

Tools and Misc

- git 1.9 and later
- cmake 2.6 and later (2.6 and 2.8 are tested)
- firewall off
- root privilege or user account in sudo group

Ubuntu Packages

- libc6-dev-i386
- liblapack-dev
- graphicsmagick
- libblas-dev

2.9.4.5 Tested Environments

Driver versions

- **Boltzmann Early Release Driver + dGPU**
  - Radeon Open Compute Kernel (ROCK) driver : https://github.com/RadeonOpenCompute/ROCK-Kernel-Driver
  - HSA runtime API and runtime for Boltzmann: https://github.com/RadeonOpenCompute/ROCR-Runtime
- Traditional HSA driver + APU (Kaveri)

GPU Cards

- Radeon R9 Nano
- Radeon R9 FuryX
- Radeon R9 Fury
- Kaveri and Carizo APU

Server System

- Supermicro SYS 2028GR-THT 6 R9 NANO
- Supermicro SYS-1028GQ-TRT 4 R9 NANO
- Supermicro SYS-7048GR-TR Tower 4 R9 NANO

2.9.4.6 Unit testing

a) Automated testing:

Follow these steps to start automated testing:

```
cd ~/hcRNG/
./build.sh --test=on
```

b) Manual testing:

(i) Google testing (GTEST) with Functionality check

```
cd ~/hcRNG/build/test/unit/bin/
```

All functions are tested against google test.

2.9.5 hipeigen

Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

For more information go to http://eigen.tuxfamily.org/.

2.9.5.1 Installation instructions for ROCm

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.

To insatll rocm, please follow:

2.9.5.2 Installing from AMD ROCm repositories

AMD is hosting both debian and rpm repositories for the ROCm 2.0 packages. The packages in both repositories have been signed to ensure package integrity. Directions for each repository are given below:

- Debian repository - apt-get
- Add the ROCm apt repository

Complete installation steps of ROCm can be found Here

or

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

```
wget -qO - http://packages.amd.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
sudo sh -c 'echo deb [arch=amd64] http://packages.amd.com/rocm/apt/debian/ xenial␣
↪main > /etc/apt/sources.list.d/rocm.list'
```

The gpg key might change, so it may need to be updated when installing a new release.

Install or Update

Next, update the apt-get repository list and install/update the rocm package:

> **Warning:** Before proceeding, make sure to completely uninstall any pre-release ROCm packages

:: sudo apt-get update sudo apt-get install rocm

Then, make the ROCm kernel your default kernel. If using grub2 as your bootloader, you can edit the GRUB_DEFAULT variable in the following file:

:: sudo vi /etc/default/grub sudo update-grub

Once complete, **reboot your system.**

We recommend you verify your installation to make sure everything completed successfully.

2.9.5.3 Installation instructions for Eigen

Explanation before starting

Eigen consists only of header files, hence there is nothing to compile before you can use it. Moreover, these header files do not depend on your platform, they are the same for everybody.

Method 1. Installing without using CMake

You can use right away the headers in the Eigen/ subdirectory. In order to install, just copy this Eigen/ subdirectory to your favorite location. If you also want the unsupported features, copy the unsupported/ subdirectory too.

Method 2. Installing using CMake

Let's call this directory 'source_dir' (where this INSTALL file is). Before starting, create another directory which we will call 'build_dir'.

Do:

```
cd build_dir
cmake source_dir
make install
```

The make install step may require administrator privileges.

You can adjust the installation destination (the "prefix") by passing the -DCMAKE_INSTALL_PREFIX=myprefix option to cmake, as is explained in the message that cmake prints at the end.

2.9.5.4 Build and Run hipeigen direct tests

To build the direct tests for hipeigen:

```
cd build_dir
make check -j $(nproc)
```

Note: All direct tests should pass with ROCm2.0

2.9.6 clFFT

For Github Repository 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.

Pre-built binaries are available here.

2.9.6.1 Introduction to clFFT

The FFT is an implementation of the Discrete Fourier Transform (DFT) that makes use of symmetries in the FFT definition to reduce the mathematical intensity required from O(N^2) to O(N log2(N)) when the sequence length N is the product of small prime factors. Currently, there is no standard API for FFT routines. Hardware vendors usually provide a set of high-performance FFTs optimized for their systems: no two vendors employ the same interfaces for their FFT routines. clFFT provides a set of FFT routines that are optimized for AMD graphics processors, but also are functional across CPU and other compute devices.

The clFFT library is an open source OpenCL library implementation of discrete Fast Fourier Transforms. The library:

- provides a fast and accurate platform for calculating discrete FFTs.
- works on CPU or GPU backends.
- supports in-place or out-of-place transforms.
- supports 1D, 2D, and 3D transforms with a batch size that can be greater than 1.
- supports planar (real and complex components in separate arrays) and interleaved (real and complex components as a pair contiguous in memory) formats.

- supports dimension lengths that can be any combination of powers of 2, 3, 5, 7, 11 and 13.
- Supports single and double precision floating point formats.

2.9.6.2 clFFT library user documentation

Library and API documentation for developers is available online as a GitHub Pages website

2.9.6.3 API semantic versioning

Good software is typically the result of the loop of feedback and iteration; software interfaces no less so. clFFT follows the semantic versioning guidelines. The version number used is of the form MAJOR.MINOR.PATCH.

2.9.6.4 clFFT Wiki

The project wiki contains helpful documentation, including a build primer

2.9.6.5 Contributing code

Please refer to and read the Contributing document for guidelines on how to contribute code to this open source project. The code in the /master branch is considered to be stable, and all pull-requests must be made against the /develop branch.

2.9.6.6 License

The source for clFFT is licensed under the Apache License , Version 2.0

2.9.6.7 Example

The following simple example shows how to use clFFT to compute a simple 1D forward transform

```
#include <stdlib.h>

/* No need to explicitely include the OpenCL headers */
#include <clFFT.h>

int main( void )
{
   cl_int err;
   cl_platform_id platform = 0;
   cl_device_id device = 0;
   cl_context_properties props[3] = { CL_CONTEXT_PLATFORM, 0, 0 };
   cl_context ctx = 0;
   cl_command_queue queue = 0;
   cl_mem bufX;
       float *X;
   cl_event event = NULL;
   int ret = 0;
       size_t N = 16;

       /* FFT library realted declarations */
```

(continues on next page)

(continued from previous page)

```
      clfftPlanHandle planHandle;
      clfftDim dim = CLFFT_1D;
      size_t clLengths[1] = {N};

  /* Setup OpenCL environment. */
  err = clGetPlatformIDs( 1, &platform, NULL );
  err = clGetDeviceIDs( platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL );

  props[1] = (cl_context_properties)platform;
  ctx = clCreateContext( props, 1, &device, NULL, NULL, &err );
  queue = clCreateCommandQueue( ctx, device, 0, &err );

  /* Setup clFFT. */
      clfftSetupData fftSetup;
      err = clfftInitSetupData(&fftSetup);
      err = clfftSetup(&fftSetup);

      /* Allocate host & initialize data. */
      /* Only allocation shown for simplicity. */
      X = (float *)malloc(N * 2 * sizeof(*X));

  /* Prepare OpenCL memory objects and place data inside them. */
  bufX = clCreateBuffer( ctx, CL_MEM_READ_WRITE, N * 2 * sizeof(*X), NULL, &err );

  err = clEnqueueWriteBuffer( queue, bufX, CL_TRUE, 0,
      N * 2 * sizeof( *X ), X, 0, NULL, NULL );

      /* Create a default plan for a complex FFT. */
      err = clfftCreateDefaultPlan(&planHandle, ctx, dim, clLengths);

      /* Set plan parameters. */
      err = clfftSetPlanPrecision(planHandle, CLFFT_SINGLE);
      err = clfftSetLayout(planHandle, CLFFT_COMPLEX_INTERLEAVED, CLFFT_COMPLEX_
↪INTERLEAVED);
      err = clfftSetResultLocation(planHandle, CLFFT_INPLACE);

  /* Bake the plan. */
      err = clfftBakePlan(planHandle, 1, &queue, NULL, NULL);

      /* Execute the plan. */
      err = clfftEnqueueTransform(planHandle, CLFFT_FORWARD, 1, &queue, 0, NULL,␣
↪NULL, &bufX, NULL, NULL);

      /* Wait for calculations to be finished. */
      err = clFinish(queue);

      /* Fetch results of calculations. */
      err = clEnqueueReadBuffer( queue, bufX, CL_TRUE, 0, N * 2 * sizeof( *X ), X, 0,
↪ NULL, NULL );

  /* Release OpenCL memory objects. */
  clReleaseMemObject( bufX );

      free(X);

      /* Release the plan. */
      err = clfftDestroyPlan( &planHandle );
```

(continues on next page)

(continued from previous page)

```
  /* Release clFFT library. */
  clfftTeardown( );

  /* Release OpenCL working objects. */
  clReleaseCommandQueue( queue );
  clReleaseContext( ctx );

  return ret;
}
```

2.9.6.8 Build dependencies

Library for Windows

To develop the clFFT library code on a Windows operating system, ensure to install the following packages on your system:

- Windows® 7/8.1
- Visual Studio 2012 or later
- Latest CMake
- An OpenCL SDK, such as APP SDK 3.0

Library for Linux

To develop the clFFT library code on a Linux operating system, ensure to install the following packages on your system:

- GCC 4.6 and onwards
- Latest CMake
- An OpenCL SDK, such as APP SDK 3.0

Library for Mac OSX

To develop the clFFT library code on a Mac OS X, it is recommended to generate Unix makefiles with cmake.

Test infrastructure

To test the developed clFFT library code, ensure to install the following packages on your system:

- Googletest v1.6
- Latest FFTW
- Latest Boost

2.9.6.9 Performance infrastructure

To measure the performance of the clFFT library code, ensure that the Python package is installed on your system.

2.9.7 clBLAS

For Github repository 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.

clBLAS update notes 01/2017

v2.12 is a bugfix release as a rollup of all fixes in /develop branch Thanks to @pavanky, @iotamudelta, @shahsan10, @psyhtest, @haahh, @hughperkins, @tfauck @abhiShandy, @IvanVergiliev, @zougloub, @mgates3 for contributions to clBLAS v2.12 Summary of fixes available to read on the releases tab

2.9.7.1 clBLAS library user documentation

Library and API documentation for developers is available online as a GitHub Pages website

clBLAS Wiki

The project wiki contains helpful documentation, including a build primer

Contributing code

Please refer to and read the Contributing document for guidelines on how to contribute code to this open source project. The code in the /master branch is considered to be stable, and all pull-requests should be made against the /develop branch.

2.9.7.2 License

The source for clBLAS is licensed under the Apache License, Version 2.0

2.9.7.3 Example

The simple example below shows how to use clBLAS to compute an OpenCL accelerated SGEMM

```
#include <sys/types.h>
#include <stdio.h>

/* Include the clBLAS header. It includes the appropriate OpenCL headers */
#include <clBLAS.h>

/* This example uses predefined matrices and their characteristics for
 * simplicity purpose.
*/

#define M  4
#define N  3
#define K  5

static const cl_float alpha = 10;
```

(continues on next page)

(continued from previous page)

```
static const cl_float A[M*K] = {
11, 12, 13, 14, 15,
21, 22, 23, 24, 25,
31, 32, 33, 34, 35,
41, 42, 43, 44, 45,
};
static const size_t lda = K;        /* i.e. lda = K */

static const cl_float B[K*N] = {
11, 12, 13,
21, 22, 23,
31, 32, 33,
41, 42, 43,
51, 52, 53,
};
static const size_t ldb = N;        /* i.e. ldb = N */

static const cl_float beta = 20;

static cl_float C[M*N] = {
    11, 12, 13,
    21, 22, 23,
    31, 32, 33,
    41, 42, 43,
};
static const size_t ldc = N;        /* i.e. ldc = N */

static cl_float result[M*N];

int main( void )
{
cl_int err;
cl_platform_id platform = 0;
cl_device_id device = 0;
cl_context_properties props[3] = { CL_CONTEXT_PLATFORM, 0, 0 };
cl_context ctx = 0;
cl_command_queue queue = 0;
cl_mem bufA, bufB, bufC;
cl_event event = NULL;
int ret = 0;

/* Setup OpenCL environment. */
err = clGetPlatformIDs( 1, &platform, NULL );
err = clGetDeviceIDs( platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL );

props[1] = (cl_context_properties)platform;
ctx = clCreateContext( props, 1, &device, NULL, NULL, &err );
queue = clCreateCommandQueue( ctx, device, 0, &err );

/* Setup clBLAS */
err = clblasSetup( );

/* Prepare OpenCL memory objects and place matrices inside them. */
bufA = clCreateBuffer( ctx, CL_MEM_READ_ONLY, M * K * sizeof(*A),
                       NULL, &err );
bufB = clCreateBuffer( ctx, CL_MEM_READ_ONLY, K * N * sizeof(*B),
                       NULL, &err );
```

(continues on next page)

(continued from previous page)

```
bufC = clCreateBuffer( ctx, CL_MEM_READ_WRITE, M * N * sizeof(*C),
                       NULL, &err );

err = clEnqueueWriteBuffer( queue, bufA, CL_TRUE, 0,
    M * K * sizeof( *A ), A, 0, NULL, NULL );
err = clEnqueueWriteBuffer( queue, bufB, CL_TRUE, 0,
    K * N * sizeof( *B ), B, 0, NULL, NULL );
err = clEnqueueWriteBuffer( queue, bufC, CL_TRUE, 0,
    M * N * sizeof( *C ), C, 0, NULL, NULL );

    /* Call clBLAS extended function. Perform gemm for the lower right sub-matrices */
    err = clblasSgemm( clblasRowMajor, clblasNoTrans, clblasNoTrans,
                            M, N, K,
                            alpha, bufA, 0, lda,
                            bufB, 0, ldb, beta,
                            bufC, 0, ldc,
                            1, &queue, 0, NULL, &event );

/* Wait for calculations to be finished. */
err = clWaitForEvents( 1, &event );

/* Fetch results of calculations from GPU memory. */
err = clEnqueueReadBuffer( queue, bufC, CL_TRUE, 0,
                            M * N * sizeof(*result),
                            result, 0, NULL, NULL );

/* Release OpenCL memory objects. */
clReleaseMemObject( bufC );
clReleaseMemObject( bufB );
clReleaseMemObject( bufA );

/* Finalize work with clBLAS */
clblasTeardown( );

/* Release OpenCL working objects. */
clReleaseCommandQueue( queue );
clReleaseContext( ctx );

return ret;
}
```

2.9.7.4 Build dependencies

Library for Windows

- Windows® 7/8
- Visual Studio 2010 SP1, 2012
- An OpenCL SDK, such as APP SDK 2.8
- Latest CMake

Library for Linux

- GCC 4.6 and onwards
- An OpenCL SDK, such as APP SDK 2.9

- Latest CMake

Library for Mac OSX

- Recommended to generate Unix makefiles with cmake

Test infrastructure

- Googletest v1.6
- Latest Boost
- CPU BLAS
- Netlib CBLAS (recommended) Ubuntu: install by "apt-get install libblas-dev" Windows: download & install lapack-3.6.0 which comes with CBLAS
- or ACML on windows/linux; Accelerate on Mac OSX

2.9.7.5 Performance infrastructure

Python

2.9.8 clSPARSE

For Github repository clSPARSE

an OpenCL™ library implementing Sparse linear algebra routines. This project is a result of a collaboration between AMD Inc. and Vratis Ltd..

2.9.8.1 What's new in clSPARSE v0.10.1

- **bug fix release**
  - Fixes for travis builds
  - Fix to the matrix market reader in the cuSPARSE benchmark to synchronize with the regular MM reader
  - Replace cl.hpp with cl2.hpp (thanks to arrayfire)
  - **Fixes for the Nvidia platform; tested 352.79**
    - Fixed buffer overruns in CSR-Adaptive kernels
    - Fix invalid memory access on Nvidia GPUs in CSR-Adaptive SpMV kernel

2.9.8.2 clSPARSE features

- Sparse Matrix - dense Vector multiply (SpM-dV)
- Sparse Matrix - dense Matrix multiply (SpM-dM)
- Sparse Matrix - Sparse Matrix multiply Sparse Matrix Multiply(SpGEMM) - Single Precision
- Iterative conjugate gradient solver (CG)
- Iterative biconjugate gradient stabilized solver (BiCGStab)
- Dense to CSR conversions (& converse)

- COO to CSR conversions (& converse)
- Functions to read matrix market files in COO or CSR format

True in spirit with the other clMath libraries, clSPARSE exports a "C" interface to allow projects to build wrappers around clSPARSE in any language they need. A great deal of thought and effort went into designing the API's to make them less 'cluttered' compared to the older clMath libraries. OpenCL state is not explicitly passed through the API, which enables the library to be forward compatible when users are ready to switch from OpenCL 1.2 to OpenCL 2.0 3

2.9.8.3 API semantic versioning

Good software is typically the result of iteration and feedback. clSPARSE follows the semantic versioning guidelines, and while the major version number remains '0', the public API should not be considered stable. We release clSPARSE as beta software (0.y.z) early to the community to elicit feedback and comment. This comes with the expectation that with feedback, we may incorporate breaking changes to the API that might require early users to recompile, or rewrite portions of their code as we iterate on the design.

clSPARSE Wiki

The project wiki contains helpful documentation. A build primer is available, which describes how to use cmake to generate platforms specific build files

Samples

clSPARSE contains a directory of simple OpenCL samples that demonstrate the use of the API in both C and C++. The superbuild script for clSPARSE also builds the samples as an external project, to demonstrate how an application would find and link to clSPARSE with cmake.

clSPARSE library documentation

API documentation is available at http://clmathlibraries.github.io/clSPARSE/. The samples give an excellent starting point to basic library operations.

Contributing code

Please refer to and read the Contributing document for guidelines on how to contribute code to this open source project. Code in the /master branch is considered to be stable and new library releases are made when commits are merged into /master. Active development and pull-requests should be made to the develop branch.

2.9.8.4 License

clSPARSE is licensed under the Apache License, Version 2.0

Compiling for Windows

- Windows® 7/8
- Visual Studio 2013 and above
- CMake 2.8.12 (download from Kitware)
- Solution (.sln) or
- Nmake makefiles
- An OpenCL SDK, such as APP SDK 3.0

Compiling for Linux

- GCC 4.8 and above
- CMake 2.8.12 (install with distro package manager)

- **Unix makefiles or**
  - KDevelop or
  - QT Creator
  - An OpenCL SDK, such as APP SDK 3.0

Compiling for Mac OSX

- CMake 2.8.12 (install via brew)
- Unix makefiles or
- XCode
- An OpenCL SDK (installed via xcode-select –install)

Bench & Test infrastructure dependencies

- Googletest v1.7
- Boost v1.58
- Footnotes

[1]: Changed to reflect CppCoreGuidelines: F.21

[2]: Changed to reflect CppCoreGuidelines: NL.8

[3]: OpenCL 2.0 support is not yet fully implemented; only the interfaces have been designed

2.9.9 clRNG

For Github repository 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.

2.9.9.1 What's New

Libraries related to clRNG, for probability distributions and quasi-Monte Carlo methods, are available:

- clProbDist
- clQMC

Releases

The first public version of clRNG is v1.0.0 beta. Please go to releases for downloads.

2.9.9.2 Building

1. **Install the runtime dependency:**
   - An OpenCL SDK, such as APP SDK.
2. Install the build dependencies:
   - The CMake cross-platform build system. Visual Studio users can use CMake Tools for Visual Studio.

- A recent C compiler, such as GCC 4.9 , or Visual Studio 2013.

3. Get the clRNG source code.
4. Configure the project using CMake (to generate standard makefiles) or CMake Tools for Visual Studio (to generate solution and project files).
5. Build the project.
6. Install the project (by default, the library will be installed in the package directory under the build directory).
7. Point the environment variable CLRNG_ROOT to the installation directory, i.e., the directory under which include/clRNG can be found. This step is optional if the library is installed under /usr, which is the default.
8. In order to execute the example programs (under the bin subdirectory of the installation directory) or to link clRNG into other software, the dynamic linker must be informed where to find the clRNG shared library. The name and location of the shared library generally depend on the platform.
9. Optionally run the tests.

2.9.9.3 Example Instructions for Linux

On a 64-bit Linux platform, steps 3 through 9 from above, executed in a Bash-compatible shell, could consist of:

```
git clone https://github.com/clMathLibraries/clRNG.git
mkdir clRNG.build; cd clRNG.build; cmake ../clRNG/src
make
make install
export CLRNG_ROOT=$PWD/package
export LD_LIBRARY_PATH=$CLRNG_ROOT/lib64:$LD_LIBRARY_PATH
$CLRNG_ROOT/bin/CTest
```

Examples

Examples can be found in src/client. The compiled client program examples can be found under the bin subdirectory of the installation package ($CLRNG_ROOT/bin under Linux). Note that the examples expect an OpenCL GPU device to be available.

Simple example

The simple example below shows how to use clRNG to generate random numbers by directly using device side headers (.clh) in your OpenCL kernel.

```
#include <stdlib.h>
#include <string.h>

#include "clRNG/clRNG.h"
#include "clRNG/mrg31k3p.h"

int main( void )
{
    cl_int err;
    cl_platform_id platform = 0;
    cl_device_id device = 0;
    cl_context_properties props[3] = { CL_CONTEXT_PLATFORM, 0, 0 };
    cl_context ctx = 0;
    cl_command_queue queue = 0;
    cl_program program = 0;
    cl_kernel kernel = 0;
    cl_event event = 0;
```

(continues on next page)

(continued from previous page)

```
    cl_mem bufIn, bufOut;
    float *out;
    char *clrng_root;
    char include_str[1024];
    char build_log[4096];
    size_t i = 0;
    size_t numWorkItems = 64;
    clrngMrg31k3pStream *streams = 0;
    size_t streamBufferSize = 0;
    size_t kernelLines = 0;

    /* Sample kernel that calls clRNG device-side interfaces to generate random␣
↪numbers */
    const char *kernelSrc[] = {
    "    #define CLRNG_SINGLE_PRECISION                                    \n",
    "    #include <clRNG/mrg31k3p.clh>                                     \n",
    "                                                                      \n",
    "    __kernel void example(__global clrngMrg31k3pHostStream *streams, \n",
    "                          __global float *out)                        \n",
    "    {                                                                 \n",
    "        int gid = get_global_id(0);                                   \n",
    "                                                                      \n",
    "        clrngMrg31k3pStream workItemStream;                           \n",
    "        clrngMrg31k3pCopyOverStreamsFromGlobal(1, &workItemStream,    \n",
    "                                               &streams[gid]);        \n",
    "                                                                      \n",
    "        out[gid] = clrngMrg31k3pRandomU01(&workItemStream);           \n",
    "    }                                                                 \n",
    "                                                                      \n",
    };

  /* Setup OpenCL environment. */
  err = clGetPlatformIDs( 1, &platform, NULL );
  err = clGetDeviceIDs( platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL );

  props[1] = (cl_context_properties)platform;
  ctx = clCreateContext( props, 1, &device, NULL, NULL, &err );
  queue = clCreateCommandQueue( ctx, device, 0, &err );

  /* Make sure CLRNG_ROOT is specified to get library path */
  clrng_root = getenv("CLRNG_ROOT");
  if(clrng_root == NULL) printf("\nSpecify environment variable CLRNG_ROOT as␣
↪described\n");
  strcpy(include_str, "-I ");
  strcat(include_str, clrng_root);
  strcat(include_str, "/include");

  /* Create sample kernel */
  kernelLines = sizeof(kernelSrc) / sizeof(kernelSrc[0]);
  program = clCreateProgramWithSource(ctx, kernelLines, kernelSrc, NULL, &err);
  err = clBuildProgram(program, 1, &device, include_str, NULL, NULL);
  if(err != CL_SUCCESS)
  {
      printf("\nclBuildProgram has failed\n");
      clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 4096, build_log,␣
↪NULL);
      printf("%s", build_log);
```

(continues on next page)

(continued from previous page)

```
  }
  kernel = clCreateKernel(program, "example", &err);

  /* Create streams */
  streams = clrngMrg31k3pCreateStreams(NULL, numWorkItems, &streamBufferSize,␣
↪(clrngStatus *)&err);

  /* Create buffers for the kernel */
  bufIn = clCreateBuffer(ctx, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,␣
↪streamBufferSize, streams, &err);
  bufOut = clCreateBuffer(ctx, CL_MEM_WRITE_ONLY | CL_MEM_HOST_READ_ONLY,␣
↪numWorkItems * sizeof(cl_float), NULL, &err);

  /* Setup the kernel */
  err = clSetKernelArg(kernel, 0, sizeof(bufIn),  &bufIn);
  err = clSetKernelArg(kernel, 1, sizeof(bufOut), &bufOut);

  /* Execute the kernel and read back results */
  err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &numWorkItems, NULL, 0, NULL, &
↪event);
  err = clWaitForEvents(1, &event);
  out = (float *)malloc(numWorkItems * sizeof(out[0]));
  err = clEnqueueReadBuffer(queue, bufOut, CL_TRUE, 0, numWorkItems * sizeof(out[0]),␣
↪out, 0, NULL, NULL);

  /* Release allocated resources */
  clReleaseEvent(event);
  free(out);
  clReleaseMemObject(bufIn);
  clReleaseMemObject(bufOut);

  clReleaseKernel(kernel);
  clReleaseProgram(program);

  clReleaseCommandQueue(queue);
  clReleaseContext(ctx);

  return 0;
}
```

2.9.9.4 Building the documentation manually

The documentation can be generated by running make from within the doc directory. This requires Doxygen to be installed.

2.9.10 hcFFT

2.9.10.1 Installation

The following are the steps to use the library

- ROCM 2.0 Kernel, Driver and Compiler Installation (if not done until now)
- Library installation.

ROCM 2.0 Installation

To Know more about ROCM refer https://github.com/RadeonOpenCompute/ROCm/blob/master/README.md

a. Installing Debian ROCM repositories

Before proceeding, make sure to completely uninstall any pre-release ROCm packages.

Refer https://github.com/RadeonOpenCompute/ROCm#removing-pre-release-packages for instructions to remove pre-release ROCM packages.

Steps to install rocm package are,

```
wget -qO - http://packages.amd.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -

sudo sh -c 'echo deb [arch=amd64] http://packages.amd.com/rocm/apt/debian/ xenial␣
↪main > /etc/apt/sources.list.d/rocm.list'

sudo apt-get update

sudo apt-get install rocm
```

Then, make the ROCm kernel your default kernel. If using grub2 as your bootloader, you can edit the GRUB_DEFAULT variable in the following file:

```
sudo vi /etc/default/grub

sudo update-grub
```

and Reboot the system

b. Verifying the Installation

Once Reboot, to verify that the ROCm stack completed successfully you can execute HSA vector_copy sample application:

- cd /opt/rocm/hsa/sample
- make
- ./vector_copy

Library Installation

a. Install using Prebuilt debian

```
wget https://github.com/ROCmSoftwarePlatform/hcFFT/blob/master/pre-builds/hcfft-
↪master-87a37f5-Linux.deb
sudo dpkg -i hcfft-master-87a37f5-Linux.deb
```

b. Build debian from source

```
git clone https://github.com/ROCmSoftwarePlatform/hcFFT.git && cd hcFFT

chmod +x build.sh && ./build.sh
```

build.sh execution builds the library and generates a debian under build directory.

c. Install CPU based FFTW3 library

```
sudo apt-get install fftw3 fftw3-dev pkg-config
```

2.9.10.2 Introduction

This repository hosts the HCC based FFT Library, that targets GPU acceleration of FFT routines on AMD devices. To know what HCC compiler features, refer here.

The following are the sub-routines that are implemented

1. R2C : Transforms Real valued input in Time domain to Complex valued output in Frequency domain.
2. C2R : Transforms Complex valued input in Frequency domain to Real valued output in Real domain.
3. C2C : Transforms Complex valued input in Frequency domain to Complex valued output in Real domain or vice versa

2.9.10.3 KeyFeature

- Support 1D, 2D and 3D Fast Fourier Transforms
- Supports R2C, C2R, C2C, D2Z, Z2D and Z2Z Transforms
- Support Out-Of-Place data storage
- Ability to Choose desired target accelerator
- Single and Double precision

Prerequisites

This section lists the known set of hardware and software requirements to build this library

Hardware

- CPU: mainstream brand, Better if with >=4 Cores Intel Haswell based CPU
- System Memory >= 4GB (Better if >10GB for NN application over multiple GPUs)
- Hard Drive > 200GB (Better if SSD or NVMe driver for NN application over multiple GPUs)
- Minimum GPU Memory (Global) > 2GB

GPU cards supported

- dGPU: AMD R9 Fury X, R9 Fury, R9 Nano
- APU: AMD Kaveri or Carrizo

AMD Driver and Runtime

- Radeon Open Compute Kernel (ROCK) driver : https://github.com/RadeonOpenCompute/ROCK-Kernel-Driver
- HSA runtime API and runtime for Boltzmann: https://github.com/RadeonOpenCompute/ROCR-Runtime

System software

- Ubuntu 14.04 trusty and later
- GCC 4.6 and later
- CPP 4.6 and later (come with GCC package)
- python 2.7 and later
- python-pip
- BeautifulSoup4 (installed using python-pip)
- HCC 0.9 from here

Tools and Misc

- git 1.9 and later
- cmake 2.6 and later (2.6 and 2.8 are tested)
- firewall off
- root privilege or user account in sudo group

Ubuntu Packages

- libc6-dev-i386
- liblapack-dev
- graphicsmagick
- libblas-dev

2.9.10.4 Examples

FFT 1D R2C example:

file: hcfft_1D_R2C.cpp

```
#!c++

#include <iostream>
#include <cstdlib>
#include "hcfft.h"
#include "hc_am.hpp"
#include "hcfftlib.h"

int main(int argc, char* argv[]) {
  int N = argc > 1 ? atoi(argv[1]) : 1024;
  // HCFFT work flow
  hcfftHandle plan;
  hcfftResult status  = hcfftPlan1d(&plan, N, HCFFT_R2C);
  assert(status == HCFFT_SUCCESS);
  int Rsize = N;
  int Csize = (N / 2) + 1;
  hcfftReal* input = (hcfftReal*)calloc(Rsize, sizeof(hcfftReal));
  int seed = 123456789;
  srand(seed);

  // Populate the input
  for(int i = 0; i < Rsize ; i++) {
    input[i] = rand();
  }

  hcfftComplex* output = (hcfftComplex*)calloc(Csize, sizeof(hcfftComplex));

  std::vector<hc::accelerator> accs = hc::accelerator::get_all();
  assert(accs.size() && "Number of Accelerators == 0!");
  hc::accelerator_view accl_view = accs[1].get_default_view();

  hcfftReal* idata = hc::am_alloc(Rsize * sizeof(hcfftReal), accs[1], 0);
  accl_view.copy(input, idata, sizeof(hcfftReal) * Rsize);
  hcfftComplex* odata = hc::am_alloc(Csize * sizeof(hcfftComplex), accs[1], 0);
```

(continues on next page)

(continued from previous page)

```
  accl_view.copy(output,  odata, sizeof(hcfftComplex) * Csize);
  status = hcfftExecR2C(plan, idata, odata);
  assert(status == HCFFT_SUCCESS);
  accl_view.copy(odata, output, sizeof(hcfftComplex) * Csize);
  status =  hcfftDestroy(plan);
  assert(status == HCFFT_SUCCESS);
  free(input);
  free(output);
  hc::am_free(idata);
  hc::am_free(odata);
}
```

- Compiling the example code:

Assuming the library and compiler installation is followed as in installation.

/opt/rocm/hcc/bin/clang++ *`/opt/rocm/hcc/bin/hcc-config –cxxflags –ldflags`* -lhc_am -lhcfft -I../lib/include -L../build/lib/src hcfft_1D_R2C.cpp

2.9.10.5 Tested Environments

This sections enumerates the list of tested combinations of Hardware and system softwares.

Driver versions

- **Boltzmann Early Release Driver + dGPU**
  - Radeon Open Compute Kernel (ROCK) driver : https://github.com/RadeonOpenCompute/ROCK-Kernel-Driver
  - HSA runtime API and runtime for Boltzmann: https://github.com/RadeonOpenCompute/ROCR-Runtime
- Traditional HSA driver + APU (Kaveri)

GPU Cards

- Radeon R9 Nano
- Radeon R9 FuryX
- Radeon R9 Fury
- Kaveri and Carizo APU

Server System

- Supermicro SYS 2028GR-THT 6 R9 NANO
- Supermicro SYS-1028GQ-TRT 4 R9 NANO
- Supermicro SYS-7048GR-TR Tower 4 R9 NANO

2.9.11 Tensile

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 GPU.

Overview for creating a custom TensileLib backend library for your application:

1. Install Tensile (optional), or at least install the PyYAML dependency (mandatory).

2. Create a benchmark config.yaml file.
3. Run the benchmark to produce a library logic.yaml file.
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 or OpenCL.
    - Solution classes which enqueue the kernels.
    - APIs which call the fastest solution for a problem.

```
sudo apt-get install python-yaml
mkdir Tensile
cd Tensile
git clone https://github.com/RadeonOpenCompute/Tensile.git repo
mkdir build
cd build
python ../repo/Tensile/Tensile.py ../repo/Tensile/Configs/sgemm_5760.yaml ./
```

After a while 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 5760
```

2.9.11.1 Benchmark Config

Example Benchmark config.yaml

```
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: False
```

(continues on next page)

(continued from previous page)

```
    - # BenchmarkProblemSizeGroup
      InitialSolutionParameters:
      BenchmarkCommonParameters:
        - ProblemSizes:
          - Range: [ [5760], 0, 0 ]
        - LoopDoWhile: [False]
        - NumLoadsCoalescedA: [-1]
        - NumLoadsCoalescedB: [1]
        - WorkGroupMapping: [1]
      ForkParameters:
         - ThreadTile:
         - [ 8, 8 ]
         - [ 4, 8 ]
         - [ 4, 4 ]
        - WorkGroup:
          - [  8, 16,  1 ]
          - [ 16, 16,  1 ]
        - LoopTail: [False, True]
        - EdgeType: ["None", "Branch", "ShiftPtr"]
        - DepthU: [ 8, 16]
        - VectorWidth: [1, 2, 4]
      BenchmarkForkParameters:
      JoinParameters:
        - MacroTile
      BenchmarkJoinParameters:
      BenchmarkFinalParameters:
        - ProblemSizes:
          - Range: [ [5760], 0, 0 ]

LibraryLogic:

LibraryClient:
```

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.

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.

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 or False.

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.

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.

2.9.11.2 Benchmark Protocol

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.

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.

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
   - # Benchmark Size-Group
    InitialSolutionParameters:
      - WorkGroup: [ [ 16, 16, 1 ] ]
      - NumLoadsCoalescedA: [ 1 ]
      - NumLoadsCoalescedB: [ 1 ]
      - ThreadTile: [ [ 4, 4 ] ]

    BenchmarkCommonParameters:
      - ProblemSizes:
        - Range: [ [512], [512], [512] ]
      - EdgeType: ["Branch", "ShiftPtr"]
        PrefetchGlobalRead: [False, True]
```

(continues on next page)

(continued from previous page)

```
ForkParameters:
  - WorkGroup: [ [8, 32, 1], [16, 16, 1], [32, 8, 1] ]
    ThreadTile: [ [2, 8], [4, 4], [8, 2] ]

BenchmarkForkParameters:
  - ProblemSizes:
    - Exact: [ 2880, 2880, 2880 ]
  - NumLoadsCoalescedA: [ 1, 2, 4, 8 ]
  - NumLoadsCoalescedB: [ 1, 2, 4, 8 ]

JoinParameters:
  - MacroTile

BenchmarkJoinParameters:
  - LoopUnroll: [8, 16]

BenchmarkFinalParameters:
  - ProblemSizes:
    - Range: [ [16, 128], [16, 128], [256] ]
```

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.

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

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.

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.

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.

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.

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.

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.

2.9.11.3 Dependencies

CMake

- CMake 2.8

Python

- Python 2.7
- PyYAML (Can be installed via apt, apt-get, yum, pip...; module is typically named python-yaml, pyyaml or PyYAML.)

Compilers

- **For Tensile_BACKEND = OpenCL1.2**
  - 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
- **For Tensile_BACKEND = HIP**
  - ROCM 2.0

Installation

Tensile can be installed via:

1. Install directly from repo using pip:

```
pip install git+https://github.com/RadeonOpenCompute/Tensile.git@develop
tensile config.yaml benchmark_path
```

2. Download repo and install manually:

```
git clone https://github.com/RadeonOpenCompute/Tensile.git
cd Tensile
sudo python setup.py install
tensile config.yaml benchmark_path
```

3. Download repo and don't install; install PyYAML dependency manually and call python scripts manually:

```
git clone https://github.com/RadeonOpenCompute/Tensile.git
python Tensile/Tensile/Tensile.py config.yaml benchmark_path
```

2.9.11.4 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.

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

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.

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.

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.

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.

2.9.11.5 Languages

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.

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.

Device Languages

The device languages Tensile supports for the gpu kernels is

- OpenCL 1.2
- HIP
- **Assembly**
  - gfx803
  - gfx900

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.

2.9.11.6 Problem Nomenclature

Example Problems

- C[i,j] = Sum[k] A[i,k] * B[k,j] (GEMM; 2 free indices and 1 summation index)
- C[i,j,k] = Sum[l] A[i,l,k] * B[l,j,k] (batched-GEMM; 2 free indices, 1 batched index and 1 summation index)
- 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

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).

Limitations

Problem supported by Tensile must meet the following conditions:

There must be at least one pair of free indices.

2.9.11.7 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 )
```

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.

2.9.12 rocALUTION

2.9.12.1 Introduction

2.9.12.2 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

2.9.12.3 Building and Installing

2.9.12.4 Installing from AMD ROCm repositories

TODO, not yet available

2.9.12.5 Building rocALUTION from Open-Source repository

2.9.12.6 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.

2.9.12.7 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.

2.9.12.8 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.

2.9.12.9 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
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>
```

2.9.12.10 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

2.9.12.11 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

./cg gr_30_30.mtx
```

For more information regarding rocALUTION library and corresponding API documentation, refer rocALUTION

2.9.13 rocSPARSE

2.9.13.1 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 code is open and hosted here: https://github.com/ROCmSoftwarePlatform/rocSPARSE

2.9.13.2 Device and Stream Management

hipSetDevice() and *hipGetDevice()* are HIP device management APIs. They are NOT part of the rocSPARSE API.

2.9.13.3 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.

2.9.13.4 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.

2.9.13.5 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.

2.9.13.6 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.

2.9.13.7 Building and Installing

2.9.13.8 Installing from AMD ROCm repositories

rocSPARSE can be installed from AMD ROCm repositories by

```
sudo apt install rocsparse
```

2.9.13.9 Building rocSPARSE from Open-Source repository

2.9.13.10 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.

2.9.13.11 Using *install.sh* to build dependencies + library

The following table lists common uses of *install.sh* to build dependencies + library.

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 used once. For subsequent invocations of *install.sh* it is not necessary to rebuild the dependencies.
./install.sh	Build library in your local directory. It is assumed dependencies are available.
./install.sh -i	Build library, then build and install rocSPARSE package in */opt/rocm/rocsparse*. You will be prompted for sudo access. This will install for all users.

2.9.13.12 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 used once. For subsequent invocations of *install.sh* it is not necessary to rebuild the 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 */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*. You will be prompted for sudo access. This will install for all users.

2.9.13.13 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
```

(continues on next page)

(continued from previous page)

```
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
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 ..

# 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

# Compile rocSPARSE library
make -j$(nproc)

# Install rocSPARSE to /opt/rocm
sudo make install
```

2.9.13.14 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

Regarding more information about rocSPARSE and it's functions, corresponding API's, Please refer rocsparse

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
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 \
    -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 \
```

(continues on next page)

(continued from previous page)

```
    -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 finalizer extension for ROCR runtime
- 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

- The image extension is currently not supported for discrete GPUs. An image extension library is not provided in the binary package. The standard hsa_ext_image.h extension include file is provided for reference.
- Each HSA process creates and internal DMA queue, but there is a system-wide limit of four DMA queues. The fifth simultaneous HSA process will fail hsa_init() with HSA_STATUS_ERROR_OUT_OF_RESOURCES. To run an unlimited number of simultaneous HSA processes, set the environment variable HSA_ENABLE_SDMA=0.

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.

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] [-i] [-t] [-c] [-g] [-f] [-p] [-P] [-o] [-l] [-s] [-a] [-r] [–setsclk LEVEL [LEVEL . . .]] [–setmclk LEVEL [LEVEL . . .]] [–setfan LEVEL] [–setperflevel LEVEL] [–setoverdrive %] [–setprofile # # # # #] [–resetprofile] [–load FILE | –save FILE] [–autorespond RESPONSE]

AMD ROCm System Management Interface

optional arguments:

-h, –help	show this help message and exit
–load FILE	Load Clock, Fan, Performance and Profile settings from FILE
–save FILE	Save Clock, Fan, Performance and Profile settings to FILE

-d 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
-b, –showbw	Show estimated PCIe use
-S, –showclkvolt	Show supported GPU and Memory Clocks and Voltages
-a, –showallinfo	Show all SMI-supported values values
-r, –resetclocks	Reset clocks to default values
–setsclk LEVEL [LEVEL . . .]	Set GPU Clock Frequency Level Mask
–setmclk LEVEL [LEVEL . . .]	Set GPU Memory Clock Frequency Mask
–setpclk 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 manual|high 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
–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

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 #: 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

-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

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

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

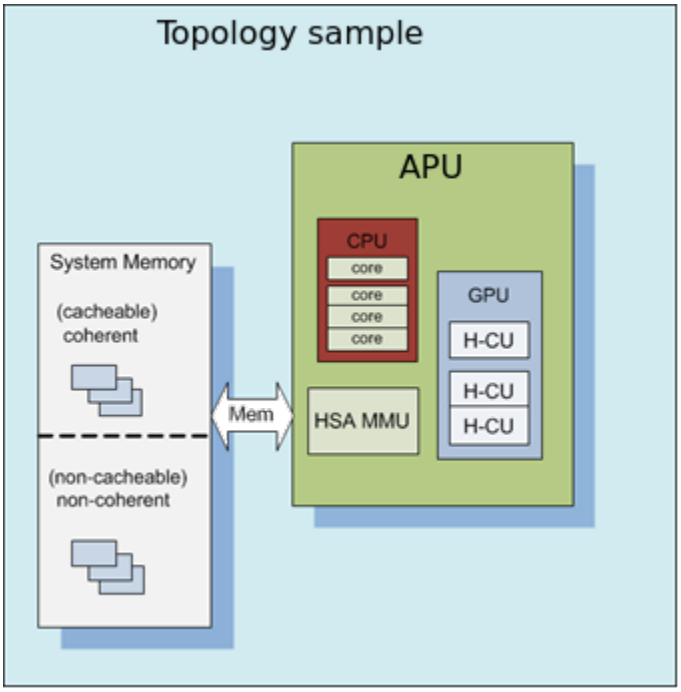
The IO links provides HSA agent interconnect information with latency (cost) between agents. This is useful for peer-to-peer transfers.

2.11.3.6 How to use topology information

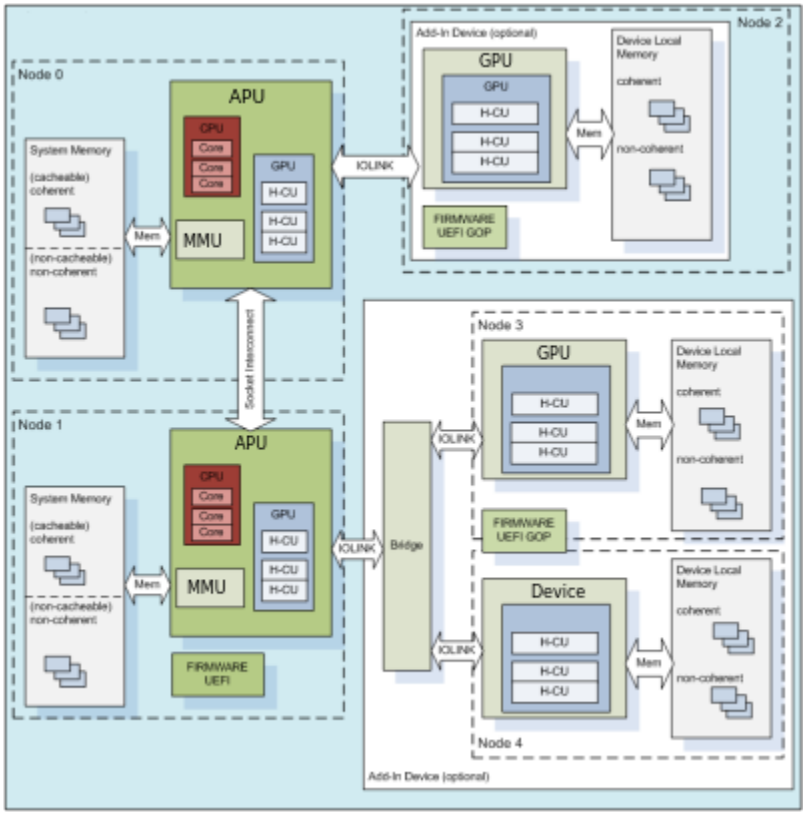
The information provided in sysfs should not be directly used by application software. Application software should always use Thunk library API (libhsakmt) to access topology information. Please refer to Thunk API for more information.

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/$dev/device)
        if [ -e /sys/bus/pci/devices/$dev/driver ]; then
                echo $dev > /sys/bus/pci/devices/$dev/driver/unbind
```

(continues on next page)

(continued from previous page)

```
        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 to https://wiki.archlinux.org/index.php/PCI_passthrough_via_OVMF 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

- 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 --rm --device="/dev/kfd" 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 (packages.amd.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

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 ROCnRDMA

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
                                           * which this page table described
                                           */

        uint64_t        size;           /**< Specify total size of
                                           * allocation
                                           */

       struct pid       *pid;           /**< Specify process pid to which
                                           * virtual address belongs
```

(continues on next page)

(continued from previous page)

```
                                        */

       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
 * \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

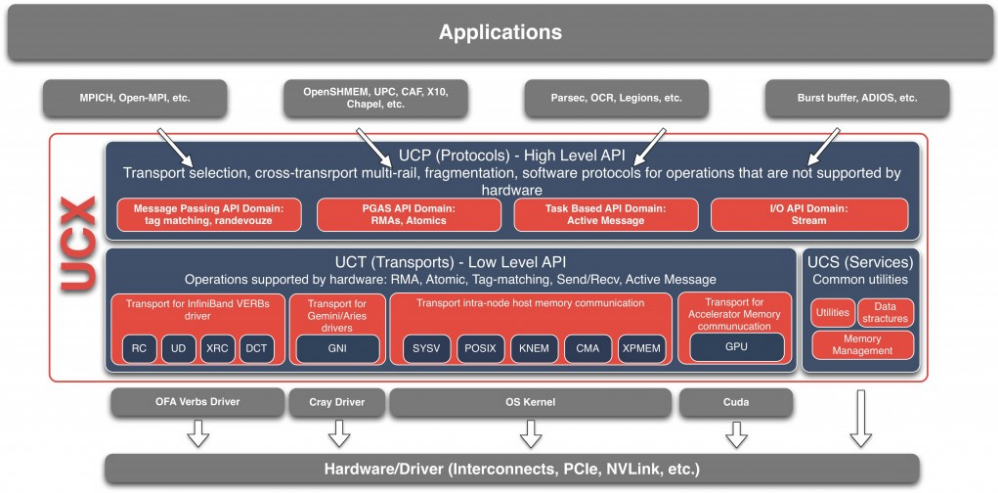
```
/**
 * This function makes the pages underlying a range of GPU virtual memory
 * accessible for DMA operations from another PCIe device
 *
 * \param   address       - The start address in the Unified Virtual Address
 *                          space in the specified process
 * \param   length        - 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



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 ar**
  - "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 (v1.2)

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 ROCm Tensorflow v1.12 Release

We are excited to announce the release of ROCm enabled TensorFlow v1.12 for AMD GPUs.

2.14.2 Tensorflow Installation

First, you'll need to install the open-source ROCm 2.0 stack. Details can be found here: https://rocm.github.io/ROCmInstall.html

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 v1.12 is installed!

2.14.3 Tensorflow More Resources

Tensorflow docker images are also publicly available, more details can be found here: https://hub.docker.com/r/rocm/tensorflow/

Please connect with us for any questions, our official github repository is here: https://github.com/ROCmSoftwarePlatform/tensorflow-upstream

2.14.4 ROCm MIOpen v1.6 Release

Announcing our new Foundation for Deep Learning acceleration MIOpen 1.6 which introduces support for Convolution Neural Network (CNN) acceleration — built to run on top of the ROCm software stack!

This release includes the following:

- Training in fp16 (half precision) including mixed-precision is now fully supported
- Batch Normalization in fp16 (half precision) including mixed-precision are now available
- Performance improvements for 3x3 and 1x1 single-precision convolutions
- Layer fusions for BatchNorm+Activation are now available
- Layer fusions with convolutions now support varying strides and padding configurations
- Support for OpenCL and HIP enabled frameworks API's
- MIOpen Driver enables the testing of forward/backward calls of any particular layer in MIOpen.
- Binary Package support for Ubuntu 16.04 and Fedora 24
- Source code at https://github.com/ROCmSoftwarePlatform/MIOpen
- **Documentation**
  - MIOpen
  - MIOpenGemm

2.14.5 Porting from cuDNN to MIOpen

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

2.14.6 The ROCm 2.0 has prebuilt packages for MIOpen

Install the ROCm MIOpen implementation (assuming you already have the 'rocm' and 'rocm-opencl-dev" package installed):

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

2.14.7 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 2.0 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
```

2.14.8 Recommended:Install using published PyTorch ROCm docker image:

2. Obtain docker image:

```
docker pull rocm/pytorch:rocm2.0
```

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:rocm2.0
```

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'
```

2.14.9 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'
```

2.14.10 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'
```

2.14.11 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.

2.14.12 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-2.0s 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.

2.14.13 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)

- Launch the docker container

```
docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video␣
↪rocm/caffe2:238-2.0
```

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.

2.14.14 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
sgit submodule update --init --recursive
```

2. Launch the docker container

```
docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video -
↪v $PWD:/pytorch rocm/caffe2:unbuilt-238-2.0
```

3. Build Caffe2 from source

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
```

2.14.15 Test the Caffe2 Installation

To validate Caffe2 installation, for both options, run

1. Test Command

```
cd build_caffe2 && python -c 'from caffe2.python import core' 2>/dev/null && echo
↪"Success" || echo "Failure"
```

If the test fails, make sure the following environment variables are set. LD_LIBRARY_PATH=/pytorch/build_caffe2/lib

```
PYTHONPATH=/pytorch/build_caffe2
```

2. Running unit tests in Caffe2

```
.jenkins/caffe2/test.sh
```

2.14.16 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	https://github.com/ROCmSoftwarePlatform/caffe2
Torch	HIP	Upstreaming	Development inProgress	https://github.com/ROCmSoftwarePlatform/cutorch_hip
HIPnn	Upstreaming	Development		https://github.com/ROCmSoftwarePlatform/cunn_hip
PyTorch	Development	Development		
MxNet	Development	Development		https://github.com/ROCmSoftwarePlatform/mxnet
CNTK	Development	Development		

2.14.17 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
- 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

R