ROCM Software Stack
IWOMP 2020 Vendor Presentation
Greg Rodgers
Derek Bouius

Sept 2020
AMD PUBLIC
Cautionary Statement

This presentation contains forward-looking statements concerning Advanced Micro Devices, Inc. (AMD) including, but not limited to the features, functionality, availability, timing, expectations and expected benefits of AMD’s products, which are made pursuant to the Safe Harbor provisions of the Private Securities Litigation Reform Act of 1995. Forward-looking statements are commonly identified by words such as "would," "may," "expects," "believes," "plans," "intends," "projects" and other terms with similar meaning. Investors are cautioned that the forward-looking statements in this presentation are based on current beliefs, assumptions and expectations, speak only as of the date of this presentation and involve risks and uncertainties that could cause actual results to differ materially from current expectations. Such statements are subject to certain known and unknown risks and uncertainties, many of which are difficult to predict and generally beyond AMD's control, that could cause actual results and other future events to differ materially from those expressed in, or implied or projected by, the forward-looking information and statements. Investors are urged to review in detail the risks and uncertainties in AMD's Securities and Exchange Commission filings, including but not limited to AMD’s Quarterly Report on Form 10-Q for the quarter ended June 27, 2020.
Differentiated Strategy
Optimal Efficiency Through Domain-Specific Optimizations

RDNA
Graphics-Optimized
GPU Architecture

CDNA
Compute-Optimized
GPU Architecture

General Purpose
GPU Architecture (GPGPU)

Efficiency

Real-Time Rendering (Frames/Sec)

High-Performance Compute (Flops/Sec)
AMD EPYC™ CPUs & Radeon Instinct™ GPUs
Leading The Exascale Era

>2 ExaFLOPS Expected

Expected to be More Powerful than Today’s 200 Fastest Supercomputers Combined

AMD Shipments in 2022

Source: https://www.top500.org/lists/top500/2020/06/
AMD CDNA Architecture

Compute DNA for the Data Center

<table>
<thead>
<tr>
<th>Performance</th>
<th>Efficiency</th>
<th>Features</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerate ML/HPC with Compute/Tensor OPS</td>
<td>Help Reduce TCO with High Perf-per-Watt</td>
<td>Enhance Enterprise RAS, Security and Virtualization</td>
<td>Scale Performance with AMD Infinity Architecture</td>
</tr>
</tbody>
</table>
What is ROCm™?

An Open Software Platform for GPU-accelerated Computing

<table>
<thead>
<tr>
<th>Frameworks and Applications</th>
<th>Libraries</th>
<th>Programmer and system tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow, PyTorch, Caffe2</td>
<td>MIOpen, roc* libraries</td>
<td>-debug -profile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming models</th>
<th>Intermediate runtimes/compilers</th>
<th>Runtimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIP, C/C++, Python</td>
<td>OpenMP, HIP, OpenCL</td>
<td>ROCm</td>
</tr>
</tbody>
</table>
Data Center Software Evolution

Steady Progress and Growing Ecosystem Support

2018: AMD ROCm™ 2.0 Platform
Building the Foundation

2019: AMD ROCm™ 3.0 Platform
Focused on Machine Learning

2020 Plan: AMD ROCm™ 4.0 Platform
Complete Exascale Solution for ML/HPC
Machine Intelligence

Natural Language Processing

Image Recognition

Recommendation Engines

Industrial Automation

Revolutionizing Applications in Every Field

Exponentially Growing Demands for Performance

AMD Champions Open Source Solutions
AMD CPU + GPU + SW Advantages
Driving High-performance Computing Leadership

- Fully Integrated CPU and GPU Systems and Unified Tools
- Infinity Architecture for Bandwidth and Coherency
- Open Source Software Optimized for Performance

NAMD 2.13 Benchmark

<table>
<thead>
<tr>
<th>NS / DAY</th>
<th>2x Xeon® Platinum 8280</th>
<th>2x Xeon® Platinum 8280 + 8x Tesla® V100</th>
<th>2x AMD EPYC™ 7742 CPU + 8x AMD Radeon Instinct™ MI50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Up to 26%</td>
</tr>
</tbody>
</table>

Testing Conducted by AMD performance lab as of 11-10-2019 using NAMD 2.13, STMV 1M Atom benchmark. Best-in-class based on industry-standard pin-based (LGA) X86 processors. Results may vary. (RIV-20)
HIP: Multi-Platform Capability for TCO Optimization

Easy to Deploy Porting Capability

CUDA-based application

“HIPify”

Virtually Automatic Conversion

Portable HIP C++

Developer maintains HIP port

Resulting C++ code runs on NVIDIA or AMD GPUs
Fast-Growing ROCm™ Ecosystem

- Containers
- Sylabs Singularity
- Data Center Workload Manager
- Container Orchestration

- Performance Profiling & System Tracer via PAPI
- Eclipse C/C++ Development Tooling Based on ROC-GDB
- Upstream ML Frameworks
- Exascale Tools, Programming Models and Applications
Docker®

- Set permissions and add user to docker group
  - groups
  - sudo usermod -a -G docker $LOGNAME

- ROCm™ Docker Hub
  - https://hub.docker.com/u/rocm/

- Run Docker Image
  - docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video --cap-add=SYS_PTRACE --security-opt seccomp=unconfined -v /home/user:/home/user -v /dev:/dev -v /ubuntu:ubuntu bash

- Show running image
  - docker image ls

- Save container to your own image
  - Run docker commit on another terminal window
  - docker commit <container id> <my_docker_image>
Machine Learning Models
Deployable Today with Continuous Optimizations

**Image Classification**
- ResNet50/101
- ResNet152
- Inception3/4
- VGG16/19
- ShuffleNet
- MobileNet
- DenseNet
- AlexNet
- SqueezeNet
- GoogleNet
- ResNext101

**Object Detection**
- Faster-RCNN-ResNet50
- Mask-RCNN-ResNet50
- SSD-Resnet50

**Neural Machine Translation**
- GNMT: LSTMs
- Translate: LSTMs
- BERT: Transformer
- GPT-2: Transformer

**Reinforcement Learning**
- Atari
- Cart_Pole
- VizDoom

**Recommender Systems**
- DLRM

**Generative Models**
- DCGAN
- Fast Neural Style Transfer
AMD GPU

Compilers:

C/C++

**HIP (hip-clang)**
- HIP (Heterogeneous Interface for Portability) is an interface that provides similar functionality to CUDA API
- Compiles HIP code and emits AMDGCN into binary
- `hipcc -> hip-clang -> amdgcn`
- Compiles to NVIDIA GPU with NVCC & its tool chain
- All the x86 pieces are dealt with in the same way

**AOMP (AMD OpenMP Compiler)**
- LLVM
- Compiles C/C++ code with OpenMP “target” pragmas
- Sources feeds into ROCm compiler for future unified LLVM compiler

**OpenCL™**
- LLVM
- Khronos Industry Standard accelerator language

*The GCN ISA is free and open!*
https://developer.amd.com/resources/developer-guides-manuals/
AMD GPU

Compilers:

Fortran

AOMP
- LLVM clang driver for flang
- Limited flang support for OpenMP 4.5+ target offload
- Will move to F18 in the future
- Feeds into ROCm compiler

hipfort
- New package for HIP and ROCm library APIs in FORTRAN
- Offload kernels to GPU using Fortran 2003 C-binding
- Generated FORTRAN interface and mod files
- Designed for multiple compilers, default is gfortran

Frontier
- See Frontier spec sheet for what is expected to be supported:
Unified CPU & GPU Debugger
Easily Integrated with Industry Standard Tools

- Uniﬁed CPU & GPU Debugger
- Easily Integrated with Industry Standard Tools
- ROC-gdb (gdb with GPU support)
- ROC-dbgapi (GPU Low Level Debug API Library)
- Linux Kernel (ptrace)
- GPU Kernel
- Driver

- Released Q2-2020

Tools:
- Eclipse
- THEIA
- arm DDT
- CCDB
- TotalView
- ROC-gdb
- ROC-dbgapi
- RogueWave Software

- A Hewlett Packard Enterprise company
- AMD Public Use

IWOMP 2020 – Radeon Instinct and ROCm – Sept 2020
ROCgdb

- ROCgdb is the ROCm source-level debugger for Linux
- ROCgdb is based on GDB, the GNU source-level debugger
  - https://github.com/ROCm-Developer-Tools/ROCgdb
- Compile executable using hipcc with "--ggdb"
- ROCgdb location:
  - /opt/rocm/bin/rocgdb
- To debug an executable
  - rocgdb $EXE
- To attach to a running process
  - rocgdb -p <pid>
ROC-Profiler / Tracer
Easily Integrated with Industry Standard Tools

ROC-Profiler Lib

ROC-Tracer Lib

roprof CLI

Easily Integrated with Industry Standard Tools

Memcopy time interval
API call time interval
GPU kernel time interval
Kernel parameters
Kernel counters
Kernel timestamps

Released Q4-2019
rocprof

- rocprof is the AMD GPU profiler library
- It profiles with perf-counters and derived metrics
- To run rocprof to generate a kernel profile (text)
  - `rocprof --obj-tracking on --stats $EXE`
  - The default `results.stats.csv` will be generated
  - Comma-separated list of kernel activities

- Run rocprof to generate a trace file
  - `rocprof --obj-tracking on --sys-trace $EXE`
  - Start Google Chrome
  - Type `chrome://tracing`
  - Load (or Drag and Drop) the JSON file to view
- [https://github.com/ROCm-Developer-Tools/rocprofiler/](https://github.com/ROCm-Developer-Tools/rocprofiler/)
**ROCm™ Installation v3.8.0(latest) – Ubuntu® 18.04**

1. **Ensure that the system is up to date**
   ```bash
   sudo apt update
   sudo apt dist-upgrade
   sudo apt install libnuma-dev
   sudo reboot
   ```

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

3. **Install the ROCm meta-package & rocm-dkms meta-package**
   ```bash
   sudo apt update
   sudo apt install --y rocm-dkms miopen-hip rocblas
   ```
Set permissions and add user to video group

```
groups  # identify the groups member
sudo usermod -a -G video $LOGNAME
```

Restart the system

Test the basic ROCm installation

```
/opt/rocm/bin/rocminfo
dpkg -l | grep rocm  #Report installed ROCm versions
```
Visit AMD.com/ROCm

Link to more training information:

https://community.amd.com/community/radeon-instinct-accelerators/blog/
Thank You!
Disclaimers and Attributions

The information contained herein is for informational purposes only, and is subject to change without notice. Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change. “Polaris”, “Vega”, “Radeon Vega”, “Navi”, “Zen” and “Naples” are codenames for AMD architectures, and are not product names.

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.

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. GD-18

©2020 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, EPYC, ROCm, RDNA, 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.
What is ROCm™?

An *Open* Software Platform for GPU-accelerated Computing

---

<table>
<thead>
<tr>
<th>Frameworks and Applications</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow, PyTorch, Caffe2</td>
<td>MIOpen, roc* libraries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming models</th>
<th>Intermediate runtimes/compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIP, C/C++, Python</td>
<td>OpenMP, HIP, OpenCL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Runtimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROCm</td>
</tr>
</tbody>
</table>
ROCm™ Screen Info

Agent 2

Name: gfx906
Marketing Name: Vega 20
Vendor Name: AMD

Features:
- KERNEL DISPATCH
- BASE PROFILE
- Float Round Mode: NEAR

Max Queue Number: 128 (0x80)
Queue Min Size: 4096 (0x1000)
Queue Max Size: 131072 (0x200000)

Node: 1
Device Type: GPU
Cache Info:
- All: 16760832 (0xffff0000) KB

Chip ID: 26817 (0x6661)
Cache Line Size: 64 (0x40)
Max Clock Freq. (MHz): 1801

ISA Info:
- ISA 1:
  - Name: amdgcn-amd-amdhsa-gfx906
  - Machine Models: HSA_MACHINE_MODEL_LARGE
  - Profiles: HSA_PROFILE5_BASE
  - Default Rounding Mode: NEAR
  - Default Alignment: NEAR
  - Fast P16: TRUE
  - Workgroup Max Size: 1024 (0x400)
  - Grid Max Size: 4294967295 (0xffffffff)
  - Max Barriers/Workgrp: 32

Max Barriers/Workgrp: 32
Pool Info:
- GLOBAL; FLAGS: COARSE GRAINED
- Size: 16760832 (0xffff0000) KB
- Allocatable: TRUE
- Alloc Granule: 4 KB
- Alloc Alignment: 4 KB
- Accessible by all: FALSE

Pool 2
- Segment: cgroup
- Size: 64 (0x40) KB
- Allocatable: FALSE
- Alloc Granule: 0 KB
- Alloc Alignment: 0 KB
- Accessible by all: FALSE
ROCM™ Version Details

Note: demo purpose only, please check the release notes for the latest rocm lib versions
Basic ROCm™ Tools

1. rocm-smi

2. rocm-bandwidth-test (https://github.com/RadeonOpenCompute/rocm_bandwidth_test)
   
   
   
   ./rocm-bandwidth-test -b 2,0  # gpu0↔cpu0 bidirectional
   ./rocm-bandwidth-test -b 2,3  # gpu0↔gpu1 bidirectional

3. rocblas-bench (DGEMM, SGEMM)

   
   
   
   ./rocblas-bench -f gemm -r d -m 8640 -n 8640 -k 8640 --transposeB T --initialization trig_float -i 200 --device 0 &

4. rvs (rocm-validation-suite)  # Cluster management tool

   
   
   
   sudo ./rvs -c conf/Artus_dgemm_gst.conf -d 3 -l RVS_dgemm_result.log
ROCm™ SMI Screen Info

```bash
isvperf@isvperf-SYS-4029GP-TRT2:~$ rocm-smi

------------------------ROCm System Management Interface------------------------

<table>
<thead>
<tr>
<th>GPU</th>
<th>Temp</th>
<th>AvgPwr</th>
<th>SCLK</th>
<th>MCLK</th>
<th>Fan</th>
<th>Perf</th>
<th>PwrCap</th>
<th>VRAM %</th>
<th>GPU %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>65.0c</td>
<td>210.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>44%</td>
<td>99%</td>
</tr>
<tr>
<td>2</td>
<td>28.0c</td>
<td>26.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>30.0c</td>
<td>29.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>29.0c</td>
<td>29.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>5</td>
<td>31.0c</td>
<td>26.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>6</td>
<td>29.0c</td>
<td>30.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>7</td>
<td>29.0c</td>
<td>24.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>1.96%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>29.0c</td>
<td>28.0W</td>
<td>1725Mhz</td>
<td>1000Mhz</td>
<td>0.0%</td>
<td>high</td>
<td>225.0W</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

End of ROCm SMI Log
```
ROCm™ Bandwidth Test – Installation

1. **# Tools**
   
   ```bash
   sudo bash
   ```

2. **# Add rocm bandwidth-tests**
   
   ```bash
   # apt-get -y update && sudo apt-get install -y libpci3 libpci-dev doxygen unzip cmake git
   cd /opt/rocm
   git clone https://github.com/RadeonOpenCompute/rocm_bandwidth_test.git
   cd rocm_bandwidth_test;mkdir ./build;cmake ./ -B./build;make -C ./build
   ```

3. **# Install rocm-bandwithh-test package from ROCm repo**
   
   ```bash
   # apt install -y rocm-bandwidth-test
   # exit
   ```

4. **# Run RBT**
   
   ```bash
   $rocm-bandwidth-test
   ```
ROCm™ Bandwidth Test – Platform

Note: This is the reference platform with following config:
- Dual Socket AMD EPYC 7742,
- 8x MI50
- 512 GB DDR4 3200
- 960GB NVMe drive
- 256 GB HBM2 @ 8 TB/s

Note: with P2P connected with xGMI, the achievable bandwidth could be updating the light blue arrows from 26 to 32 for unidirectional and 52 to 59 for bidirectional.
### ROCm™ Bandwidth – Bidirectional (Target-pcie-gen4)

<table>
<thead>
<tr>
<th>D/D</th>
<th>CPU 0</th>
<th>CPU 1</th>
<th>GPU 0</th>
<th>GPU 1</th>
<th>GPU 2</th>
<th>GPU 3</th>
<th>GPU 4</th>
<th>GPU 5</th>
<th>GPU 6</th>
<th>GPU 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 0</td>
<td></td>
<td></td>
<td>49.3</td>
<td>48.9</td>
<td>48.7</td>
<td>49.3</td>
<td>35.2</td>
<td>32.2</td>
<td>32.2</td>
<td>32.1</td>
</tr>
<tr>
<td>CPU 1</td>
<td></td>
<td></td>
<td>32.2</td>
<td>34.1</td>
<td>34.2</td>
<td>32.2</td>
<td>48.7</td>
<td>49.3</td>
<td>48.4</td>
<td>49.4</td>
</tr>
<tr>
<td>GPU 0</td>
<td>49.3</td>
<td>32.2</td>
<td></td>
<td></td>
<td>52.4</td>
<td>52.1</td>
<td>52.7</td>
<td>45.7</td>
<td>43.7</td>
<td>43.7</td>
</tr>
<tr>
<td>GPU 1</td>
<td>48.9</td>
<td>34.1</td>
<td>52.4</td>
<td></td>
<td></td>
<td>52.4</td>
<td>52.1</td>
<td>52.7</td>
<td>45.8</td>
<td>45.4</td>
</tr>
<tr>
<td>GPU 2</td>
<td>48.7</td>
<td>34.2</td>
<td>52.1</td>
<td>52.1</td>
<td></td>
<td></td>
<td>52.1</td>
<td>45.8</td>
<td>45.6</td>
<td>45.6</td>
</tr>
<tr>
<td>GPU 3</td>
<td>49.3</td>
<td>32.2</td>
<td>52.7</td>
<td>52.4</td>
<td>52.1</td>
<td></td>
<td>45.8</td>
<td>44.6</td>
<td>44.6</td>
<td>43.7</td>
</tr>
<tr>
<td>GPU 4</td>
<td>35.2</td>
<td>48.7</td>
<td>45.7</td>
<td>45.8</td>
<td>45.8</td>
<td>45.8</td>
<td></td>
<td>52.3</td>
<td>51.4</td>
<td>52.3</td>
</tr>
<tr>
<td>GPU 5</td>
<td>32.2</td>
<td>49.3</td>
<td>43.7</td>
<td>45.4</td>
<td>45.6</td>
<td>44.6</td>
<td>52.3</td>
<td></td>
<td>51.4</td>
<td>52.7</td>
</tr>
<tr>
<td>GPU 6</td>
<td>32.2</td>
<td>48.4</td>
<td>43.7</td>
<td>45.3</td>
<td>45.6</td>
<td>44.6</td>
<td>51.4</td>
<td>51.4</td>
<td></td>
<td>51.4</td>
</tr>
<tr>
<td>GPU 7</td>
<td>32.1</td>
<td>49.4</td>
<td>44.4</td>
<td>45.0</td>
<td>45.0</td>
<td>43.7</td>
<td>52.3</td>
<td>52.7</td>
<td>51.4</td>
<td></td>
</tr>
</tbody>
</table>

Note: with GPU (MI50) to GPU connected with xGMI, the achievable bandwidth could be updating the above table’s light blue area from ~52 to ~59 for bidirectional.
rocBLAS – Installation and Build

1. # Tools
   $sudo bash

2. # Add rocBLAS git and build
   
   cd /opt/rocm
   git clone https://github.com/ROCmSoftwarePlatform/rocBLAS.git
   cd rocBLAS;
   mkdir ./build; cmake ./ -B./build; make -C ./build;./install.sh -idc

3. # Install rocBLAS
   
   apt install -y rocblas
   exit

4. # Run Rocblas-Bench
   
   $ /opt/rocm/rocBLAS/build/release/clients/staging/ rocblas-bench -f gemm -r d -m 8640 -n 8640 -k 8640 --transposeB T --initialization trig_float -i 200 --device 0 &
rocBLAS-Bench DGEMM Results

Note: the above measurement is collected with a GigaByte Z52 system with 2x2nd Gen EPYC + 8xMI50 with ROCm3.3
ROCm™ Validation Suite (RVS) Introduction

The ROCm Validation Suite (RVS) is a system administration and cluster management tool for detecting and troubleshooting common problems affecting AMD GPU(s) running in a high-performance computing environment. RVS is enabled using the ROCm software stack on a compatible platform.

The RVS focuses on software and system configuration issues, diagnostics, topological concerns, and relative systems performance.

1. Deployment and Software Issues
2. Hardware Issues and Diagnostics
3. Integration Issues
4. System Stress Checks
5. Troubleshooting
6. Integration into Cluster Scheduler and Cluster Management Applications
7. Help Reduce Downtime and Failed GPU jobs
RVS Installation

- Linux® System Support Only
- RVS is Open Source Code

- Ex: ubuntu 18.04 command line
  
  ```
  # git clone https://github.com/ROCm-Developer-Tools/ROCmValidationSuite.git
  ```

- Detail configure and build RVS - reference below website
  
  https://github.com/ROCm-Developer-Tools/ROCmValidationSuite
TensorFlow installation: TF-ROCm2.2.0-beta1

1. Install other relevant ROCm packages
   
   sudo apt update
   sudo apt install rocm-libs miopen-hip rccl

2. Install TensorFlow (via the Python Package Index)
   
   sudo apt install wget python3-pip
   pip3 install --user tensorflow-rocm


Note: some prerequisites libraries need to be installed first such as
$ sudo apt-get install python3-pip
$ sudo pip3 install -U pip
Basic Tensorflow Benchmark: CNN-ResNet50

1. Clone from Github
   `git clone https://github.com/tensorflow/benchmarks.git`

2. Pull the Docker® Container
   (install docker if necessary following steps @ https://phoenixnap.com/kb/how-to-install-docker-on-ubuntu-18-04)
   `docker pull rocm/tensorflow:rocm3.3-tf1.15-dev`

3. Run the Container in Detached mode (will generate ID)
   `sudo docker run -d -it --network=host -v $HOME:/data --security-opt seccomp=unconfined -v $HOME/dockerx:/dockerx -v /data/imagenet-inception:/imagenet --privileged --device=/dev/kfd --device=/dev/dri --group-add video --cap-add=SYS_PTRACE rocm/tensorflow:rocm3.3-tf1.15-dev`

4. Attach to the container with the output ID
   `docker attach generate_ID`
Tensorflow Benchmark: CNN-ResNet50 (Cont.)

5. Navigate to the Benchmarks
   
   ```bash
   cd /data/benchmarks/scripts/tf_cnn_benchmarks
   ```

6. Run ResNet50 with synthetic data w/o distortions with 1xGPU
   
   ```bash
   python3 tf_cnn_benchmarks.py --model=resnet50 --batch_size=128 --print_training_accuracy=True --variable_update=parameter_server --local_parameter_device=gpu --num_gpus=1
   ```

**model:** Model to use, e.g. resnet50, inception3, vgg16, and alexnet  
**num_gpus:** Number of GPUs to use  
**data_dir:** Path to data to process. If not set, synthetic data is used  
**batch_size:** Batch size for each GPU  
**variable_update:** The method for managing variables: parameter_server, replicated, distributed_replicated, independent  
**local_parameter_device:** Device to use as parameter server: cpu or gpu
Tensorflow CNN-ResNet50 Screen Capture
PyTorch Installation – Docker® Image

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. Obtain Docker image
   
   ```
   docker pull rocm/pytorch:rocm3.0_ubuntu16.04_py3.6_pytorch
   ```

3. Clone PyTorch repository on the host
   
   ```
   cd ~
   git clone https://github.com/pytorch/pytorch.git
   cd pytorch
   git submodule init
   git submodule update
   ```
PyTorch Installation – Build

4. Start a docker container using the downloaded image
   
   ```bash
   sudo docker run -it -v $HOME:/data --privileged --rm --device=/dev/kfd --device=/dev/dri --group-add video rocm/pytorch:rocm3.0Ubuntu16.04_py3.6_pytorch
   ```

5. Build PyTorch
   
   ```bash
   cd /data/pytorch
   .jenkins/pytorch/build.sh
   ```

6. Confirm working installation
   
   ```bash
   PYTORCH_TEST_WITH_ROCM=1 python3.6 test/run_test.py --verbose
   ```

7. Install Torchvision & Commit container to preserve Pytorch install
   
   ```bash
   pip install torchvision
   sudo docker commit <container_id> -m 'pytorch installed'
   ```
Basic PyTorch Benchmark – ResNet50

1. Pull the Docker® Image
   
   ```
   docker pull rocm/pytorch:rocm3.3_ubuntu16.04_py3.6_pytorch
   ```

2. Run the Docker Container
   
   ```
   alias ptdrun=’sudo docker run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add video --cap-add=SYS_PTRACE --security-opt seccomp=unconfined -v $HOME/dockerx:/dockerx --shm-size=64G’
   ptdrun rocm/pytorch:rocm3.3_ubuntu16.04_py3.6_pytorch
   ```

3. In docker container, install dependencies and download py script
   
   ```
   pip3.6 install torchvision==0.6.0 --no-dependencies
   ```
Run ResNet50 Training

```bash
export ROCR_VISIBLE_DEVICES=0
python3.6 micro_benchmarking_pytorch.py --network resnet50 --batch-size 128
```

```
root@isyperf-Precision-Tower-5810:/pt-micro-bench# python3.6 micro_benchmarking_pytorch.py --network resnet50 --batch-size 128
  Found GPU0 Device 66af which is of cuda capability 3.0.
  warnings.warn(old_gpu_warn % (d, name, major, capability[1]))
INFO: running forward and backward for warmup.
INFO: running the benchmark..
OK: finished running benchmark..
```

**Microbenchmark for network:** resnet50

- **Mini batch size [img]:** 128
- **Time per mini-batch:** 0.45507651567459106
- **Throughput [img/sec]:** 201.27138094624996
rocFFT

rocFFT is a software library for computing Fast Fourier Transforms (FFT) written in HIP

https://github.com/ROCmSoftwarePlatform/rocFFT

To build the rocfft-rider test, we need to build from source using the flag: -DBUILD_CLIENTS_TESTS=on

Installation

- sudo apt -y install libboost-program-options-dev libfftw3-dev
- cd ~
- git clone https://github.com/ROCmSoftwarePlatform/rocFFT.git
- cd rocFFT
- mkdir build; cd build
- cmake .. -DCXX=/opt/rocm/bin/hipcc -DBUILD_CLIENTS_BENCHMARKS=ON -DBUILD_CLIENTS_RIDER=ON -DBUILD_CLIENTS_TESTS=on
- make -j

Executables will be in

~/.rocFFT/build/clients/staging

Run tests with the rocfft-rider benchmark executable. For example:

- ./rocfft-rider --length $((2 ** 24)) -b 10
  - in-place
  - Running profile with 1 samples
  - length: 16777216

- Execution gpu time: 44.3606 ms
- Execution gflops: 453.841
MPI and UCX

- Using the installation script to install Open MPI with UCX
  - `setup_rocm_ompi_ucx.sh`

- Run as root
  - `sudo su -`
  - `cd /opt/rocm`
  - `./setup_rocm_ompi_ucx.sh true`

- Expected performance
  - XGMI between 2 GPUs
    - 36GB/s bandwidth at 2MB messages
    - 1.8us latency at 1-byte

- For InfiniBand setup
  - Install MLNX_OFED before ROCm install to ensure PeerDirect support is in place for Mellanox drivers

- More info for Open MPI + UCX

- MPICH support with UCX for AMD GPU also available. To enable MPICH with ROCm-enabled UCX:
  - `./configure --with-device=ch4:ucx --with-ucx=<path/to/ucx/install>`
rocHPCG

rocHPCG is the implementation of HPCG that runs on AMD GPU:
- [https://github.com/ROCmSoftwarePlatform/rocHPCG.git](https://github.com/ROCmSoftwarePlatform/rocHPCG.git)

To build rocHPCG
- `git clone https://github.com/ROCmSoftwarePlatform/rocHPCG.git`
- `cd rocHPCG`
- `./install.sh`

The executable will be located at
- `rocHPCG/build/release/bin/rochpcg`

The local domain size to run for a 16GB GPU should be “280 280 280”

A qualified HPCG run would run for 30 minutes

rocHPCG/build/release/bin/rochpcg 280 280 280 1860

- **DDOT** = 115.8 GFlop/s ( 926.2 GB/s) 115.8 GFlop/s per process ( 926.2 GB/s per process)
- **WAXPBY** = 56.9 GFlop/s ( 683.1 GB/s) 56.9 GFlop/s per process ( 683.1 GB/s per process)
- **SpMV** = 112.7 GFlop/s ( 710.0 GB/s) 112.7 GFlop/s per process ( 710.0 GB/s per process)
- **MG** = 159.0 GFlop/s ( 1227.5 GB/s) 159.0 GFlop/s per process ( 1227.5 GB/s per process)
- **Total** = 145.1 GFlop/s (1100.1 GB/s) 145.1 GFlop/s per process (1100.1 GB/s per process)
- **Final** = 143.7 GFlop/s (1089.7 GB/s) 143.7 GFlop/s per process (1089.7 GB/s per process)
### BabelStream

- BabelStream measures memory transfer rates to/from global device memory on GPUs
- This benchmark is similar in spirit, and based on, the STREAM benchmark for CPUs
- To build BabelStream
  - `git clone https://github.com/UoB-HPC/BabelStream.git`
  - `cd BabelStream; make VERBOSE=1 -f HIP.make`
- To run BabelStream
  - `./hip-stream`
  - BabelStream
  - Version: 3.4
  - Implementation: HIP
  - Running kernels 100 times
  - Precision: double
  - Array size: 268.4 MB (=0.3 GB)
  - Total size: 805.3 MB (=0.8 GB)
  - Using HIP device Vega 20
  - Driver: 313700
  - Function | MBytes/sec | Min (sec) | Max       | Average
  - Copy    | 804349.192  | 0.00067   | 0.00068   | 0.00067
  - Mul     | 805412.280  | 0.00067   | 0.00068   | 0.00067
  - Add     | 775833.239  | 0.00104   | 0.00104   | 0.00104
  - Triad   | 774988.060  | 0.00104   | 0.00104   | 0.00104
  - Dot     | 553257.856  | 0.00097   | 0.00099   | 0.00098
LAMMPS

LAMMPS is a popular molecular dynamics simulation application

LAMMPS has ‘gpu’ and ‘kokkos’ backends to support AMD GPU. The ‘gpu’ backend is shown below.

Install rocPRIM and hipCUB:
- sudo apt install rocprim hipcub

Clone the repo:
- git clone https://github.com/lammps/lammps.git

Get cub 1.8.0 and add it to the LAMMPS libraries:
- wget https://github.com/NVlabs/cub/archive/1.8.0.zip
- unzip 1.8.0.zip; mv cub-1.8.0/ lammps/lib/gpu/

Edit HIP_ARCH in lammps/lib/gpu/Makefile.hip
- set HIP_ARCH = gfx906 for MI50

Set the following environment variable:
- export HIP_PLATFORM=hcc
- cd lammps/lib/gpu; make -f Makefile.hip -j
- Cd lammps/src; make yes-gpu; make hip -j

Run the example, in examples/melt or bench/KEPLER:
- mpirun -np 1 ../../../src/lmp_hip -in in.melt -sf gpu -pk gpu 1
GROMACS

GROMACS - GROningen MAchine for Chemical Simulations
Molecular dynamics package mainly designed for simulations of proteins, lipids, and nucleic acids
The current hipified GROMACS source is in a private repository, enable by request
https://github.com/ROCmSoftwarePlatform/Gromacs.git

Build instructions:

- git clone https://github.com/ROCmSoftwarePlatform/Gromacs.git
- cd Gromacs; git checkout develop-2020.1
- mkdir build
- cd build
- rm -rf ../*build/*
- cmake -DBUILD_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=Release -DCMAKE_C_COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpicxx -DCMAKE_MPI=on -DCMAKE_GPU=on -DCMAKE_GPU_USE_AMD=on -DCMAKE_OPENMP=on -DCMAKE_GPU_DETECTION_DONE=on -DCMAKE_SIMD=AVX2_256 -DREGRESSIONTEST_DOWNLOAD=OFF -DCMAKE_PREFIX_PATH=/opt/rocm -DCMAKE_INSTALL_PREFIX=$HOME/MI50
- make -j install

Alternatively, we can use the GROMACS rocmx docker:

- https://hub.docker.com/r/rocmx/gromacs
- sudo docker run --name=gromacs_docker_script rocmx/hpc:rocm_3.3_hpc_gromacs_2020.1_a
NAMD

NAnoscale Molecular Dynamics (NAMD)

- NAMD is a highly scalable molecular dynamics (MD) code
- NAMD geared towards the simulation of large biomolecular systems

The current hipified NAMD source is in a private repository, enable by request

- [https://github.com/ROCmSoftwarePlatform/NAMD](https://github.com/ROCmSoftwarePlatform/NAMD)

We can use the NAMD docker to run NAMD

```bash
docker run -it --privileged --device=/dev/kfd --device=/dev/dri/ --cap-add=SYS_RAWIO --device=/dev/mem --group-add video --network host jparada/ubuntu-18.04_namd:rocm-3.3_0416
cd ~/NAMD/NAMD_benchmarks/
source ~/namd_hip.rc
python3 run_benchmarks.py -b apoA1 stmv -c 16-16 -d 0 # 1x GPU
python3 run_benchmarks.py -b apoA1 stmv -c 16-16 -d 0,1,2,3,4,5,6,7 # 8x GPUs
```
HIP: High Performance & Portable

C++ runtime API and kernel language that allows developers to create portable applications that can run on AMD’s accelerators as well as CUDA devices.

- Is open-source
- Provides an API for an application to leverage GPU acceleration for both AMD and CUDA devices
- Syntactically similar to CUDA. Most CUDA API calls can be converted in place: cuda -> hip
- Supports a strong subset of CUDA runtime functionality
CUDA VECTOR ADD

```c
__global__ void add(int n, double *x, double *y)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;
    for (int i = index; i < n; i += stride)
    {
        y[i] = x[i] + y[i];
    }
```

HIP VECTOR ADD

```c
__global__ void add(int n, double *x, double *y)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;
    for (int i = index; i < n; i += stride)
    {
        y[i] = x[i] + y[i];
    }
```

KERNELS ARE SYNTACTICALLY IDENTICAL
## Seamless Porting from CUDA APIs

<table>
<thead>
<tr>
<th>CUDA</th>
<th>HIP</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cudaMemcpyAsync(d_npos,h_npos,sizeof(float4)*SIZE,cudaMemcpyHostToDevice,stream);</code></td>
<td><code>hipMemcpyAsync(d_npos,h_npos,sizeof(float4)*SIZE,hipMemcpyHostToDevice,stream);</code></td>
</tr>
<tr>
<td><code>cudaMemcpyAsync(d_mask,h_mask,sizeof(MASK_T)*cnt,cudaMemcpyHostToDevice,stream);</code></td>
<td><code>hipMemcpyAsync(d_mask,h_mask,sizeof(MASK_T)*cnt,hipMemcpyHostToDevice,stream);</code></td>
</tr>
<tr>
<td><code>calcHHCullenDehnen&lt;&lt;&lt;blocksPerGrid,threadsPerBlock,0,stream&gt;&gt;&gt;(cnt, SIZE, d_npos, d_mask, rsm);</code></td>
<td><code>hipLaunchKernelGGL((calcHHCullenDehnen), dim3(blocksPerGrid), dim3(threadsPerBlock), 0, stream, cnt, SIZE, d_npos, d_mask, rsm);</code></td>
</tr>
<tr>
<td><code>cudaMemcpyAsync(h_pos,d_npos+(SIZE-cnt),sizeof(float4)*cnt,cudaMemcpyDeviceToHost,stream);</code></td>
<td><code>hipMemcpyAsync(h_pos,d_npos+(SIZE-cnt),sizeof(float4)*cnt,hipMemcpyDeviceToHost,stream);</code></td>
</tr>
<tr>
<td><code>cudaMemcpyAsync(h_mask,d_mask,sizeof(MASK_T)*cnt,cudaMemcpyDeviceToHost,stream);</code></td>
<td><code>hipMemcpyAsync(h_mask,d_mask,sizeof(MASK_T)*cnt,hipMemcpyDeviceToHost,stream);</code></td>
</tr>
</tbody>
</table>
AMD GPU Libraries

A note on naming conventions:
- roc* -> AMDGCN library usually written in HIP
- cu* -> NVIDIA PTX libraries
- hip* -> usually interface layer on top of roc*/cu* backends

hip* libraries:
- Can be compiled by hipcc and can generate a call for the device you have:
  - hipcc->AMDGCN
  - hipcc->nvcc (inlined)->NVPTX

rocBLAS

hipBLAS

cuBLAS
## CUDA Equivalent Libraries

<table>
<thead>
<tr>
<th>CUDA Library</th>
<th>ROCm Library</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuBLAS</td>
<td>rocBLAS</td>
<td>Basic Linear Algebra Subroutines</td>
</tr>
<tr>
<td>cuFFT</td>
<td>rocFFT</td>
<td>Fast Fourier Transfer Library</td>
</tr>
<tr>
<td>cuSPARSE</td>
<td>rocSPARSE</td>
<td>Sparse BLAS + SPMV</td>
</tr>
<tr>
<td>cuSolver</td>
<td>rocSolver</td>
<td>Lapack Library</td>
</tr>
<tr>
<td>AMG-X</td>
<td>rocALUTION</td>
<td>Sparse iterative solvers &amp; preconditioners with Geometric &amp; Algebraic MultiGrid</td>
</tr>
<tr>
<td>Thrust</td>
<td>hipThrust</td>
<td>C++ parallel algorithms library</td>
</tr>
<tr>
<td>CUB</td>
<td>rocPRIM</td>
<td>Low Level Optimized Parallel Primitives</td>
</tr>
<tr>
<td>cuDNN</td>
<td>MIOpen</td>
<td>Deep learning Solver Library</td>
</tr>
<tr>
<td>cuRAND</td>
<td>rocRAND</td>
<td>Random Number Generator Library</td>
</tr>
<tr>
<td>EIGEN</td>
<td>EIGEN – HIP port</td>
<td>C++ template library for linear algebra: matrices, vectors, numerical solvers</td>
</tr>
<tr>
<td>NCCL</td>
<td>RCCL</td>
<td>Communications Primitives Library based on the MPI equivalents</td>
</tr>
</tbody>
</table>
HIPIFY Tools:
Converting CUDA Code for Portability

**Hipify-perl**
- Easy to use – point at a directory and it will attempt to hipify CUDA code
- Very simple string replacement technique: may make incorrect translations
- `sed -e 's/cuda/hip/g', (e.g., cudaMemcpy becomes hipMemcpy)`
- Recommended for quick scans of projects

**Hipify-clang**
- Requires clang compiler to parse files and perform semantic translation
- More robust translation of the code
- Generates warnings and assistance for additional analysis
- High quality translation, particularly for cases where the user is familiar with the make system
Getting QUDA Rocking with HIP

HIP Solutions

- QUDA depends on many CUDA libraries
  - Eigen - hip support in Eigen's development branch, CuFFT + hipFFT, CuBLAS - rocBLAS + hipBLAS, CuRAND - rocRAND + hipRAND, Thrust - hipThrust, CUB - hipCUB

- QUDA is a large project (For a single-person porting project!)
  - 10000 lines of hand-tuned CUDA kernels - hipify converted these without problems
  - 35000 lines of header code - hipify mostly converted these, but needed manual switch to new library dependencies
  - 74000 lines of library code - Mostly successful hipify conversion, required some manual changes
  - 34000 lines of test suite code – hipify converted without problems
  - Heavily interconnected due to template use – No solution to this, it is part of library design

Results

- Time to running executable on AMD hardware
  - 15 developer-days

- Without AMD tools (hip) and library support, would have taken significantly longer.

- Work ongoing for rest of test suite
**hipify-perl**

- **hipify-perl** is autogenerated perl-based script which heavily uses regular expressions

**Advantages**
- Ease in use
- No need to check the input source CUDA code for correctness
- No dependencies on 3rd party tools, including CUDA

**Disadvantages**
- Limitation in transforming the following constructs
  - macros expansion
  - namespaces
    - redefines of CUDA entities in user namespaces
    - using directive
  - templates (some cases)
  - device/host function calls distinguishing
  - header files correct injection
  - complicated argument lists parsing

**Available in ROCm install:**
- /opt/rocm/bin/hipify-perl
- Convert all files in a directory
  - /opt/rocm/bin/hipconvertingplace.sh
**hipify-clang**

- **hipify-clang** is a clang-based tool for translation CUDA sources into HIP sources
- Translates CUDA source into abstract syntax tree, then traversed by transformation matchers
- After applying all the matchers, the output HIP source is produced

**Advantages:**
- It is a translator, therefore complicated constructs can be parsed successfully, or an error will be reported
- Supports clang options like -I, -D, --cuda-path, etc
- Seamless support of new CUDA versions for LLVM Clang
- Easier to support

**Disadvantages:**
- CUDA should be installed and provided in case of multiple installations by --cuda-path option
- The input CUDA code needs to be compliable. Incorrect code cannot be translated to HIP
- Include’s and define’s should be provided to transform code successfully

**Available in ROCm repo for download:**
- `apt install hipify-clang`
- Or build from HIPIFY github
- [https://github.com/ROCm-Developer-Tools/HIPIFY](https://github.com/ROCm-Developer-Tools/HIPIFY)
Hipify Samples

**HIP Samples**
- In `/opt/rocm/hip/samples/0_Intro/square`

**SpecFEM3D Cartesian**
- Fortran code base with one C file to abstract GPU stubs
- Very clean GPU implementation with all 18 *.cu files contained in one directory:`/specfem3d/src/cuda`
- Porting process: (10 min)
  - `~/specfem3d/src$ hipconvertinplace-perl.sh`
- Minor build changes:
  - Makefile and configuration work – 100 line section to modify and add AMD support
- Converted 1120 CUDA->HIP refs in 16783 Lines of Code
  - with 1 warning: comment containing the word “CUDA”
  - ~500 were memory management (hipMemcpy, hipFree, hipMemcpyHostToDevice, hipMalloc, hipMemcpyDeviceToHost, hipMemcpy2D, hipMemset, etc)
  - ~250 numeric literal operations
- 86 kernel launches of 15 separate kernels:

<table>
<thead>
<tr>
<th>compute_acoustic_vectorial_seismogram_kernel</th>
<th>store_dataT</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute_kernels_hess_ac_cuda_kernel</td>
<td>kernel_3_acoustic_cuda_device</td>
</tr>
<tr>
<td>noise_read_add_surface_movie_cuda_kernel</td>
<td>Kernel_2_noatt_iso_impl</td>
</tr>
<tr>
<td>add_sources_el_SIM_TYPE_2_OR_3_kernel</td>
<td>enforce_free_surface_cuda_kernel</td>
</tr>
<tr>
<td>assemble_boundary_potential_on_device</td>
<td>compute_coupling_elastic_ac_kernel</td>
</tr>
<tr>
<td>compute_stacey_acoustic_kernel</td>
<td>check_array_ispec_kernel</td>
</tr>
<tr>
<td>add_sources_ac_SIM_TYPE_2_OR_3_kernel</td>
<td>compute_stacey_elastic_sim3_kernel</td>
</tr>
<tr>
<td>assemble_boundary_accel_on_device</td>
<td></td>
</tr>
</tbody>
</table>
General information and resources

- **ROCm platform**: [https://github.com/RadeonOpenCompute/ROCm/](https://github.com/RadeonOpenCompute/ROCm/)
  - With instructions for installing from binary repositories, and links to source repositories for all components
- **ROCm/HIP libraries**: [https://github.com/ROCmSoftwarePlatform](https://github.com/ROCmSoftwarePlatform)
- **rocprofiler**: [https://github.com/ROCm-Developer-Tools/rocprofiler](https://github.com/ROCm-Developer-Tools/rocprofiler)
  - Collects application traces and performance counters
  - Trace timeline can be visualized with chrome://tracing
- **YouTube videos**
  - Includes YouTube videos on ROCm software, programming concepts and more details on hardware devices
  - [https://community.amd.com/community/radeon-instinct-accelerators/blog/2020/06/10/rocm-open-software-ecosystem-for-accelerated-compute](https://community.amd.com/community/radeon-instinct-accelerators/blog/2020/06/10/rocm-open-software-ecosystem-for-accelerated-compute)
References

- ROCm™ Installation

- ROCm Bandwidth Test
  https://github.com/RadeonOpenCompute/rocm_bandwidth_test

- RVS Installation
  https://github.com/ROCm-Developer-Tools/ROCMValidationSuite

- TensorFlow Installation & Benchmark
  https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks

- PyTorch Installation & Benchmark
  https://github.com/ROCmSoftwarePlatform/pytorch/wiki/Performance-analysis-of-PyTorch

- Link to more training information
  https://community.amd.com/community/radeon-instinct-accelerators/blog/