f

AMD ROCm™

New Feature Document

for

HIP-Clang, VDI, GDB

ROCm v2.10

Publication # **1.0**	Revision: **1.0**
Issue Date: **February 2020**	

As of Jun 19, 2019. Radeon Instinct™ MI50 and MI60 "Vega 7nm" technology-based accelerators support PCIe® Gen 4.0* providing up to 64 GB/s peak theoretical transport data bandwidth from CPU to GPU per card. Previous Gen Radeon Instinct compute GPU cards are based on PCIe Gen 3.0 providing up to 32 GB/s peak theoretical transport rate bandwidth performance. Peak theoretical transport rate performance is calculated by Baud Rate * width in bytes *# directions = GB/s per card. PCIe Gen3: 8 * 2 * 2 = 32 GB/s. PCIe Gen4: 16 * 2 * 2 = 64 GB/s.
Radeon Instinct™ MI50 and MI60 "Vega 7nm" technology-based accelerators include dual Infinity Fabric™ Links providing up to 184 GB/s peak theoretical GPU to GPU or Peer-to-Peer (P2P) transport rate bandwidth performance per GPU card. Combined with PCIe Gen 4 compatibility providing an aggregate GPU card I/O peak bandwidth of up to 248 GB/s. Performance guidelines are estimated only and may vary. Previous Gen Radeon Instinct compute GPU cards provide up to 32 GB/s peak PCIe Gen 3.0 bandwidth performance. Infinity Fabric Link technology peak theoretical transport rate performance is calculated by Baud Rate * width in bytes *# directions *# links = GB/s per card. Infinity Fabric Link: 23 * 2 * 2 = 92 GB/s. MI50 | MI60 each have two links: 92 GB/s * 2 links per GPU = 184 GB/s. Refer to server manufacture PCIe Gen 4.0 compatibility and performance guidelines for potential peak performance of the specified server model numbers. Server manufacturers may vary configuration offerings yielding different results. https://pcisig.com/ , https://www.chipestimate.com/PCI-Ex- press-Gen-4-a-Big-Pipe-for-Big-Data/Cadence/Technical-Article/2014/04/15 , https://www.tomshardware.com/news/pcie-4.0-power-speed-express,32525.html AMD has not independently tested or verified external/third party results/data and bears no responsibility for any errors or omissions therein. RIV-18

Table of Contents

Chapter 1 What's New

This guide provides information on the following new features for v2.10:

- HIP compiler changed from HCC to HIP-Clang
- Runtime Virtual Device Interface (VDI)
- GNU Debugger (GDB)

1.1 HIP-Clang

The HIP compiler and runtime infrastructure for HIP are now changed from HCC to HIP-Clang.

The new HIP compiler shares the same code base as CUDA-clang and conforms better to C++ standard and CUDA language syntax/semantics. The LLVM/Clang community has implemented a strict review of the new HIP compiler code.

1.1.1 Supported Operating Systems

1.1.1.1 Software Support

- Ubuntu 16.04.6(Kernel 4.15)
- CentOS v7.5 (Using devtoolset-7 runtime support)

Known Issue on CentOS

Issue: When running the HIP sample under `rocmgdb` on **CentOS**, the following error may appear:

```
"error while loading shared libraries: libhip_hcc.so: cannot open shared object file: No such file or directory"
```

Resolution: This is a known issue. Use the following workaround to resolve the issue:

```
export LD_LIBRARY_PATH=$LD_LIRBARY_PATH:/opt/rocm/hip/lib
```

1.1.1.2 Hardware Support

The following AMD Radeon™ GPUs are supported:

- GPUs previously code-named "Vega 10", also denoted as gfx900
- GPUs previously code-named "Vega 7nm", also denoted as gfx906
- GPUs code-named "Arcturus", also denoted as gfx908

1.1.2 New Features

The new HIP-Clang compiler provides the following new features:

- Triple-chevron syntax is supported to launch kernels
- GPU architecture-specific code: Users can use predefined macros. For example, `#if __gfx900__` to conditionally compile code for a specific GPU architecture.
- GPU architecture built-in functions: Users can use Clang built-in functions available for a specific GPU architecture. For example, `__builtin_amdgcn_fdot2`, which is only available on gfx9+.
- Use constant address space instead of global address space for `__constant__` variables, which results in better performance.
- Use `-ffp-contract=fast` by default, which is consistent with CUDA-Clang and NVCC. This results in better performance and increased accuracy for FMA operations.
- Full support of Clang `-v`, `-###`, and `-save-temps` options to facilitate debugging
- compilation issues.
- Support device and host-only compilation by Clang options `--cuda-device-only` and `--cuda-host-only`. Support clang option `-emit-llvm` to emit LLVM bitcode.
- Function call support is enabled by default.
- Uses code object version 3, by default, to support device code debugging.
- Support clang option `-parallel-jobs=n` for speeding up compilation. This option allows compiling device code for different GPU architectures in parallel.

For a list of available AMDGPU target-specific built-ins, see

https://github.com/llvm/llvmproject/blob/master/clang/include/clang/Basic/BuiltinsAMDGPU.def

1.2 Runtime Virtual Device Interface

The new HIP runtime is built on top of the Virtual Device Interface (VDI) component, which is used for OpenCL. The VDI enables HIP to use different backend systems, including PAL and ROCm.

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

```
scl enable devtoolset 7 bash
```

1.3 Known Issues for Hip-Clang and VDI

The following known issues will be fixed in a future release.

- RCCL does not work and tests fail on multi-node configurations
- Compilation time for PyTorch and Caffe2 with HIP-Clang is significantly higher compared to HIP/HCC
- Application and framework performance on HIP-Clang + VDI is not optimized. There is a loss of performance when compared to the existing HIP/HCC infrastructure
- Caffe2 bench.sh seemingly crashes in our test environments. The root cause is being triaged
- TensorFlow tests fail on Vega7nm
- TensorFlow unit tests and Convolutional Neural Networks (CNN) benchmarks crash on Vega 7nm
- *printf* in kernels do not work as the *device_printf* feature is not enabled on HIP Clang and VDI

1.4 ROC GNU Debugger

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

The current ROCm Debugger (ROCgdb) is an initial prototype that focuses on source-line debugging and does not provide symbolic variable debugging capabilities. The user guide presents features and commands that may be implemented in future versions.

For more information about ROCm v2.10, see:

https://github.com/RadeonOpenCompute/ROCm

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

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

You can refer to the following chapters in the **ROCgdb User Guide** for more specific information about debugging heterogenous programs on ROCm:

- *Debugging Heterogeneous Programs* provides general information about debugging heterogeneous programs.
- *Configuration-Specific Information > Architectures > AMD GPU* provides specific information about debugging heterogenous programs on ROCm with supported AMD GPU chips. This section also lists the features, commands, and known issues that may be implemented and resolved in future releases.

For more information about the GNU Debugger (GDB), check the GNU Debugger (GDB) web site at:

http://www.gnu.org/software/gdb