AMD ROCm™ v4.3
Release Notes

Revision	**0802**
Issue Date:	**August 2021**

0802 Rev. August 2021

DISCLAIMER

[This page is left blank intentionally]

Table of Contents

ROCm Installation Updates

This document describes the features, fixed issues, and information about downloading and installing the AMD ROCm™ software.

It also covers known issues and deprecations in this release.

LIST OF SUPPORTED OPERATING SYSTEMS

The AMD ROCm platform supports the following operating systems:

OS	Kernel
SLES15 SP2	5.3.18-24.49
RHEL 7.9	3.10.0-1160.6.1.el7
CentOS 7.9	3.10.0-1127
RHEL 8.3	4.18.0-193.1.1.el8
CentOS 8.3	4.18.0-193.el8
Ubuntu 18.04.5	5.4.0-71-generic
Ubuntu 20.04.2	5.8.0-48-generic
Host OS	Azure RS1.86
Guest OS	Ubuntu 20.04

COMPLETE INSTALLATION OF AMD ROCM V4.3 RECOMMENDED

Complete uninstallation of previous ROCm versions is required before installing a new version of ROCm. **An upgrade from earlier releases to AMD ROCm v4.3 is not supported.** For more information, refer to the *AMD ROCm Installation Guide.*

Note: AMD ROCm release v3.3 or prior releases are not fully compatible with AMD ROCm v3.5 and higher versions. You must perform a fresh ROCm installation if you want to upgrade from AMD ROCm v3.3 or older to 3.5 or higher versions and vice-versa.

Note: *render group* is required only for Ubuntu v20.04. For all other ROCm supported operating systems, continue to use *video group*.

- For ROCm v3.5 and releases thereafter, the *clinfo* path is changed to */opt/rocm/opencl/bin/clinfo*.
- For ROCm v3.3 and older releases, the *clinfo* path remains */opt/rocm/opencl/bin/x86_64/clinfo*.

ROCM MULTI-VERSION INSTALLATION UPDATE

With the AMD ROCm v4.3 release, the following ROCm multi-version installation changes apply:

The meta packages *rocm-dkms<version>* are now deprecated for multi-version ROCm installs. For example, *rocm-dkms3.7.0, rocm-dkms3.8.0*.

- Multi-version installation of ROCm should be performed by installing *rocm-dev<version>* using each of the desired ROCm versions. For example, *rocm-dev3.7.0, rocm-dev3.8.0, rocm-dev3.9.0.*
- Install the rock-dkms loadable kernel modules using a single *rock-dkms* package.
- ROCm v4.3 and above will not set any *ldconfig* entries for ROCm libraries for multi-version installation. Users must set *LD_LIBRARY_PATH* to load the ROCm library version of choice.

NOTE: The single version installation of the ROCm stack remains the same. The *rocm-dkms* package can be used for single version installs and is not deprecated at this time.

SUPPORT FOR ENVIRONMENT MODULES

Environment modules are now supported. This enhancement in the ROCm v4.3 release enables users to switch between ROCm v4.2 and ROCm v4.3 easily and efficiently.

For more information about installing environment modules, refer to

https://modules.readthedocs.io/en/latest/

AMD ROCm V4.3 Documentation Updates

AMD ROCM INSTALLATION GUIDE

The AMD ROCm Installation Guide in this release includes the following updates:

- Supported Environments
- Installation Instructions
- HIP Installation Instructions

HIP DOCUMENTATION UPDATES

- HIP Programming Guide v4.3

  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_HIP_Programming_Guide_v4.3.pdf

- HIP API Guide v4.3

  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_HIP_API_Guide_v4.3.pdf

- HIP-Supported CUDA API Reference Guide v4.3

  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_HIP_Supported_CUDA_API_Reference_Guide_v4.3.pdf

- **NEW** - AMD ROCm Compiler Reference Guide v4.3

  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_Compiler_Reference_Guide_v4.3.pdf

- HIP FAQ

  https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-FAQ.html#hip-faq

ROCM DATA CENTER TOOL USER AND API GUIDE

- ROCm Data Center Tool User Guide
  - Prometheus (Grafana) Integration with Automatic Node Detection

  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_ROCm_DataCenter_Tool_User_Guide_v4.3.pdf

- ROCm Data Center Tool API Guide

  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_RDC_API_Guide_v4.3.pdf

ROCM SMI API GUIDE

- ROCm SMI API Guide
  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_ROCm_SMI_Guide_v4.3.pdf

ROC DEBUGGER USER AND API GUIDE

- ROCDebugger User Guide
  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_ROCDebugger_User_Guide.pdf
- Debugger API Guide
  https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_ROCDebugger_API.pdf

AMD ROCM GENERAL DOCUMENTATION LINKS

- For AMD ROCm documentation, see
  https://rocmdocs.amd.com/en/latest/
- For installation instructions on supported platforms, see
  https://rocmdocs.amd.com/en/latest/Installation_Guide/Installation-Guide.html
- For AMD ROCm binary structure, see
  https://rocmdocs.amd.com/en/latest/Installation_Guide/Software-Stack-for-AMD-GPU.html
- For AMD ROCm release history, see
  https://rocmdocs.amd.com/en/latest/Current_Release_Notes/ROCm-Version-History.html

What's New in This Release

HIP ENHANCEMENTS

The ROCm v4.3 release consists of the following HIP enhancements:

HIP Versioning Update

The HIP version definition is updated from the ROCm v4.2 release as follows:

```
HIP_VERSION=HIP_VERSION_MAJOR * 10000000 + HIP_VERSION_MINOR * 100000 +
HIP_VERSION_PATCH)
```

The HIP version can be queried from a HIP API call

```
hipRuntimeGetVersion(&runtimeVersion);
```

Note: The version returned will be greater than the version in previous ROCm releases.

Support for Managed Memory Allocation

HIP now supports and automatically manages Heterogeneous Memory Management (HMM) allocation. The HIP application performs a capability check before making the managed memory API call *hipMallocManaged*.

Note: The *_managed_* keyword is unsupported currently.
For example,

```
int managed_memory = 0;
HIPCHECK(hipDeviceGetAttribute(&managed_memory,
  hipDeviceAttributeManagedMemory,p_gpuDevice));
if (!managed_memory ) {
  printf ("info: managed memory access not supported on the device %d\n
Skipped\n", p_gpuDevice);
}
else {
  HIPCHECK(hipSetDevice(p_gpuDevice));
  HIPCHECK(hipMallocManaged(&Hmm, N * sizeof(T)));
. . .
}
```

Kernel Enqueue Serialization

Developers can control kernel command serialization from the host using the following environment variable,
AMD_SERIALIZE_KERNEL

- AMD_SERIALIZE_KERNEL = 1, Wait for completion before enqueue,
- AMD_SERIALIZE_KERNEL = 2, Wait for completion after enqueue,
- AMD_SERIALIZE_KERNEL = 3, Both.

This environment variable setting enables HIP runtime to wait for *GPU idle* before/after any GPU command.

NUMA-aware Host Memory Allocation

The Non-Uniform Memory Architecture (NUMA) policy determines how memory is allocated and selects a CPU closest to each GPU.

NUMA also measures the distance between the GPU and CPU devices. By default, each GPU selects a Numa CPU node that has the least NUMA distance between them; the host memory is automatically allocated closest to the memory pool of the NUMA node of the current GPU device.

Note, using the *hipSetDevice* API with a different GPU provides access to the host allocation. However, it may have a longer NUMA distance.

New Atomic System Scope Atomic Operations

HIP now provides new APIs with *_system* as a suffix to support system scope atomic operations. For example, *atomicAnd atomic* is dedicated to the GPU device, and *atomicAnd_system* allows developers to extend the atomic operation to system scope from the GPU device to other CPUs and GPU devices in the system.

For more information, refer to the HIP Programming Guide at,

https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_HIP_Programming_Guide_v4.3.pdf

Indirect Function Call and C++ Virtual Functions

While the new release of the ROCm compiler supports indirect function calls and C++ virtual functions on a device, there are some known limitations and issues.

Limitations

- An address to a function is device specific. Note, a function address taken on the host can not be used on a device, and a function address taken on a device can not be used on the host. On a system with multiple devices, an address taken on one device can not be used on a different device.
- C++ virtual functions only work on the device where the object was constructed.
- Indirect call to a device function with function scope shared memory allocation is not supported. For example, LDS.
- Indirect call to a device function defined in a source file different than the calling function/kernel is only supported when compiling the entire program with *-fgpu-rdc*.

Known Issues in This Release

- Programs containing kernels with different launch bounds may crash when making an indirect function call. This issue is due to a compiler issue miscalculating the register budget for the callee function.
- Programs may not work correctly when making an indirect call to a function that uses more resources. For example, scratch memory, shared memory, registers made available by the caller.
- Compiling a program with objects with pure or deleted virtual functions on the device will result in a linker error. This issue is due to the missing implementation of some C++ runtime functions on the device.
- Constructing an object with virtual functions in private or shared memory may crash the program due to a compiler issue when generating code for the constructor.

ROCM DATA CENTER TOOL

Prometheus (Grafana) Integration with Automatic Node Detection

The ROCm Data Center (RDC) tool enables you to use Consul to discover the rdc_prometheus service automatically. Consul is “a service mesh solution providing a full-featured control plane with service discovery, configuration, and segmentation functionality.” For more information, refer to their website at *https://www.consul.io/docs/intro*.

The ROCm Data Center Tool uses Consul for health checks of RDC’s integration with the Prometheus plug-in (*rdc_prometheus*), and these checks provide information on its efficiency.

Previously, when a new compute node was added, users had to change *prometheus_targets.json* to use Consul manually. Now, with the Consul agent integration, a new compute node can be discovered automatically.

https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_ROCm_DataCenter_Tool_User_Guide_v4.3.pdf

Coarse Grain Utilization

This feature provides a counter that displays the coarse grain GPU usage information, as shown below.

Sample output

```
               $ rocm_smi.py --showuse
============================== % time GPU is busy ==============================
               GPU[0] : GPU use (%): 0
               GPU[0] : GFX Activity: 3401
```

Add 64-bit Energy Accumulator In-band

This feature provides an average value of energy consumed over time in a free-flowing RAPL counter, a 64-bit Energy Accumulator.
Sample output

```
      $ rocm_smi.py --showenergycounter
============================== Consumed Energy
================================
GPU[0] : Energy counter: 2424868
```

```
GPU[0] : Accumulated Energy (uJ): 0.0
```

Support for Continuous Clocks Values

ROCm SMI will support continuous clock values instead of the previous discrete levels. Moving forward the updated sysfs file will consist of only MIN and MAX values and the user can set the clock value in the given range.
Sample output:

```
$ rocm_smi.py --setsrange 551 1270
Do you accept these terms? [y/N]
y

============================== Set Valid sclk Range=======
GPU[0]          : Successfully set sclk from 551(MHz) to
1270(MHz)
GPU[1]          : Successfully set sclk from 551(MHz) to
1270(MHz)
================================================================================


$ rocm_smi.py --
showsclkrange


============================== Show Valid sclk Range======


GPU[0]          : Valid sclk range: 551Mhz -
1270Mhz
GPU[1]          : Valid sclk range: 551Mhz - 1270Mhz
```

Memory Utilization Counters

This feature provides a counter display memory utilization information as shown below.

Sample output

```
       $ rocm_smi.py --showmemuse
========================== Current Memory Use ============================


GPU[0] : GPU memory use (%): 0
GPU[0] : Memory Activity: 0
```

Performance Determinism

ROCm SMI supports performance determinism as a unique mode of operation. Performance variations are minimal as this enhancement allows users to control the entry and exit to set a soft maximum (ceiling) for the GFX clock.

Sample output

```
$ rocm_smi.py --setperfdeterminism 650
cat pp_od_clk_voltage
GFXCLK:
0: 500Mhz
1: 650Mhz *
2: 1200Mhz
$ rocm_smi.py --resetperfdeterminism
```

Note: The idle clock will not take up higher clock values if no workload is running. After enabling determinism, users can run a GFX workload to set performance determinism to the desired clock value in the valid range.

- GFX clock could either be less than or equal to the max value set in this mode. GFX clock will be at the max clock set in this mode only when required by the running workload.
- VDDGFX will be higher by an offset (75mv or so based on PPTable) in the determinism mode.

HBM Temperature Metric Per Stack

This feature will enable ROCm SMI to report all HBM temperature values as shown below. Sample output

```
   $ rocm_smi.py –showtemp
================================ Temperature ================================
GPU[0] : Temperature (Sensor edge) (C): 29.0
GPU[0] : Temperature (Sensor junction) (C): 36.0
GPU[0] : Temperature (Sensor memory) (C): 45.0
GPU[0] : Temperature (Sensor HBM 0) (C): 43.0
GPU[0] : Temperature (Sensor HBM 1) (C): 42.0
GPU[0] : Temperature (Sensor HBM 2) (C): 44.0
GPU[0] : Temperature (Sensor HBM 3) (C): 45.0
```

ROCM MATH AND COMMUNICATION LIBRARIES

In this release, ROCm Math and Communication Libraries consists of the following enhancements and fixes:

Library	Changes
rocBLAS	**Optimizations** ▪ Improved performance of non-batched and batched rocblas_Xgemv for gfx908 when m <= 15000 and n <= 15000 ▪ Improved performance of non-batched and batched rocblas_sgemv and rocblas_dgemv for gfx906 when m <= 6000 and n <= 6000 ▪ Improved the overall performance of non-batched and batched rocblas_cgemv for gfx906 ▪ Improved the overall performance of rocblas_Xtrsv
hipBLAS	**Enhancements** ▪ Added hipblasStatusToString **Fixed** ▪ Added catch() blocks around API calls to prevent the leak of C++ exceptions
rocFFT	**Changes** • Re-split device code into single-precision, double-precision, and miscellaneous kernels. **Fixed Issues** • double-precision planar->planar transpose. • 3D transforms with unusual strides, for SBCC-optimized sizes. • Improved buffer placement logic.
hipFFT	**Fixed Issues** • CMAKE updates • Added callback API in hipfftXt.h header.
rocSPARSE	**Enhancements** ▪ (batched) tridiagonal solver with and without pivoting ▪ dense matrix sparse vector multiplication (gemvi) ▪ support for gfx90a ▪ sampled dense-dense matrix multiplication (sddmm) **Improvements**

	• client matrix download mechanism • boost dependency in clients removed
rocALUTION	**Enhancements** • Support for gfx90a target • Support for gfx1030 target **Improvements** ▪ Install script
rocSOLVER	**Enhancements** ▪ Linear solvers for general non-square systems: ▪ GELS now supports underdetermined and transposed cases ▪ Inverse of triangular matrices ▪ TRTRI (with batched and strided_batched versions) ▪ Out-of-place general matrix inversion ▪ GETRI_OUTOFPLACE (with batched and strided_batched versions) ▪ Argument names for the benchmark client now match argument names from the public API **Fixed Issues** • Known issues with Thin-SVD. The problem was identified in the test specification, not in the thin-SVD implementation or the rocBLAS gemm_batched routines. • Benchmark client longer crashes as a result of leading dimension or stride arguments not being provided on the command line. **Optimizations** ▪ Improved general performance of matrix inversion (GETRI)
rocTHRUST	**Enhancements** ▪ Updated to match upstream Thrust 1.11 ▪ gfx90a support added ▪ gfx803 support re-enabled
RCCL	**Enhancements** ▪ Ability to select the number of channels to use for clique-based all reduce (RCCL_CLIQUE_ALLREDUCE_NCHANNELS). This can be adjusted to tune for performance when computation kernels are being executed in parallel. **Optimizations**

	▪ Additional tuning for clique-based kernel AllReduce performance (still requires opt in with RCCL_ENABLE_CLIQUE=1) ▪ Modification of default values for number of channels / byte limits for clique-based all reduce based on device architecture **Changes** ▪ Replaced RCCL_FORCE_ENABLE_CLIQUE to RCCL_CLIQUE_IGNORE_TOPO ▪ Clique-based kernels can now be enabled on topologies where all active GPUs are XGMI-connected ▪ Topologies not normally supported by clique-based kernels require RCCL_CLIQUE_IGNORE_TOPO=1 **Fixed Issues** ▪ Install script '-r' flag invoked alone no longer incorrectly deletes any existing builds.
hipCUB	**Enhancements** ▪ DiscardOutputIterator to backend header
rocPRIM	**Fixed** • Bugfix & minor performance improvement for merge_sort when input and output storage are the same. **Enhancements** • gfx90a support added. **Deprecated** • The warp_size() function is now deprecated; please switch to host_warp_size() and device_warp_size() for host and device references respectively.
rocRAND	**Enhancements** • gfx90a support added • gfx1030 support added • gfx803 supported re-enabled **Fixed** • Memory leaks in Poisson tests has been fixed. • Memory leaks when generator has been created but setting seed/offset/dimensions display an exception has been fixed.

For more information about ROCm Libraries, refer to the documentation at

https://rocmdocs.amd.com/en/latest/ROCm_Libraries/ROCm_Libraries.html

ROCPROFILER ENHANCEMENTS

Tracing Multiple MPI Ranks

When tracing multiple MPI ranks in ROCm v4.3, users must use the form:

```
mpirun ... <mpi args> ... rocprof ... <rocprof args> ... application ...
<application args>
```

NOTE: This feature differs from ROCm v4.2 (and lower), which used *"rocprof ... mpirun ... application"*.

This change was made to enable ROCProfiler to handle process forking better and launching via *mpirun* (and related) executables. From a user perspective, this new execution mode requires:

1. Generation of trace data per MPI (or process) rank.

2. Use of a new "merge_traces.sh" utility script (see: <insert link here>) to combine traces from multiple processes into a unified trace for profiling.

For example, to accomplish step #1, ROCm provides a simple bash wrapper that demonstrates how to generate a unique output directory per process:

```
$ cat wrapper.sh
#! /usr/bin/env bash
if [[ -n ${OMPI_COMM_WORLD_RANK+z} ]]; then
  # mpich
  export MPI_RANK=${OMPI_COMM_WORLD_RANK}
elif [[ -n ${MV2_COMM_WORLD_RANK+z} ]]; then
  # ompi
  export MPI_RANK=${MV2_COMM_WORLD_RANK}
fi
args="$*"
pid="$$"
outdir="rank_${pid}_${MPI_RANK}"
outfile="results_${pid}_${MPI_RANK}.csv"
```

```
eval "rocprof -d ${outdir} -o ${outdir}/${outfile} $*"
```

This script:

- Determines the global MPI rank (implemented here for OpenMPI and MPICH only)
- Determines the process id of the MPI rank
- Generates a unique output directory using the two

To invoke this wrapper, use the following command:

```
mpirun <mpi args> ./wrapper.sh --hip-trace <application> <args>
```

This generates an output directory for each used MPI rank. For example,

```
$ ls -ld rank_* | awk {'print $5" "$9'}
4096 rank_513555_0
4096 rank_513556_1
```

Finally, these traces may be combined using the merge traces script (<insert link here>). For example,

```
$  ./merge_traces.sh -h
Script for aggregating results from multiple rocprofiler out directries.
Full path: /opt/rocm/bin/merge_traces.sh
Usage:
  merge_traces.sh -o <outputdir> [<inputdir>...]
```

Use the following input arguments to the *merge_traces.sh* script to control which traces are merged and where the resulting merged trace is saved.

- -o <outputdir> - output directory where the results are aggregated.
- <inputdir>... - space-separated list of rocprofiler directories. If not specified, CWD is used.

The file *'unified/results.json'* is generated, and the resulting unified/results.json file contains trace data from both MPI ranks.

Known issue

Collecting several counter collection passes (multiple "pmc:" lines in an counter input file) is not supported in a single run.

The workaround is to break the multiline counter input file into multiple single-line counter input files and execute runs.

Known Issues in This Release

The following are the known issues in this release.

UPGRADE TO AMD ROCM V4.3 NOT SUPPORTED

An upgrade from previous releases to AMD ROCm v4.3 is not supported. Complete uninstallation of previous ROCm versions is required before installing a new version of ROCm.

*_LAUNCH BOUNDS_*IGNORED DURING KERNEL LAUNCH

The HIP runtime returns the *hipErrorLaunchFailure* error code when an application tries to launch kernel with a block size larger than the launch bounds mentioned during compile time. If no launch bounds were specified during the compile time, the default value of 1024 is assumed. Refer to the HIP trace for more information about the failing kernel. A sample error in the trace is shown below:

Snippet of the HIP trace

```
:3:devprogram.cpp           :2504: 2227377746776 us: Using Code Object V4.
:3:hip_module.cpp           :361 : 2227377768546 us: 7670 : [7f7c6eddd180]
ihipModuleLaunchKernel ( 0x0x16fe080, 2048, 1, 1, 1024, 1, 1, 0, stream:<null>,
0x7ffded8ad260, char array:<null>, event:0, event:0, 0, 0 )
:1:hip_module.cpp           :254 : 2227377768572 us: Launch params (1024, 1, 1)
are larger than launch bounds (64) for kernel _Z8MyKerneliPd
:3:hip_platform.cpp         :667 : 2227377768577 us: 7670 : [7f7c6eddd180]
ihipLaunchKernel: Returned hipErrorLaunchFailure :
:3:hip_module.cpp           :493 : 2227377768581 us: 7670 : [7f7c6eddd180]
hipLaunchKernel: Returned hipErrorLaunchFailure :
```

There is no known workaround at this time.

PYCACHE FOLDER EXISTS AFTER ROCM SMI LIBRARY UNINSTALLATION

Users may observe that the */opt/rocm-x/bin/__pycache__* folder continues to exist even after the *rocm_smi_lib* uninstallation.

Workaround: Delete the */opt/rocm-x/bin/__pycache__* folder manually before uninstalling *rocm_smi_lib*.

Hardware and Software Support

HARDWARE SUPPORT

ROCm is focused on using AMD GPUs to accelerate computational tasks such as machine learning, engineering workloads, and scientific computing. To focus our development efforts on these domains of interest, ROCm supports the following targeted set of hardware configurations.

Supported Graphics Processing Units

As the AMD ROCm platform has a focus on specific computational domains, AMD offers official support for a selection of GPUs that are designed to offer good performance and price in these domains.

Note: The integrated GPUs of Ryzen are not officially supported targets for ROCm.

ROCm officially supports AMD GPUs that use the following chips:

- GFX9 GPUs
  - "Vega 10" chips, such as on the AMD Radeon RX Vega 64 and Radeon Instinct MI25
  - "Vega 7nm" chips, such as on the Radeon Instinct MI50, Radeon Instinct MI60, AMD Radeon VII, Radeon Pro VII
- CDNA GPUs
  - MI100 chips such as on the AMD Instinct™ MI100

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 is enabled in the ROCm software. However, full support is not guaranteed:

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

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

The integrated GPUs in AMD APUs are not officially supported targets for ROCm. As described below, "Carrizo", "Bristol Ridge", and "Raven Ridge" APUs are enabled in AMD upstream drivers and the ROCm

OpenCL runtime. However, they are not enabled in the HIP runtime, and may not work due to motherboard or OEM hardware limitations. Note, they are not yet officially supported targets for ROCm.

GFX8 GPUS

Note: The GPUs require a host CPU and platform with PCIe 3.0 with support for PCIe atomics.

GFX8 GPUs			
Fiji (AMD)	**Polaris 10 (AMD)**	**Polaris 11 (AMD)**	**Polaris 12 (Lexa) (AMD)**
• **Radeon R9 Fury** • **Radeon R9 Nano** • **Radeon R9 Fury X** • **Radeon Pro Duo (Fiji)** • **FirePro S9300 X2** • **Radeon Instinct MI8**	• Radeon RX 470 • Radeon RX 480 • Radeon RX 570 • Radeon RX 580 • Radeon Pro Duo (Polaris) • Radeon Pro WX 5100 • Radeon Pro WX 7100 • Radeon Instinct MI6	• Radeon RX 460 • Radeon RX 560 • Radeon Pro WX 4100	• Radeon RX 540 • Radeon RX 550 • Radeon Pro WX 2100 • Radeon Pro WX 3100

GFX9 GPUS

ROCm offers support for two chips from AMD's most recent "gfx9" generation of GPUs.

GFX9 GPUs	
Vega 10 (AMD)	**Vega 7nm (AMD)**
• **Radeon RX Vega 56** • **Radeon RX Vega 64** • **Radeon Vega Frontier Edition** • **Radeon Pro WX 8200**	• Radeon VII • Radeon Instinct MI50 • Radeon Instinct MI60 •

• **Radeon Pro WX 9100** • **Radeon Pro V340** • **Radeon Pro V340 MxGPU** • **Radeon Instinct MI25** **Note:** ROCm does not support Radeon Pro SSG.	

•

SUPPORTED CPUS

As described above, GFX8 GPUs require PCIe 3.0 with PCIe atomics 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.

The current CPUs which support PCIe Gen3 + PCIe Atomics are:

- AMD Ryzen CPUs
- 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 made more options available for many 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, AMD recommends that you use a CPU from the list provided above for compatibility purposes.

NOT SUPPORTED OR LIMITED SUPPORT UNDER ROCM

LIMITED SUPPORT

- ROCm 4.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.

- 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 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 configuration parameters such as the 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 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 configuration parameters such as the 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.

NOT SUPPORTED

- "Tonga", "Iceland", "Vega M", and "Vega 12" GPUs are not supported.
- AMD does not support GFX8-class GPUs (Fiji, Polaris, etc.) on CPUs that do not have PCIe3.0 with PCIe atomics.
    - AMD Carrizo and Kaveri APUs as hosts for such GPUs are not supported
    - Thunderbolt 1 and 2 enabled GPUs are not supported by GFX8 GPUs on ROCm. Thunderbolt 1 & 2 are based on PCIe 2.0.

In the default ROCm configuration, GFX8 and GFX9 GPUs require PCI Express 3.0 with PCIe atomics. The ROCm platform leverages these advanced capabilities to allow features such as user-level submission of work from the host to the GPU. This includes PCIe atomic Fetch and Add, Compare and Swap, Unconditional Swap, and AtomicOp Completion.

Current CPUs which support PCIe 3.0 + PCIe Atomics:

AMD	INTEL
Ryzen CPUs (Family 17h Model 01h-0Fh) • **Ryzen 3 1300X** • **Ryzen 3 2300X** • **Ryzen 5 1600X** • **Ryzen 5 2600X** • **Ryzen 7 1800X** • **Ryzen 7 2700X**	Intel Core i3, i5, and i7 CPUs from Haswell and beyond. This includes: • Haswell CPUs such as the Core i7 4790K • Broadwell CPUs such as the Core i7 5775C • Skylake CPUs such as the Core i7 6700K • Kaby Lake CPUs such as the Core i7 7740X • Coffee Lake CPUs such as the Core i7 8700K • Xeon CPUs from “v3” and newer • Some models of “Ivy Bridge-E” processors
Ryzen APUs (Family 17h Model 10h-1Fh – previously code-named Raven Ridge) such as:	

• **Athlon 200GE** • **Ryzen 5 2400G** **Note:** The integrated GPU in these devices is not guaranteed to work with ROCm.	
Ryzen Threadripper Workstation CPUs (Family 17h Model 01h-0Fh) such as: • **Ryzen Threadripper 1950X** • **Ryzen Threadripper 2990WX**	
EPYC Server CPUs (Family 17h Model 01h-0Fh) such as: • **Epyc 7551P** • **Epyc 7601**	

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 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 Linux distributions or versions	Features and hardware support varies depending on the kernel version
	Not currently supported on kernels newer than 5.4	Limits GPU's usage of system memory to 3/8 of system memory (before 5.6). For 5.6 and beyond, both DKMS and upstream kernels allow the use of 15/16 of system memory.
		IPC and RDMA capabilities are not yet enabled
		Not tested by AMD to the same level as rock-dkms package
		Does not include the most up-to-date firmware