RVS Documentation

Release 1.1.0

Advanced Micro Devices, Inc.

Aug 08, 2025

INSTALL

The ROCm Validation Suite (RVS) is a system validation and diagnostics tool for monitoring, stress testing, detecting, and troubleshooting issues that affect the functionality and performance of AMD GPUs operating in a high-performance/AI/ML computing environment. RVS is enabled using the ROCm software stack on a compatible software and hardware platform.

RVS is a collection of tests, benchmarks, and qualification tools, each targeting a specific subsystem of the ROCm platform. All of the tools are implemented in software and share a common command-line interface. Each test set is implemented in a "module", which is a library encapsulating the functionality specific to the tool. The CLI can specify the directory containing modules when searching for libraries to load. Each module may have a set of options that it defines and a configuration file that supports its execution.

For more information, refer to GitHub.

Install

- *ROCm Validation Suite installation*

How to

- *Configure ROCm Validation Suite*

Conceptual

- *ROCm Validation Suite modules*

To contribute to the documentation, refer to Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

INSTALLING ROCM VALIDATION SUITE

You can obtain ROCm Validation Suite (RVS) by building it from:

- the source code base
- a pre-built package

1.1 Building from source code

RVS has been developed as an open source solution. You can find the source code and related documentation here.

1.2 Package manager installation

Based on the OS, use the appropriate package manager to install the rocm-validation-suite package.

For more details, refer to ROCm Validation Suite GitHub site.

RVS package components are installed in */opt/rocm*. The package contains:

- executable binary (located in _install-**base**_/bin/rvs)
- public shared libraries (located in _install-**base**_/lib)
- module specific shared libraries (located in _install-**base**_/lib/rvs)
- default configuration files (located in _install-**base**_/share/rocm-validation-suite/conf)
- GPU specific configuration files (located in _install-**base**_/share/rocm-validation-suite/conf/<GPU folder>)
- testscripts (located in _install-**base**_/share/rocm-validation-suite/testscripts)
- user guide (located in _install-**base**_/share/rocm-validation-suite/userguide)
- man page (located in _install-**base**_/share/man)

1.3 Prerequisites

Ensure you review the following prerequisites carefully for each operating system before compiling/installing the ROCm Validation Suite (RVS) package.

Ubuntu

```
sudo apt-get -y update && sudo apt-get install -y libpci3 libpci-dev doxygen␣
↪unzip cmake git libyaml-cpp-dev
```

RHEL

```
sudo yum install -y cmake3 doxygen rpm rpm-build git gcc-c++ yaml-cpp-devel

wget http://mirror.centos.org/centos/7/os/x86_64/Packages/pciutils-devel-3.5.1-3.el7.x86_
↪64.rpm

sudo rpm -ivh pciutils-devel-3.5.1-3.el7.x86_64.rpm
```

SLES

```
sudo zypper  install -y cmake doxygen pciutils-devel libpci3 rpm git rpm-build gcc-c++␣
↪yaml-cpp-devel
```

CentOS

```
sudo yum install -y cmake3 doxygen pciutils-devel rpm rpm-build git gcc-c++ yaml-cpp-
↪devel
```

1.4 Install ROCm stack, rocBLAS, and ROCm-SMI-lib

1. Install the ROCm software tack for Ubuntu/CentOS/SLES/RHEL. Refer to the ROCm installation guide for more details.

> **Note**
>
> The rocm_smi64 package has been renamed to rocm-smi-lib64 from >= ROCm3.0. If you are using ROCm release < 3.0 , install the package as "rocm_smi64". The *rocm-smi-lib64* package has been renamed to rocm-smi-lib from >= ROCm4.1.

2. Install rocBLAS and rocm-smi-lib.

Ubuntu

```
sudo apt-get install rocblas rocm-smi-lib
```

CentOS and RHEL

```
sudo yum install --nogpgcheck rocblas rocm-smi-lib
```

SUSE

```
sudo zypper install rocblas rocm-smi-lib
```

If rocm-smi-lib is already installed but */opt/rocm/lib/librocm_smi64.so* doesn't exist, perform the following steps:

Ubuntu

```
sudo dpkg -r rocm-smi-lib && sudo apt install rocm-smi-lib
```

CentOS and RHEL

```
sudo rpm -e  rocm-smi-lib && sudo yum install  rocm-smi-lib
```

SUSE

```
sudo rpm -e  rocm-smi-lib && sudo zypper install  rocm-smi-lib
```

1.5 Building from source

This section explains how to get and compile current development stream of RVS.

1. Clone the repository.

```
git clone https://github.com/ROCm/ROCmValidationSuite.git
```

2. Use the following instruction to configure.

```
cd ROCmValidationSuite
cmake -B ./build -DROCM_PATH=<rocm_installed_path> -DCMAKE_INSTALL_PREFIX=<rocm_
↪installed_path> -DCPACK_PACKAGING_INSTALL_PREFIX=<rocm_installed_path>
```

For example, if ROCm 5.5 was installed, use the following instruction,

```
cmake -B ./build -DROCM_PATH=/opt/rocm-5.5.0 -DCMAKE_INSTALL_PREFIX=/opt/rocm-5.5.0 -
↪DCPACK_PACKAGING_INSTALL_PREFIX=/opt/rocm-5.5.0
```

3. Build the binary.

```
make -C ./build
```

4. Build the package.

```
cd ./build
make package
```

Based on your OS, only DEB or RPM package will be built.

You may ignore an error for unrelated configurations.

5. Install the built package.

Ubuntu

```
sudo dpkg -i rocm-validation-suite*.deb
```

CentOS, RHEL, and SUSE

```
sudo rpm -i --replacefiles --nodeps rocm-validation-suite*.rpm
```

RVS is getting packaged as part of ROCm release starting from 3.0. You can install the pre-compiled package as indicated below. Ensure Prerequisites, ROCm stack, rocblas and rocm-smi-lib64 are already installed.

6. Install package packaged with ROCm release.

Ubuntu

```
sudo apt install rocm-validation-suite
```

CentOS and RHEL

```
sudo yum install rocm-validation-suite
```

SUSE

```
sudo zypper install rocm-validation-suite
```

1.6 Reporting

Test results, errors, and verbose logs are printed as terminal output. To enable JSON logging use "-j" command line option. The json output file is stored in /var/tmp folder and the name of the file will be printed.

You can build RVS from the source code base or by installing from a pre-built package. See the preceding sections for more details.

1.7 Running RVS

1.7.1 Run version built from source code

```
cd <source folder>/build/bin

Command examples
./rvs --help ; Lists all options to run RVS test suite
./rvs -g ; Lists supported GPUs available in the machine
./rvs -d 3 ; Run set of RVS default sanity tests (in rvs.conf) with verbose level 3
./rvs -c conf/gst_single.conf ; Run GST module default test configuration
```

1.7.2 Run version pre-compiled and packaged with ROCm release

```
cd /opt/rocm/bin

Command examples
./rvs --help ; Lists all options to run RVS test suite
./rvs -g ; Lists supported GPUs available in the machine
./rvs -d 3 ; Run set of RVS sanity tests (in rvs.conf) with verbose level 3
./rvs -c ../share/rocm-validation-suite/conf/gst_single.conf ; Run GST default test␣
↪configuration
```

To run GPU specific test configuration, use configuration files from GPU folders in "/opt/rocm/share/rocm-validation-suite/conf"

```
./rvs -c ../share/rocm-validation-suite/conf/MI300X/gst_single.conf ; Run MI300X␣
↪specific GST test configuration
./rvs -c ../share/rocm-validation-suite/conf/nv32/gst_single.conf ; Run Navi 32 specific␣
↪GST test configuration
```

Note: If present, always use GPU specific configurations instead of default test configurations.

1.8 Building documentation

Run the steps below to build documentation locally.

```
cd docs

pip3 install -r .sphinx/requirements.txt

python3 -m sphinx -T -E -b html -d _build/doctrees -D language=en . _build/html
```

CHAPTER

TWO

CONFIGURATION FILES

The ROCm Validation Suite (RVS) tool allows users to indicate a configuration file, adhering to the YAML 1.2 specification, which details the validation tests to run and the expected results of a test, benchmark or configuration check.

The configuration file used for an execution is specified using the `--config` option. The default configuration file used for a run is `rvs.conf`, which will include default values for all defined tests, benchmarks and configurations checks, as well as device specific configuration values. The format of the configuration files determines the order in which actions are executed, and can provide the number of times the test will be executed as well.

Configuration file is, in YAML terms, mapping of 'actions' keyword into sequence of action items. Action items are themselves YAML keyed lists. Each list consists of several *key:value* pairs. Some keys may have values which are keyed lists themselves (nested mappings).

Action item (or action for short) uses keys to define nature of validation test to be performed. Each action has some common keys – like 'name', 'module', 'deviceid' – and test specific keys which depend on the module being used.

An example of a RVS configuration file is given here:

```
actions:
- name: action_1
  device: all
  module: gpup
  properties:
    mem_banks_count:
  io_links-properties:
    version_major:
- name: action_2
  module: gpup
  device: all
  properties:
    mem_banks_count:
- name: action_3
...
```

2.1 Common Configuration Keys

Common configuration keys applicable to most modules are summarized in the following table.

Config Key	Type	Description
name	String	The name of the defined action.
device	Collection of String	This is a list of device indexes (gpu ids), or the keyword “all”. The defined actions will be executed on the specified device, as long as the action targets a device specifically (some are platform actions). If an invalid device id value or no value is specified the tool will report that the device was not found and terminate execution, returning an error regarding the configuration file.
deviceid	Integer	This is an optional parameter, but if specified it restricts the action to a specific device type corresponding to the deviceid.
parallel	Bool	If this key is false, actions will be run on one device at a time, in the order specified in the device list, or the natural ordering if the device value is “all”. If this parameter is true, actions will be run on all specified devices in parallel. If a value isn’t specified the default value is false.
count	Integer	This specifies number of times to execute the action. If the value is 0, execution will continue indefinitely. If a value isn’t specified the default is 1. Some modules will ignore this parameter.
wait	Integer	This indicates how long the test should wait between executions, in milliseconds. Some modules will ignore this parameter. If the count key is not specified, this key is ignored. duration Integer This parameter overrides the count key, if specified. This indicates how long the test should run, given in milliseconds. Some modules will ignore this parameter.

2.2 Command Line Options

Command line options are summarized in the table below:

Short option	Long option	Description
-a	--appe	When generating a debug logfile, do not overwrite the content of the current log. Use in conjunction with -d and -l options.
-c	--conf	Specify the test configuration file to use. This is a mandatory field for test execution.
-d	--debu	Specify the debug level for the output log. The range is 0 to 5, with 5 being the highest verbose level.
-g	--list	List all the GPUs available in the machine, that RVS supports and has visibility.
-i	--inde	Comma-separated list of GPU IDs or indexes to run test on. This overrides the device/ device_index parameter values specified for every action in the configuration file, including the all value.
-j	--json	Generate output file in JSON format. If a path follows this argument, it will be used as a json log file. Otherwise, a file will be created in /var/tmp/ with a timestamp in the file name.
-l	--debu	Generate the log file with output and debug information.
-t	--list	List the test modules present in RVS.
-v	--verb	Enable detailed logging. Equivalent to specifying -d 5 option.
-p	--para	Enables or disables parallel execution across multiple GPUs. Use this option in conjunction with the -c option. Accepted Values: true: Enables parallel execution. false: Disables parallel execution. If no value is provided for the option, it defaults to true.
-n	--numT	Number of times the test repeatedly executes. Use this option in conjunction with the -c option.
	--quie	No console output given. See logs and return code for errors.
	--vers	Display the version information.
-h	--help	Display usage information.

CHAPTER

THREE

ROCM VALIDATION SUITE MODULES

ROCm Validation Suite (RVS) is implemented as a set of modules each implementing a particular test functionality. Modules are invoked from one central place (aka Launcher), which is responsible for reading input (command line and test configuration file), loading and running appropriate modules and providing test output. RVS architecture is built around concept of Linux shared objects, thus allowing for easy addition of new modules in the future.

3.1 GPU Properties (GPUP)

The GPU Properties module queries the configuration of a target device and returns the device's static characteristics. These static values can be used to debug issues such as device support, performance and firmware problems.

3.2 GPU Monitor (GM module) [deprecated]

The GPU monitor tool is capable of running on one, some or all of the GPU(s) installed and will report various information at regular intervals. The module can be configured to halt another RVS modules execution if one of the quantities exceeds a specified boundary value.

3.3 PCI Express State Monitor (PESM module) [deprecated]

The PCIe State Monitor tool is used to actively monitor the PCIe interconnect between the host platform and the GPU. The module will register a "listener" on a target GPU's PCIe interconnect, and log a message whenever it detects a state change. The PESM will be able to detect the following state changes:

1. PCIe link speed changes
2. GPU power state changes

3.4 ROCm Configuration Qualification Tool (RCQT module)

The ROCm Configuration Qualification Tool ensures the platform is capable of running ROCm applications and is configured correctly. It checks the installed versions of the ROCm components and the platform configuration of the system. This includes checking the dependencies corresponding to the ROCm meta-packages are installed correctly.

3.5 PCI Express Qualification Tool (PEQT module)

The PCIe Qualification Tool is used to qualify the PCIe bus on which the GPU is connected. The qualification test will be capable of determining the following characteristics of the PCIe bus interconnect to a GPU:

1. Support for Gen 3 atomic completers
2. DMA transfer statistics

3. PCIe link speed
4. PCIe link width

3.6 SBIOS Mapping Qualification Tool (SMQT module) [deprecated]

The GPU SBIOS mapping qualification tool is designed to verify that a platform's SBIOS has satisfied the BAR mapping requirements for VDI and Radeon Instinct products for ROCm support.

3.7 P2P Benchmark and Qualification Tool (PBQT module)

The P2P Benchmark and Qualification Tool is designed to provide the list of all GPUs that support P2P and characterize the P2P links between peers. In addition to testing P2P compatibility, this test will perform a peer-to-peer throughput test between all P2P pairs for performance evaluation. The P2P Benchmark and Qualification Tool will allow users to pick a collection of two or more GPUs to run the test. The user will also be able to select whether or not they want to run the throughput test on each of the pairs.

3.8 PCI Express Bandwidth Benchmark (PEBB module)

The PCIe Bandwidth Benchmark attempts to saturate the PCIe bus with DMA transfers between system memory and a target GPU card's memory. The maximum bandwidth obtained is reported to help debug low bandwidth issues. The benchmark should be capable of targeting one, some or all of the GPUs installed in a platform, reporting individual benchmark statistics for each.

3.9 GPU Stress test (GST module)

The GPU Stress Test runs various GEMM operations as workloads to stress the GPU FLOPS performance. GEMM operations include SGEMM, DGEMM and HGEMM (Single/Double/Half-precision General Matrix Multiplication) operations based on configured parameters. The duration of the test is configurable, both in terms of time (how long to run) and iterations (how many times to run).

3.10 Input EDPp test (IET module)

The Input EDPp Test runs GEMM workloads to stress the GPU power (i.e. TGP). This test is used to verify if the GPU is capable of handling max. power stress for a sustained period of time. Also checks whether GPU power reaches a set target power.

3.11 Memory test (MEM module)

The Memory module tests the GPU memory for hard and soft errors using HIP. It consists of various tests that use algorithms like Walking 1 bit, Moving inversion and Modulo 20. The module executes the following memory tests [Algorithm, data pattern]

1. Walking 1 bit
2. Own address test
3. Moving inversions, ones & zeros
4. Moving inversions, 8 bit pattern
5. Moving inversions, random pattern

6. Block move, 64 moves
7. Moving inversions, 32 bit pattern
8. Random number sequence
9. Modulo 20, random pattern
10. Memory stress test

3.12 BABEL benchmark test (BABEL module)

The Babel module executes BabelStream (synthetic GPU benchmark based on the original STREAM benchmark for CPUs) benchmark that measures memory transfer rates (bandwidth) to and from global device memory. Various benchmark tests are implemented using GPU kernels in HIP (Heterogeneous Interface for Portability) programming language.

CHAPTER

FOUR

LICENSE