NVIDIA VALIDATION SUITE

v2.0 | June 2020

Best Practices and User Guide

TABLE OF CONTENTS

Chapter 1. OVERVIEW

The NVIDIA Validation Suite (NVVS) is the system administrator and cluster manager's tool for detecting and troubleshooting common problems affecting NVIDIA® Tesla™ GPUs in a high performance computing environments. NVVS focuses on software and system configuration issues, diagnostics, topological concerns, and relative performance.

1.1. NVVS Goals

The NVIDIA Validation Suite is designed to:

1. Provide a system-level tool, in production environments, to assess cluster readiness levels before a workload is deployed.
2. Facilitate multiple run modes:
   - Interactive via an administrator or user in plain text.
   - Scripted via another tool with easily parseable output.
3. Provide multiple test timeframes to facilitate different preparedness or failure conditions:
   - Quick tests to use as a readiness metric
   - Medium tests to use as an epilogue on failure
   - Long tests to be run by an administrator as post-mortem
4. Integrate the following concepts into a single tool to discover deployment, system software and hardware configuration issues, basic diagnostics, integration issues, and relative system performance.
   - Deployment and Software Issues
     - NVML library access and versioning
     - CUDA library access and versioning
     - Software conflicts
   - Hardware Issues and Diagnostics
     - Pending Page Retirements
     - PCIe interface checks
     - NVLink interface checks

    - Framebuffer and memory checks
    - Compute engine checks
  - Integration Issues
    - PCIe replay counter checks
    - Topological limitations
    - Permissions, driver, and cgroups checks
    - Basic power and thermal constraint checks
  - Stress Checks
    - Power and thermal stress
    - Throughput stress
    - Constant relative system performance
    - Maximum relative system performance
    - Memory Bandwidth
5. Provide troubleshooting help
6. Easily integrate into *Cluster Scheduler* and *Cluster Management* applications
7. Reduce downtime and failed GPU jobs

1.2. Beyond the Scope of the NVIDIA Validation Suite

NVVS is not designed to:

1. Provide comprehensive hardware diagnostics
2. Actively fix problems
3. Replace the field diagnosis tools. Please refer to http://docs.nvidia.com/deploy/hw-field-diag/index.html for that process.
4. Facilitate any RMA process. Please refer to http://docs.nvidia.com/deploy/rma-process/index.html for those procedures.

1.3. Dependencies

- NVVS requires a NVIDIA Linux driver to be installed. Both the standard display driver and Tesla Recommended Driver will work. You can obtain a driver from http://www.nvidia.com/object/unix.html.
- NVVS requires the standard C++ runtime library with GLIBCXX of at least version 3.4.5 or greater.

1.4. Supported Products

The NVIDIA Validation Suite supports Tesla GPUs running on 64-bit Linux (bare metal) operating systems. NVIDIA® Tesla™ Line:

- All Kepler, Maxwell, Pascal, and Volta architecture GPUs

Chapter 2. USING NVVS

The various command line options of NVVS are designed to control general execution parameters, whereas detailed changes to execution behavior are contained within the configuration files detailed in the next chapter.

2.1. Command line options

The various options for NVVS are as follows:

Short option	Long option	Description
	--statspath	Write the plugin statistics to a given path rather than the current directory.
-a	*--appendLog*	When generating a debug logfile, do not overwrite the contents of a current log. Used in conjuction with the -d and -l options.
-c	*--config*	Specify the configuration file to be used. The default is **`/etc/nvidia-validation-suite/nvvs.conf`**
	--configless	Run NVVS in a configless mode. Executes a "long" test on all supported GPUs.
-d	*--debugLevel*	Specify the debug level for the output log. The range is 0 to 5 with 5 being the most verbose. Used in conjunction with the -l flag.
-g	*--listGpus*	List the GPUs available and exit. This will only list GPUs that are supported by NVVS.
-i	*--indexes*	Comma separated list of indexes to run NVVS on.

Short option	Long option	Description
-j	*--jsonOutput*	Instructs nvvs to format the output as JSON.
-l	*--debugLogFile*	Specify the logfile for debug information. This will produce an encrypted log file intended to be returned to NVIDIA for post-run analysis after an error.
	--quiet	No console output given. See logs and return code for errors.
-p	*--pluginpath*	Specify a custom path for the NVVS plugins.
-s	*--scriptable*	Produce output in a colon-separated, more script-friendly and parseable format.
	--specifiedtest	Run a specific test in a configless mode. Multiple word tests should be in quotes, and if more than one test is specified it should be comma-separated.
	--parameters	Specify test parameters via the command-line. For example: --parameters "sm stress.test_duration=300" would set the test duration for the SM Stress test to 300 seconds.
	--statsonfail	Output statistic logs only if a test failure is encountered.
-t	*--listTests*	List the tests available to be executed through NVVS and exit. This will list only the readily loadable tests given the current path and library conditions.
-v	*--verbose*	Enable verbose reporting.
	--version	Displays the version information and exits.
-h	*--help*	Display usage information and exit.

2.2. Usage Examples

To display the list of GPUs available on the system.

```
user@hostname
$ nvvs -g

NVIDIA Validation Suite (version 352.00)

Supported GPUs available:
 [0000:01:00.0] -- Tesla K40c
 [0000:05:00.0] -- Tesla K20c
```

```
 [0000:06:00.0] -- Tesla K20c
```

An example "quick" test (explained later) using a custom configuration file.

```
user@hostname
$ nvvs -c Tesla_K40c_quick.conf

NVIDIA Validation Suite (version 352.00)

 Software
  Blacklist ....................................... PASS
  NVML Library .................................... PASS
  CUDA Main Library ............................... PASS
  CUDA Toolkit Libraries .......................... PASS
  Permissions and OS-related Blocks ............... PASS
  Persistence Mode ................................ PASS
  Environmental Variables ......................... PASS
```

To output an encrypted debug file at the highest debug level to send to NVIDIA for analysis after a problem.

```
user@hostname
$ nvvs -c Tesla_K40c_medium.conf -d 5 -l debug.log

NVIDIA Validation Suite (version 352.00)

 Software
  Blacklist ....................................... PASS
  NVML Library .................................... PASS
  CUDA Main Library ............................... PASS
  CUDA Toolkit Libraries .......................... PASS
  Permissions and OS-related Blocks ............... PASS
  Persistence Mode ................................ PASS
  Environmental Variables ......................... PASS
 Hardware
  Memory GPU0 ..................................... PASS
 Integration
  PCIe ............................................ FAIL
           *** GPU 0 is running at PCI link width 8X, which is below the minimum
 allowed link width of 16X (parameter:
min_pci_width)"
```

The output file, debug.log would then be returned to NVIDIA.

2.3. Configuration file

The NVVS configuration file is a YAML-formatted (e.g. human-readable JSON) text file with three main stanzas controlling the various tests and their execution.

The general format of a configuration file consists of:

```
%YAML 1.2
---

globals:
  key1: value
  key2: value

test_suite_name:
```

```
- test_class_name1:
    test_name1:
      key1: value
      key2: value
      subtests:
        subtest_name1:
          key1: value
          key2: value
    test_name2:
      key1: value
      key2: value
-test_class_name2:
    test_name3:
      key1: value
      key2: value

gpus:
- gpuset: name
  properties:
    key1: value
    key2: value
  tests:
    name: test_suite_name
```

There are three distinct sections: *globals, test_suite_name, and gpus* each with its own subsection of parameters and as is with any YAML document, **indentation is important** thus if errors are generated from your own configuration files please refer to this example for indentation reference.

2.4. Global parameters

Keyword	Value Type	Description
logfile	String	The prefix for all detailed test data able to be used for post-processing.
logfile_type	String	Can be *json*, *text*, or *binary*. Used in conjunction with the logfile global parameter. Default is JSON.
scriptable	Boolean	Accepts *true*, or *false*. Produces a script-friendly, colon-separated output and is identical to the *-s* command line parameter.
serial_override	Boolean	Accepts *true*, or *false*. Some tests are designed to run in parallel if multiple GPUs are given. This parameter overrides that behavior serializing execution across all tests.
require_persistence_mode	Boolean	Accepts *true*, or *false*. Persistence mode is a prerequisite for some tests, this global overrides that requirement and should only be used if it is not possible to activate persistence mode on your system.

2.5. GPU parameters

The *gpus* stanza may consist of one or more *gpusets* which will each match zero or more GPUs on the system based on their *properties*(a match of zero will produce an error).

GPUs are matched based on the following criteria with their configuration file keywords in parenthesis:

- Name of the GPU, i.e. Tesla K40c (*name*)
- Brand of the GPU, i.e. Tesla (*brand*)
- A comma separated list of indexes (*index*)
- The GPU UUID (*uuid*)
- or the PCIe Bus ID (*busid*)

The matching rules are based off of exclusion. First, the list of supported GPUs is taken and if no *properties* tag is given then all GPUs will be used in the test. Because a UUID or PCIe Bus ID can only match a single GPU, if those properties are given then only that GPU will be used if found. The remaining properties, *index*, *brand*, and *name* work in an "AND" fashion such that, if specified, the result must match at least one GPU on the system for a test to be performed.

For example, if *name* is set to "Tesla K40c" and *index* is set to "0" NVVS will error if index 0 is not a Tesla K40c. By specifying both *brand* and *index* a user may limit a test to specific "Tesla" cards for example. **In this version of NVVS, all matching GPUs must be homogeneous.**

The second identifier for a *gpuset* is *tests*. This parameter specifies either the suite of tests that a user wishes to run or the test itself.

At present the following suites are available:

- *Quick* -- meant as a pre-run sanity check to ensure that the GPUs are ready for a job. Currently runs the Deployment tests described in the next chapter.
- *Medium* -- meant as a quick, post-error check to make sure that nothing very obvious such as ECC enablement or double-bit errors have occurred on a GPU. Currently runs the Deployment, Memory/Hardware, and PCIe/Bandwidth tests. The Hardware tests are meant to be relatively short to find obvious issues.
- *Long* -- meant as a more extensive check to find potential power and/or performance problems within a cluster. Currently runs an extensive test that involves Deployment, Memory/Hardware, PCI/Bandwidth, Power, Stress, and Memory Bandwidth. The Hardware tests will run in a longer-term iterative mode that are meant to try and capture transient failures as well as obvious issues.

An individual test can also be specified. Currently the keywords are: *Memory*, *Diagnostic*, *Targeted Stress*, *Targeted Power*, *PCIe*, *SM Stress*, and *Memory Bandwidth*. Please see the "custom" section in the next subchapter to configure and tweak the parameters when this method is used.

2.6. Test Parameters

The format of the NVVS configuration file is designed for extensibility. Each test suite above can be customized in a number of ways described in detail in the following chapter for each test. Individual tests belong to a specific class of functionality which, when wanting to customize specific parameters, must also be specified.

The classes and the respective tests they perform are as follows:

Class name	Tests	Brief description
Software	Deployment	Checks for various runtime libraries, persistence mode, permissions, environmental variables, and blacklisted drivers.
Hardware	Diagnostic	Execute a series of hardware diagnostics meant to exercise a GPU or GPUs to their factory specified limits.
Integration	PCIe	Test host to GPU, GPU to host, and P2P (if possible) bandwidth. P2P between GPUs occurs over NvLink (if possible) or PCIe.
Stress	Targeted Stress	Sustain a specific targeted stress level for a given amount of time.
	Targeted Power	Sustain a specific targeted power level for a given amount of time.
	SM Stress	Sustain a workload on the Streaming Multiprocessors (SMs) of the GPU for a given amount of time.
	Memory Bandwidth	Verify that a certain memory bandwidth can be achieved on the framebuffer of the GPU.

Some tests also have subtests that can be enabled by using the *subtests* keyword and then hierarchically adding the subtest parameters desired beneath. An example would be the PCIe Bandwidth test which may have a section that looks similar to this:

```
long:
- integration:
    pcie:
      test_unpinned: false
      subtests:
        h2d_d2h_single_pinned:
          min_bandwidth: 20
          min_pci_width: 16
```

When only a specific test is given in the GPU set portion of the configuration file, both the suite and class of the test are *custom*. For example:

```
%YAML 1.2
---

globals:
  logfile: nvvs.log

custom:
- custom:
    targeted stress:
      test_duration: 60

gpus:
- gpuset: all_K40c
  properties:
    name: Tesla K40c
  tests:
    - name: targeted stress
```

Chapter 3. OVERVIEW OF PLUGINS

The NVIDIA Validation Suite consists of a series of plugins that are each designed to accomplish a different goal.

3.1. Deployment Plugin

The deployment plugin's purpose is to verify the compute environment is ready to run Cuda applications and is able to load the NVML library.

Preconditions

- LD_LIBRARY_PATH must include the path to the cuda libraries, which for version X.Y of Cuda is normally /usr/local/cuda-*X.Y*/lib64, which can be set by running *export LD_LIBRARY_PATH=/usr/local/cuda-X.Y/lib64*
- The linux nouveau driver must not be running, and should be blacklisted since it will conflict with the nvidia driver

Configuration Parameters

None at this time.

Stat Outputs

None at this time.

Failure

The plugin will fail if:

- The corresponding device nodes for the target GPU(s) are being blocked by the operating system (e.g. cgroups) or exist without r/w permissions for the current user.
- The NVML library libnvidia-ml.so cannot be loaded
- The Cuda runtime libraries cannot be loaded

- The **nouveau** driver is found to be loaded
- Any pages are pending retirement on the target GPU(s)
- Any other graphics processes are running on the target GPU(s) while the plugin runs

3.2. Hardware Disagnostic Plugin

The HW Diagnostic Plugin is designed to identify HW failures on GPU silicon and board-level components, extending out to the PCIE and NVLINK interfaces. It is not intended to identify HW or system level issues beyond the NVIDIA-provided HW. Nor is it intended to identify SW level issues above the HW, e.g. in the NVIDIA driver stack. The plugin runs a series of tests that target GPU computational correctness, GDDR/HBM memory resiliency, GPU and SRAM high power operation, SM stress and NVLINK/PCIE correctness. The plugin can run with several combinations of tests corresponding to medium and long NVVS operational modes. This plugin will take about three minutes to execute.

The plugin produces a simple pass/fail output. A failing output means that a potential HW issue has been found. However, the NVVS HW Diagnostic Plugin is not by itself a justification for GPU RMA. Any failure in the plugin should be followed by execution of the full NVIDIA Field Diagnostic after the machine has been taken offline. Only a failure of the Field Diagnostic tool constitutes grounds for RMA. Since the NVVS HW Diagnostic Plugin is a functional subset of the Field Diagnostic a failure in the plugin is a strong indicator of a future Field Diagnostic failure.

Preconditions

- No other GPU processes can be running.

Configuration Parameters

Parameter Name	Type	Default	Value Range	Description
test_duration	Float	180.0	30.0 - 3600.0	How long the performance test should run for in seconds. It is recommended to set this to at least 30 seconds to make sure you actually get some stress from the test.
use_doubles	Boolean	False	True or False	If set to true, tells the test to use double-point

Parameter Name	Type	Default	Value Range	Description
				precision in its calculations. By default, it is false and the test will use floating point precision.
temperature_max	Float	100.0	30.0 - 120.0	The maximum temperature in C that the card is allowed to reach during the test. Use nvidia-smi -q to see the normal temperature limits of your device.

Stat Outputs

Stat Name	Stat Scope	Type	Description
power_usage	GPU	Time series Float	Per second power usage of each GPU in watts. Note that for multi-GPU boards, each GPU gets a fraction of the power budget of the board.
graphics_clock	GPU	Time series Float	Per second clock rate of each GPU in MHZ
memory_clock	GPU	Time series Float	Per second clock rate of the GPU's memory in MHZ
nvml_events	GPU	Time series Int64	Any events that were read with nvmlEventSetWait - including single or double bit errors or XID errors - during the test.

Stat Name	Stat Scope	Type	Description
power_violation	GPU	Time series Float	Percentage of time this GPU was violating power constraints.
gpu_temperature	GPU	Time series Float	Per second temperature of the GPU in degrees C
thermal_violation	GPU	Time series Float	Percentage of time this GPU was violating thermal constraints.
perf_gflops	GPU	Time Series Float	The per second reading of average gflops since the test began.

Failure

The plugin will fail if:

- The corresponding device nodes for the target GPU(s) are being blocked by the operating system (e.g. cgroups) or exist without r/w permissions for the current user.
- Other GPU processes are running
- A hardware issue has been detected. *This is not an RMA actionable failure but rather an indication that more investigation is required.*
- The temperature reaches unacceptable levels during the test.
- If GPU double bit ECC errors occur or the configured amount of SBE errors occur.
- If a critical XID occurs

3.3. PCIe - GPU Bandwidth Plugin

The GPU bandwidth plugin's purpose is to measure the bandwidth and latency to and from the GPUs and the host.

Preconditions

None

Sub Tests

The plugin consists of several self-tests that each measure a different aspect of bandwidth or latency. Each subtest has either a pinned/unpinned pair or a p2p enabled/

p2p disabled pair of identical tests. Pinned/unpinned tests use either pinned or unpinned memory when copying data between the host and the GPUs.

This plugin will use NvLink to communicate between GPUs when possible. Otherwise, communication between GPUs will occur over PCIe

Each sub test is represented with a tag that is used both for specifying configuration parameters for the sub test and for outputting stats for the sub test. P2p enabled/p2p disabled tests enable or disable GPUs on the same card talking to each other directly rather than through the PCIe bus.

Sub Test Tag	Pinned/Unpinned P2P Enabled/P2P Disabled	Description
h2d_d2h_single_pinned	Pinned	Device <-> Host Bandwidth, one GPU at a time
h2d_d2h_single_unpinned	Unpinned	Device <-> Host Bandwidth, one GPU at a time
h2d_d2h_concurrent_pinned	Pinned	Device <-> Host Bandwidth, all GPUs concurrently
h2d_d2h_concurrent_unpinned	Unpinned	Device <-> Host Bandwidth, all GPUs concurrently
h2d_d2h_latency_pinned	Pinned	Device <-> Host Latency, one GPU at a time
h2d_d2h_latency_unpinned	Unpinned	Device <-> Host Latency, one GPU at a time
p2p_bw_p2p_enabled	P2P Enabled	Device <-> Device bandwidth one GPU pair at a time
p2p_bw_p2p_disabled	P2P Disabled	Device <-> Device bandwidth one GPU pair at a time
p2p_bw_concurrent_p2p_enabled	P2P Enabled	Device <-> Device bandwidth, concurrently, focusing on bandwidth between GPUs between GPUs likely to be directly connected to each other -> for each (index / 2) and (index / 2)+1
p2p_bw_concurrent_p2p_disabled	P2P Disabled	Device <-> Device bandwidth, concurrently, focusing on bandwidth between GPUs

Sub Test Tag	Pinned/Unpinned P2P Enabled/P2P Disabled	Description
		between GPUs likely to be directly connected to each other -> for each (index / 2) and (index / 2)+1
1d_exch_bw_p2p_enabled	P2P Enabled	Device <-> Device bandwidth, concurrently, focusing on bandwidth between gpus, every GPU either sending to the gpu with the index higher than itself (l2r) or to the gpu with the index lower than itself (r2l)
1d_exch_bw_p2p_disabled	P2P Disabled	Device <-> Device bandwidth, concurrently, focusing on bandwidth between gpus, every GPU either sending to the gpu with the index higher than itself (l2r) or to the gpu with the index lower than itself (r2l)
p2p_latency_p2p_enabled	P2P Enabled	Device <-> Device Latency, one GPU pair at a time
p2p_latency_p2p_disabled	P2P Disabled	Device <-> Device Latency, one GPU pair at a time

Configuration Parameters- Global

Parameter Name	Type	Default	Value Range	Description
test_pinned	Bool	True	True/False	Include subtests that test using pinned memory.
test_unpinned	Bool	True	True/False	Include subtests that test using unpinned memory.
test_p2p_on	Bool	True	True/False	Include subtests that require peer to peer (P2P) memory transfers

Parameter Name	Type	Default	Value Range	Description
				between cards to occur.
test_p2p_off	Bool	True	True/False	Include subtests that do not require peer to peer (P2P) memory transfers between cards to occur.
max_pcie_replays	Float	80.0	1.0 - 1000000.0	Maximum number of PCIe replays to allow per GPU for the duration of this plugin. This is based on an expected replay rate <8 per minute for PCIe Gen 3.0, assuming this plugin will run for less than a minute and allowing 10x as many replays before failure.

Configuration Parameters- Sub Test

Parameter Name	Default (Range)	Affected Sub Tests	Description
min_bandwidth	*Null* *(0.0 - 100.0)*	h2d_d2h_single_pinned, h2d_d2h_single_unpinned, h2d_d2h_concurrent_pinned, h2d_d2h_concurrent_unpinned	Minimum bandwidth in GB/s that must be reached for this sub-test to pass.
max_latency	100,000.0 (0.0 - 1,000,000.0)	h2d_d2h_latency_pinned, h2d_d2h_latency_unpinned	Latency in microseconds that cannot be exceeded for this sub-test to pass.
min_pci_generation	1.0 (1.0 - 3.0)	h2d_d2h_single_pinned, h2d_d2h_single_unpinned	Minimum allowed PCI generation that the GPU must be at or exceed for this sub-test to pass.
min_pci_width	1.0	h2d_d2h_single_pinned, h2d_d2h_single_unpinned	Minimum allowed PCI width that the GPU must be at or exceed

Parameter Name	Default (Range)	Affected Sub Tests	Description
	(1.0 - 16.0)		for this sub-test to pass. For example, 16x = 16.0.

Stat Outputs - Global

Stat Name	Stat Scope	Type	Description
pcie_replay_count	GPU	Float	The per second reading of PCIe replays that have occurred since the start of the GPU Bandwidth plugin.

Stat Outputs -Sub Test

Stats for the GPU Bandwidth test are also output on a test by test basis, using the sub test name as the group name key. The following stats sections are organized by sub test.

h2d_d2h_single_pinned/h2d_d2h_single_unpinned

Stat Name	Type	Description
*N*_h2d	Float	Average bandwidth from host to device for device *N*
*N*_d2h	Float	Average bandwidth from device to host for device *N*
*N*_bidir	Float	Average bandwidth from device to host and host to device at the same time for device *N*

h2d_d2h_concurrent_pinned/h2d_d2h_concurrent_unpinned

Stat Name	Type	Description
*N*_h2d	Float	Average bandwidth from host to device for device *N*
*N*_d2h	Float	Average bandwidth from device to host for device *N*
*N*_bidir	Float	Average bandwidth from device to host and host to device at the same time for device *N*

Stat Name	Type	Description
sum_bidir	Float	Sum of the average bandwidth from device to host and host to device for all devices.
sum_h2d	Float	Sum of the average bandwidth from host to device for all devices.
sum_d2h	Float	Sum of the average bandwidth from device to host for all devices.

h2d_d2h_latency_pinned/h2d_d2h_latency_unpinned

Stat Name	Type	Description
*N*_h2d	Float	Average latency from host to device for device *N*
*N*_d2h	Float	Average latency from device to host for device *N*
*N*_bidir	Float	Average latency from device to host and host to device at the same time for device *N*

p2p_bw_p2p_enabled/p2p_bw_p2p_disabled

Stat Name	Type	Description
*N*_*M*_onedir	Float	Average bandwidth from device *N* to device *M*, copying one direction at a time.
*N*_*M*_bidir	Float	Average bandwidth from device *N* to device *M*, copying both directions at the same time.

p2p_bw_concurrent_p2p_enabled/p2p_bw_concurrent_p2p_disabled

Stat Name	Type	Description
l2r_*N*_*M*	Float	Average bandwidth from device *N* to device *M*
r2l_*N*_*M*	Float	Average bandwidth from device *M* to device *N*

Stat Name	Type	Description
bidir_*N*_*M*	Float	Average bandwidth from device *M* to device *N*, copying concurrently
r2l_sum	Float	Sum of average bandwidth for all right (*M*) to left (*N*) copies
r2l_sum	Float	Sum of average bidirectional bandwidth for all right (*M*) to left (*N*) and left to right copies copies

1d_exch_bw_p2p_enabled/1d_exch_bw_p2p_disabled

Stat Name	Type	Description
l2r_*N*	Float	Average bandwidth from device *N* to device *N+1*
r2l_*N*	Float	Average bandwidth from device *N* to device *N-1*
l2r_sum	Float	Sum of all l2r average bandwidth stats
r2l_sum	Float	Sum of all l2r average bandwidth stats

p2p_latency_p2p_enabled/p2p_latency_p2p_disabled

Stat Name	Type	Description
*N*_*M*	Float	Average latency from device *N* to device *M*

Failure

The plugin will fail if:

- The latency exceeds the configured threshold for relevant tests.
- The bandwidth cannot exceed the configured threshold for relevant tests.
- If the number of PCIe retransmits exceeds a user-provided threshold.

3.4. Memory Bandwidth Plugin

The purpose of the Memory Bandwidth plugin is to validate that the bandwidth of the framebuffer of the GPU is above a preconfigured threshold.

Preconditions

This plugin only runs on GV100 GPUs at this time.

Configuration Parameters

Parameter Name	Type	Default	Value Range	Description
minimum_bandwidth	Float	Differs per GPU	1.0 - 1000000.0	Minimum framebuffer bandwidth threshold that must be achieved in order to pass this test in MB/sec.

Stat Outputs

Stat Name	Stat Scope	Type	Description
power_usage	GPU	Time series Float	Per second power usage of each GPU in watts. Note that for multi-GPU boards, each GPU gets a fraction of the power budget of the board.
memory_clock	GPU	Time series Float	Per second clock rate of the GPU's memory in MHZ
nvml_events	GPU	Time series Int64	Any events that were read with nvmlEventSetWait during the test and the timestamp it was read it.

Failure

The plugin will fail if:

- the minimum bandwidth specified in *minimum_bandwidth* cannot be achieved.
- If GPU double bit ECC errors occur or the configured amount of SBE errors occur.
- If a critical XID occurs

3.5. SM Stress Plugin

The SM performance plugin's purpose is to bring the Streaming Multiprocessors (SMs) of the target GPU(s) to a target performance level in gigaflops by doing large matrix multiplications using cublas. Unlike the Targeted Stress plugin, the SM stress plugin does not copy the source arrays to the GPU before every matrix multiplication. This allows the SM performance plugin's performance to not be capped by device to host bandwidth. The plugin calculates how many matrix operations per second are necessary to achieve the configured performance target and fails if it cannot achieve that target.

This plugin should be used to watch for thermal, power and related anomalies while the target GPU(s) are under realistic load conditions. By setting the appropriate parameters a user can ensure that all GPUs in a node or cluster reach desired performance levels. Further analysis of the generated stats can also show variations in the required power, clocks or temperatures to reach these targets, and thus highlight GPUs or nodes that are operating less efficiently.

Preconditions

None

Configuration Parameters

Parameter Name	Type	Default	Value Range	Description
test_duration	Float	90.0	30.0 - 3600.0	How long the performance test should run for in seconds. It is recommended to set this to at least 30 seconds for performance to stabilize.
temperature_max	Float	*Null*	30.0 - 120.0	The maximum temperature in C the card is allowed to reach during the test. Note that this check is disabled by default. Use nvidia-smi -q to see the normal

Parameter Name	Type	Default	Value Range	Description
				temperature limits of your device.
target_stress	Float	*Null*	SKU dependent	The maximum relative performance each card will attempt to achieve.

Stat Outputs

Stat Name	Stat Scope	Type	Description
power_usage	GPU	Time series Float	Per second power usage of each GPU in watts. Note that for multi-GPU boards, each GPU gets a fraction of the power budget of the board.
graphics_clock	GPU	Time series Float	Per second clock rate of each GPU in MHZ
memory_clock	GPU	Time series Float	Per second clock rate of the GPU's memory in MHZ
nvml_events	GPU	Time series Int64	Any events that were read with nvmlEventSetWait - including single or double bit errors or XID errors - during the test.
power_violation	GPU	Time series Float	Percentage of time this GPU was violating power constraints.
gpu_temperature	GPU	Time series Float	Per second temperature of the GPU in degrees C
perf_gflops	GPU	Time series Float	The per second reading of average

Stat Name	Stat Scope	Type	Description
			gflops since the test began.
flops_per_op	GPU	Float	Flops (floating point operations) per operation queued to the GPU stream. One operation is one call to cublasSgemm or cublasDgemm
bytes_copied_per_op	GPU	Float	How many bytes are copied to + from the GPU per operation
num_cuda_streams	GPU	Float	How many cuda streams were used per gpu to queue operations to the GPUs
try_ops_per_sec	GPU	Float	Calculated number of ops/second necessary to achieve target gigaflops

Failure

The plugin will fail if:

- The GPU temperature exceeds a user-provided threshold.
- If thermal violation counters increase
- If the target performance level cannot be reached
- If GPU double bit ECC errors occur or the configured amount of SBE errors occur.
- If a critical XID occurs

3.6. Targeted Stress Plugin

The Targeted Stress plugin's purpose is to bring the GPU to a target performance level in gigaflops by doing large matrix multiplications using cublas. The plugin calculates how many matrix operations per second are necessary to achieve the configured performance target and fails if it cannot achieve that target.

This plugin should be used to watch for thermal, power and related anomalies while the target GPU(s) are under realistic load conditions. By setting the appropriate parameters

a user can ensure that all GPUs in a node or cluster reach desired performance levels. Further analysis of the generated stats can also show variations in the required power, clocks or temperatures to reach these targets, and thus highlight GPUs or nodes that are operating less efficiently.

Preconditions

None

Configuration Parameters

Parameter Name	Type	Default	Value Range	Description
test_duration	Float	120.0	30.0 - 3600.0	How long the Targeted Stress test should run for in seconds. It is recommended to set this to at least 30 seconds for performance to stabilize.
temperature_max	Float	*Null*	30.0 - 120.0	The maximum temperature in C the card is allowed to reach during the test. Note that this check is disabled by default. Use nvidia-smi -q to see the normal temperature limits of your device.
target_stress	Float	*Null*	SKU dependent	The maximum relative stress each card will attempt to achieve.
max_pcie_replays	Float	160.0	1.0 - 1000000.0	Maximum number of PCIe replays to allow per GPU for the duration of this plugin. This is based on an expected

Parameter Name	Type	Default	Value Range	Description
				replay rate <8 per minute for PCIe Gen 3.0, assuming this plugin will run for 2 minutes (configurable) and allowing 10x as many replays before failure.

Stat Outputs

Stat Name	Stat Scope	Type	Description
power_usage	GPU	Time series Float	Per second power usage of each GPU in watts. Note that for multi-GPU boards, each GPU gets a fraction of the power budget of the board.
graphics_clock	GPU	Time series Float	Per second clock rate of each GPU in MHZ
memory_clock	GPU	Time series Float	Per second clock rate of the GPU's memory in MHZ
nvml_events	GPU	Time series Int64	Any events that were read with nvmlEventSetWait during the test and the timestamp it was read it.
power_violation	GPU	Time series Float	Percentage of time this GPU was violating power constraints.
gpu_temperature	GPU	Time series Float	Per second temperature of the GPU in degrees C
perf_gflops	GPU	Time series Float	The per second reading of average

Stat Name	Stat Scope	Type	Description
			gflops since the test began.
flops_per_op	GPU	Float	Flops (floating point operations) per operation queued to the GPU stream. One operation is one call to cublasSgemm or cublasDgemm
bytes_copied_per_op	GPU	Float	How many bytes are copied to + from the GPU per operation
num_cuda_streams	GPU	Float	How many cuda streams were used per gpu to queue operations to the GPUs
try_ops_per_sec	GPU	Float	Calculated number of ops/second necessary to achieve target gigaflops
pcie_replay_count	GPU	Float	The per second reading of PCIe replays that have occurred since the start of the Targeted Stress plugin.

Failure

The plugin will fail if:

- The GPU temperature exceeds a user-provided threshold.
- If temperature violation counters increase
- If the target stress level cannot be reached
- If GPU double bit ECC errors occur or the configured amount of SBE errors occur.
- If the number of PCIe retransmits exceeds a user-provided threshold.
- A crtical XID occurs

3.7. Power Plugin

The purpose of the power plugin is to bring the GPUs to a preconfigured power level in watts by gradually increasing the compute load on the GPUs until the desired power level is achieved. This verifies that the GPUs can sustain a power level for a reasonable amount of time without problems like thermal violations arising.

Preconditions

None

Configuration Parameters

Parameter Name	Type	Default	Value Range	Description
test_duration	Float	120.0	30.0 - 3600.0	How long the performance test should run for in seconds. It is recommended to set this to at least 60 seconds for performance to stabilize.
temperature_max	Float	*Null*	30.0 - 120.0	The maximum temperature in C the card is allowed to reach during the test. Note that this check is disabled by default. Use nvidia-smi -q to see the normal temperature limits of your device.
target_power	Float	Differs per GPU	Differs per GPU. Defaults to TDP - 1 watt.	What power level in wattage we should try to maintain. If this is set to greater than the

Parameter Name	Type	Default	Value Range	Description
				enforced power limit of the GPU, then we will try to power cap the device

Stat Outputs

Stat Name	Stat Scope	Type	Description
power_usage	GPU	Time series Float	Per second power usage of each GPU in watts. Note that for multi-GPU boards, each GPU gets a fraction of the power budget of the board.
graphics_clock	GPU	Time series Float	Per second clock rate of each GPU in MHZ
memory_clock	GPU	Time series Float	Per second clock rate of the GPU's memory in MHZ
nvml_events	GPU	Time series Int64	Any events that were read with nvmlEventSetWait during the test and the timestamp it was read it.
power_violation	GPU	Time series Float	Percentage of time this GPU was violating power constraints.
gpu_temperature	GPU	Time series Float	Per second temperature of the GPU in degrees C

Failure

The plugin will fail if:

- The GPU temperature exceeds a user-provided threshold.
- If temperature violation counters increase

- If the target performance level cannot be reached
- If GPU double bit ECC errors occur or the configured amount of SBE errors occur.
- If a critical XID occurs

Chapter 4. TEST OUTPUT

The output of tests can be collected by setting the "logfile" global parameter which represents the prefix for the detailed outputs produced by each test. The default type of output is JSON but text and binary outputs are available as well. The latter two are meant more for parsing and direct reading by custom consumers respectively so this portion of the document will focus on the JSON output.

4.1. JSON Output

The JSON output format is keyed based off of the "stats" keys given in each test overview from Chapter 3. These standard JSON files can be processed in any number of ways but two example Python scripts have been provided to aid in visualization in the default installation directory.. The first is a JSON to comma-separated value script (json2csv.py) which can be used to import key values in to a graphing spreadsheet. Proper usage would be:

```
user@hostname
$ python json2csv.py -i stats_targeted_performance.json -o stats.csv -k
 gpu_temperature,power_usage
```

Also provided is an example Python script that uses the pygal library to generate readily viewable scalar vector graphics charts (json2svg.py), able to be opened in any browser. Proper usage would be:

```
user@hostname
$ python json2svg.py -i stats_targeted_performance.json -o stats.svg -k
 gpu_temperature,power_usage
```