# MVAPICH2-GDR 2.3.7

<http://mvapich.cse.ohio-state.edu/userguide/gdr/>

# 目录

[1. Overview](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_overview)

[2. Supported Platforms](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_supported_platforms)

[3. System Requirements](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_system_requirements)

[3.1. NVIDIA GPUs](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_nvidia_gpus)

[3.2. AMD GPUs](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_amd_gpus)

[4. Strongly Recommended System Features](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_strongly_recommended_system_features)

[5. Installing MVAPICH2-GDR library](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_installing_mvapich2_gdr_library)

[6. Installing MVAPICH2-GDR using Spack](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_installing_mvapich2_gdr_using_spack)

[7. Running applications](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_running_applications)

[7.1. Example running OSU Micro Benchmark for NVIDIA GPUs](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_example_running_osu_micro_benchmark_for_nvidia_gpus)

[7.2. Example running OSU Micro Benchmark for AMD GPUs](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_example_running_osu_micro_benchmark_for_amd_gpus)

[7.3. Example running Deep Learning Frameworks with Horovod and MVAPICH2-GDR](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_example_running_deep_learning_frameworks_with_horovod_and_mvapich2_gdr)

[7.4. Example use of LD\_PRELOAD](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_example_use_of_ld_preload)

[8. Compile time and run-time check for CUDA-aware support](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_compile_time_and_run_time_check_for_cuda_aware_support)

[9. Supported Configurations](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_supported_configurations)

[9.1. Single GPU / Single HCA](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_single_gpu_single_hca)

[9.2. Single GPU / Multiple HCA](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_single_gpu_multiple_hca)

[9.3. Multiple GPU / Single HCA](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_multiple_gpu_single_hca)

[9.4. Multiple GPU / Multiple HCA](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_multiple_gpu_multiple_hca)

[9.5. Examples using OSU micro-benchmarks with multi-rail support](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_examples_using_osu_micro_benchmarks_with_multi_rail_support)

[10. Enabling Support on GPU-Clusters using regular OFED (without GPUDirect RDMA)](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_enabling_support_on_gpu_clusters_using_regular_ofed_without_gpudirect_rdma)

[11. Enabling Support for Managed Memory (NVIDIA GPUs only)](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_enabling_support_for_managed_memory_nvidia_gpus_only)

[12. Enabling Support for InfiniBand hardware UD-Multicast based collectives (NVIDIA GPUs only)](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_enabling_support_for_infiniband_hardware_ud_multicast_based_collectives_nvidia_gpus_only)

[13. Tuning and Usage Parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters)

[13.1. Basic Usage](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_basic_usage)

[13.2. Additional Parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_additional_parameters)

[13.3. Running on OpenPOWER Systems](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_running_on_openpower_systems)

[13.4. GDRCOPY Feature](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_gdrcopy_feature)

[13.5. Loopback Feature](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_loopback_feature)

[13.6. Non-Blocking Collective Support with GDR and Core-Direct](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_non_blocking_collective_support_with_gdr_and_core_direct)

[13.7. CPU Binding and Mapping Parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#cpu_binding)

[13.8. GPU Datatype Processing Feature](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_gpu_datatype_processing_feature)

[14. GPU-Aware MPI Primitives](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_gpu_aware_mpi_primitives)

[15. CUDA and OpenACC Extensions to OMB](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_cuda_and_openacc_extensions_to_omb)

[16. Managed Memory Extensions to OMB](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_managed_memory_extensions_to_omb)

[16.1. Example running OSU Micro Benchmarks with Managed Memory Support](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_example_running_osu_micro_benchmarks_with_managed_memory_support)

[17. Container Support](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_container_support)

## 1. Overview

MVAPICH2-GDR 2.3.7 binary release is based on MVAPICH2 2.3.7 and incorporates designs that take advantage of GPUDirect RDMA technology enabling direct P2P communication between NVIDIA GPUs and Mellanox InfiniBand adapters. MVAPICH2-GDR 2.3.7 also adds support for AMD GPUs via Radeon Open Compute (ROCm) software stack and exploits ROCm RDMA technology for direct communication between AMD GPUs and Mellanox InfiniBand adapters. MVAPICH2-GDR 2.3.7 offers significant improvements in latency and bandwidth for GPU-buffer based intranode and internode MPI Communication involving small and medium message sizes. For more information on the GPUDirect RDMA technology, refer to <https://www.mellanox.com/products/GPUDirect-RDMA>.

MVAPICH2-GDR 2.3.7 provides an efficient support for Non-Blocking Collectives (NBC) from GPU buffers to achieve maximal overlap. It uses novel designs that combine GPUDirect RDMA and Core-Direct technologies. Further MVAPICH2-GDR 2.3.7 also provides support for CUDA Managed memory features and optimizes large message collectives targeting Deep Learning frameworks.

Note that this release is for GPU-Cluster with GPUDirect RDMA support, if your cluster does not have this support please use the default MVAPICH2 library. For more details please refer to <http://mvapich.cse.ohio-state.edu/>.

## 2. Supported Platforms

1. Intel and other x86 Systems
2. OpenPOWER 8 and 9 Systems

## 3. System Requirements

MVAPICH2-GDR 2.3.7 binary release requires the following software to be installed on your system:

3.1. NVIDIA GPUs

1. [Mellanox OFED 3.2 and later](http://www.mellanox.com/page/products_dyn?product_family=26)
2. [NVIDIA Driver 367.48 or later](http://www.nvidia.com/Download/driverResults.aspx/69372/)
3. [NVIDIA CUDA Toolkit 7.5 and later](https://developer.nvidia.com/cuda-toolkit)
4. [NVIDIA Peer Memory (nv\_peer\_mem) module to enable GPUDirect RDMA (GDR) support](https://www.mellanox.com/products/GPUDirect-RDMA)

List of Mellanox InfiniBand adapters and NVIDIA GPU devices which support GPUDirect RDMA can be found [here](http://www.mellanox.com/page/products_dyn?product_family=116).

3.2. AMD GPUs

1. [Mellanox OFED 3.2 and later](http://www.mellanox.com/page/products_dyn?product_family=26)
2. [AMD ROCm installation (for AMD GPUs)](https://rocmdocs.amd.com/en/latest/Installation_Guide/Installation-Guide.html)

## 4. Strongly Recommended System Features

MVAPICH2-GDR 2.3.7 boosts the performance by taking advantage of the new GDRCOPY module from NVIDIA. In order to take advantage of this feature, please download and install this module from: https://github.com/NVIDIA/gdrcopy

For GDRCopy v2.x, please use MVAPICH2-GDR v2.3.3 or newer versions.

After installing this module you need to add this path to your LD\_LIBRARY\_PATH or use MV2\_GPUDIRECT\_GDRCOPY\_LIB to pass this path to the MPI library at runtime. For more details please refer to section (GDRCOPY USAGE AND TUNING) of this README. Note that even if this module is not available, MVAPICH2-GDR 2.3.7 will deliver very good performance by taking advantage of the Loopback feature. For more details refer to section (LOOPBACK FEATURE) of this README.

*For ROCm version of MVAPICH2-GDR, you do not have to specify the path to GDRCOPY library. The rest of the tuning parameters for GDRCOPY are still applicable.*

## 5. Installing MVAPICH2-GDR library

To install the MVAPICH2-GDR library you simply need to select the correct library (MOFED version, Compiler version, etc.) for your system and install the RPM using your favorite RPM tool. Please use the downloads page (<http://mvapich.cse.ohio-state.edu/downloads/>) to find the appropriate RPM link and follow the instructions below.

Install library using the downloaded RPM file

$ wget http://mvapich.cse.ohio-state.edu/download/mvapich/gdr/2.3.7/<mv2-gdr-rpm-name>.rpm

$ rpm -Uvh --nodeps <mv2-gdr-rpm-name>.rpm

The RPMs contained in our libraries are relocatable and can be installed using a prefix other than the default of ./opt/mvapich2/ used by the library in the previous example.

Install library specifying custom prefix

$ rpm --prefix /custom/install/prefix -Uvh --nodeps <mv2-gdr-rpm-name>.rpm

If you do not have root permission you can use rpm2cpio to extract the library.

Use rpm2cpio to extract the library

$ rpm2cpio <mv2-gdr-rpm-name>.rpm | cpio -id

When using the rpm2cpio method, you will need to update the MPI compiler scripts, such as mpicc, in order to point to the correct path of where you place the library.

|  |  |
| --- | --- |
| Tip | If you are using a Debian based system such as Ubuntu you can convert the rpm to a deb using a tool such as alien or follow the rpm2cpio instructions above.  sudo apt-get install alien #alien默认没有安装，所以首先要安装它  sudo alien xxxx.rpm #将rpm转换位deb，完成后会生成一个同名的xxxx.deb  sudo dpkg -i xxxx.deb #安装  注意，用alien转换的deb包并不能保证100%顺利安装，所以可以找到deb最好直接用deb |

## 6. Installing MVAPICH2-GDR using Spack

MVAPICH2-GDR can be installed using Spack without building it from source. See the Spack user guide for details:

参考：<https://mvapich.cse.ohio-state.edu/userguide/userguide_spack/>

## 7. 运行应用程序

Here are some examples running applications with the MVAPICH2-GDR software.

7.1. Example running OSU Micro Benchmark for NVIDIA GPUs

To run point-to-point and collective benchmarks for measuring internode latency between GPUs when enabling GPUDirect RDMA-based designs in MVAPICH2-GDR 2.3.7

1: $ **export** MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ **export** MV2\_GPUDIRECT\_GDRCOPY\_LIB=/path/to/GDRCOPY/install/lib64/libgdrapi.so

3: $ **export** MV2\_USE\_CUDA=1

4:

5: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

6: $MV2\_PATH/libexec/osu-micro-benchmarks/get\_local\_rank \

7: $MV2\_PATH/libexec/osu-micro-benchmarks/mpi/pt2pt/osu\_latency D D

8:

9: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

10: $MV2\_PATH/libexec/osu-micro-benchmarks/get\_local\_rank \

11: $MV2\_PATH/libexec/osu-microbenchmarks/mpi/collective/osu\_allreduce -d cuda

7.2. Example running OSU Micro Benchmark for AMD GPUs

To run point-to-point and collective tests running on AMD GPUs using ROCm-aware designs in MVAPICH2-GDR 2.3.7

1: $ **export** MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ **export** MV2\_USE\_ROCM=1

3:

4: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

5: $MV2\_PATH/libexec/osu-micro-benchmarks/get\_local\_rank \

6: $MV2\_PATH/libexec/osu-micro-benchmarks/mpi/pt2pt/osu\_latency D D

7:

8: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

9: $MV2\_PATH/libexec/osu-micro-benchmarks/get\_local\_rank \

10: $MV2\_PATH/libexec/osu-microbenchmarks/mpi/collective/osu\_allreduce -d rocm

Note: When using Spack installation, if you did not use spack load command, you may need to prepend MVAPICH2-GDR library to LD\_LIBRARY\_PATH like this:

$ export LD\_LIBRARY\_PATH=$HOME/spack/opt/spack/linux-centos7-x86\_64/gcc-<ver>/mvapich2-gdr-<ver>-<hash>/lib/:$LD\_LIBRARY\_PATH

7.3. Example running Deep Learning Frameworks with Horovod and MVAPICH2-GDR

MVAPICH2-GDR supports TensorFlow/PyTorch/MXNet with Horovod/MPI design but a special flag is needed to run the jobs properly. Please use the MV2\_SUPPORT\_DL=1 or MV2\_SUPPORT\_TENSOR\_FLOW=1 runtime variable but do not use the **LD\_PRELOAD** option. The variable MV2\_SUPPORT\_TENSOR\_FLOW will be deprecated in the future.

1: $ **export** MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ **export** MV2\_USE\_CUDA=1

3: $ **export** MV2\_SUPPORT\_DL=1

4:

5: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

6: python tf\_cnn\_benchmarks.py --model=resnet50 \

7: --variable\_update=horovod

For more details, please refer to our [**HiDL User Guide**](http://hidl.cse.ohio-state.edu/userguide/horovod/).

7.4. Example use of LD\_PRELOAD

Some cases may require **LD\_PRELOAD** to be set to the path of the MVAPICH2 library. This should be limited to cases where the application uses an interpreter such as python to load the CUDA library.

Try setting **LD\_PRELOAD** if you find the MVAPICH2 fails when using your CUDA device buffers in MPI calls.

Example running HOOMD-BLUE with LD\_PRELOAD

1: $ **export** MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ **export** MV2\_GPUDIRECT\_GDRCOPY\_LIB=/path/to/GDRCOPY/install/lib64/libgdrapi.so

3: $ **export** MV2\_USE\_CUDA=1

4:

5: $ $MV2\_PATH/bin/mpirun\_rsh -n 4 hostA hostA hostB hostB \

6: LD\_PRELOAD=$MV2\_PATH/lib/libmpi.so hoomd lj\_liquid\_bmark.hoomd

When Jsrun launcher is used on IBM OpenPOWER systems such as Summit and Sierra, please add the MVAPICH2 library path to **OMPI\_LD\_PRELOAD\_PREPEND**.

Example running OSU benchmark using Jsrun with OMPI\_LD\_PRELOAD\_PREPEND

1: $ **export** MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ **export** MV2\_GPUDIRECT\_GDRCOPY\_LIB=/path/to/GDRCOPY/install/lib64/libgdrapi.so

3: $ **export** MV2\_USE\_CUDA=1

4: $ **export** LD\_LIBRARY\_PATH=/path/to/pmi4pmix/lib:$LD\_LIBRARY\_PATH

5: $ **export** OMPI\_LD\_PRELOAD\_PREPEND=$MV2\_PATH/lib/libmpi.so

6:

7: $ jsrun -n2 ./osu\_latency D D

## 8. Compile time and run-time check for CUDA-aware support

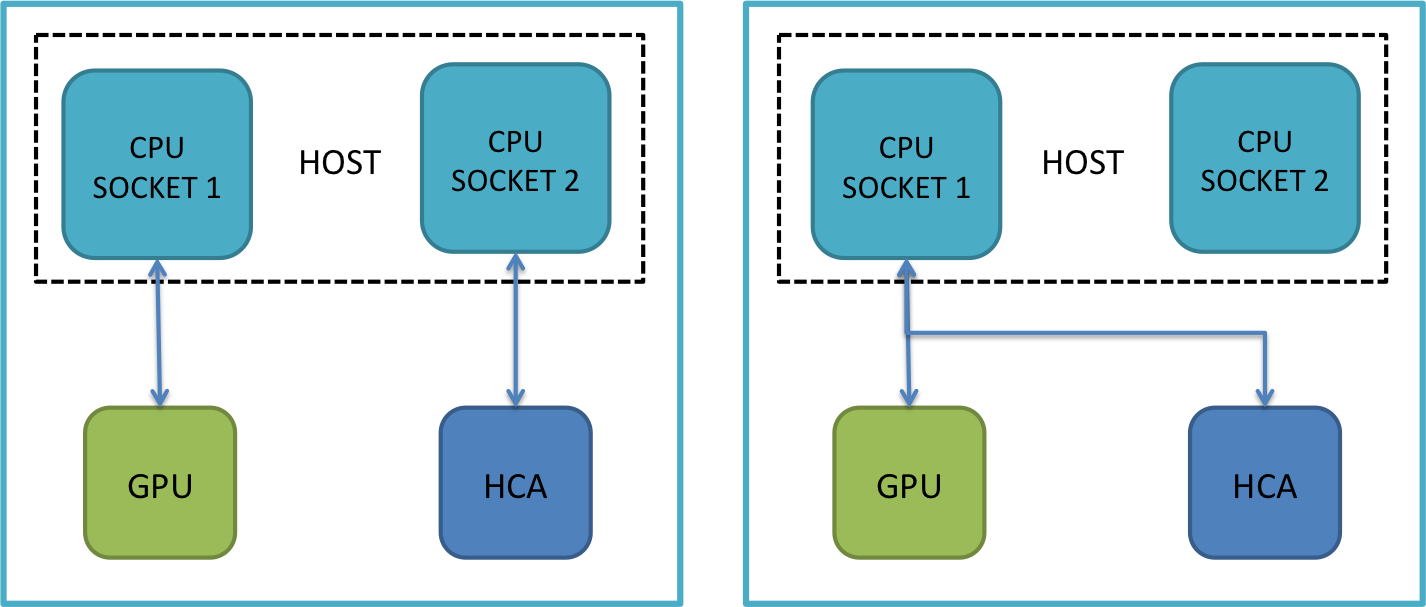
Since MVAPICH2 2.3.5, a compile-time macro **MPIX\_CUDA\_AWARE\_SUPPORT** and a run-time function **MPIX\_Query\_cuda\_support()** have been added to determine whether CUDA-aware primitives are supported or not. To access them, you need to include **mpi-ext.h** similar to OpenMPI. A sample program of using these checks can be found in <https://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support>.

## 9. Supported Configurations

Below is the list of currently supported configurations. In order to select the GPU, the application is free to use any selection method. For HCA, the parameters that are required have to exported as shown below. Note that MVAPICH2-GDR 2.3.7 does automatically and dynamically the best binding by default, and prints a warning if the user does specify a binding which is not the best mapping. See [CPU Binding and Mapping Parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#cpu_binding) for more information.

9.1. Single GPU / Single HCA

With single HCA, the default example shown earlier will work fine since there is no HCA selection involved for Single HCA configurations. （如果只是用一个网卡，默认设置就能获得很好的性能，因为不涉及HCA选择）



Single GPU / Single HCA : (a) 1 GPU and 1 HCA in Different Sockets (b) 1 GPU and 1 HCA in Same Socket

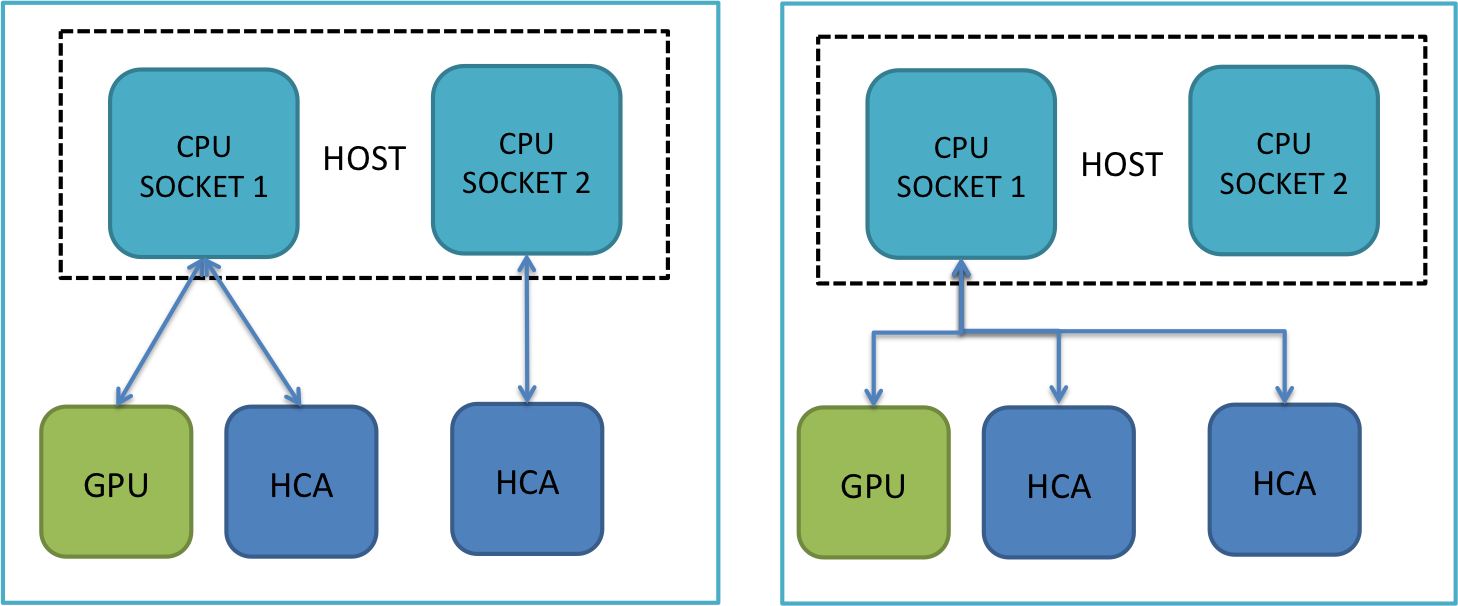
9.2. Single GPU / Multiple HCA

As mentioned earlier, MVAPICH2-GDR 2.3.7 will automatically tries to take advantage of the multirail configuration for small and large message sizes. Note that if one HCA is far from the GPU (different sockets), then the default multirail selection might affect the performance. Thus, please do an explicit HCA selection, selecting the nearest as the first HCA In order to explicitly select the near HCA, please use MV2\_IBA\_HCA or MV2\_PROCESS\_TO\_RAIL\_MAPPING parameter.

$ **export** MV2\_PROCESS\_TO\_RAIL\_MAPPING=mlx5\_0:mlx5\_1

$ or

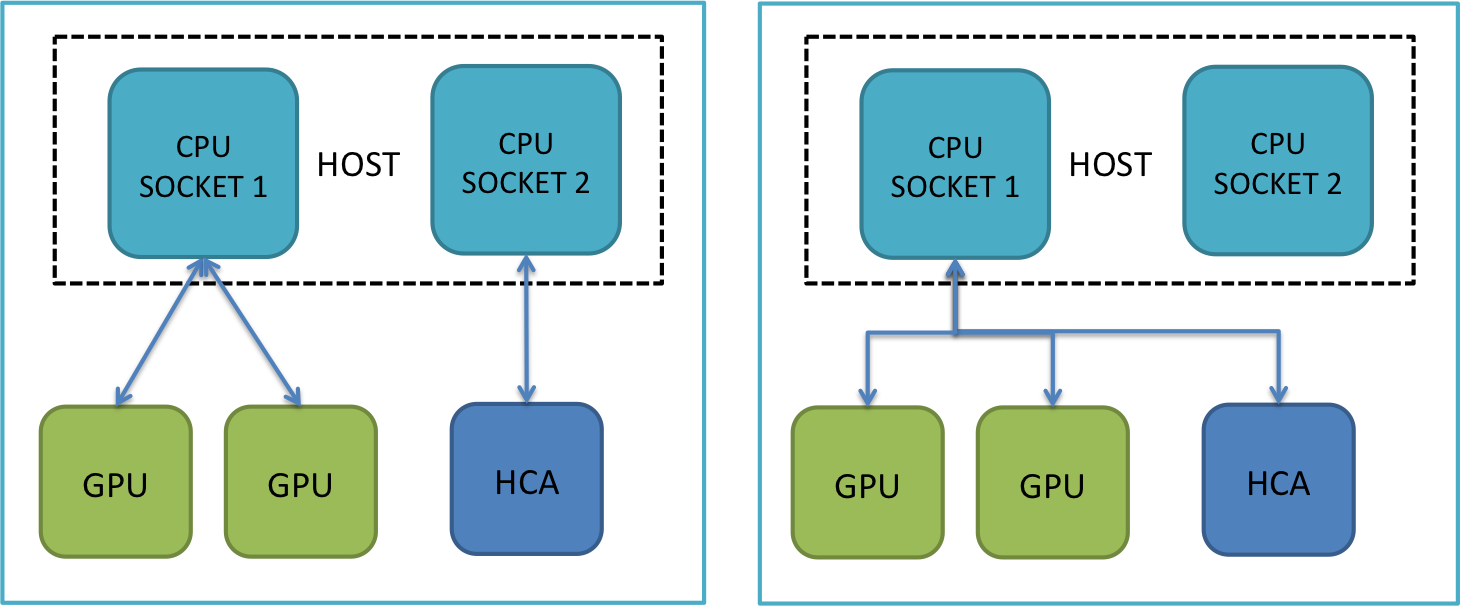
$ **export** MV2\_IBA\_HCA=mlx5\_0:mlx5\_1



**Single GPU / Multiple HCA : (a) 1 GPU and 2 HCAs in Different Sockets (b) 1 GPU and 2 HCAs in Same Socket**

9.3. Multiple GPU / Single HCA

For this configuration, users can use same parameters as Single GPU / Single



**Multiple GPU / Single HCA : (a) 2 GPUs and 1 HCA in Different Sockets (b) 2 GPUs and 1 HCA in Same Socket**

9.4. Multiple GPU / Multiple HCA

By default MVAPICH2-GDR 2.3.7 will automatically select the HCA near to the GPU select by the process and bind that process to a core on the same socket. Further, as MVAPICH2-GDR supports the dynamic initialization of the CUDA devices, the HCA re-selection is transparently and dynamically performed. The parameters below are an explicit setting to the same default setting.

$ **export** MV2\_RAIL\_SHARING\_POLICY=FIXED\_MAPPING

$ **export** MV2\_PROCESS\_TO\_RAIL\_MAPPING=mlx5\_0:mlx5\_1

$ or

$ **export** MV2\_IBA\_HCA=mlx5\_0:mlx5\_1

9.5. Examples using OSU micro-benchmarks with multi-rail support

To run the osu\_bw test with multi-rail support, use the following command.

$ **export** MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

$ **export** MV2\_GPUDIRECT\_GDRCOPY\_LIB= path to the GDRCOPY install

$ **export** MV2\_USE\_CUDA=1

$ **export** MV2\_PROCESS\_TO\_RAIL\_MAPPING=mlx5\_0:mlx5\_1

$ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

$MV2\_PATH/libexec/mvapich2/get\_local\_rank \

$MV2\_PATH/libexec/mvapich2/osu\_bw -d cuda (or -d rocm **for** AMD GPUs)

For more information about running OSU micro-benchmarks to measure MPI Multiple GPU / Multiple HCA : 2 GPUs and 2 HCAs in Different Sockets] communication performance on NVIDIA GPU clusters, please refer to:

<http://mvapich.cse.ohio-state.edu/benchmarks/>

## 10. Enabling Support on GPU-Clusters using regular OFED (without GPUDirect RDMA)

Targeted for GPU-based Systems that do not support GPUDirect RDMA feature:

A) Use regular OFED instead of Mellanox OFED (MOFED)

B) Use older GPUs that do not support GPUDirect RDMA feature.

For such systems, MVAPICH2-GDR 2.3.7 efficiently takes advantage of CUDA IPC and GDRCOPY features. Please download and install the appropriate package.

## 11. Enabling Support for Managed Memory (NVIDIA GPUs only)

Managed Memory feature provides high productivity in developing CUDA kernels by allowing the same memory allocation (pointer) to be used on both CPU and GPU. To enable this productivity for distributed memory programming, MVAPICH2-GDR 2.3.7 provides CUDA-Aware MPI support with manged memory allocation. In other words, MPI calls can be directly performed on managed memory buffers. Further, it allows mixing different memory allocations in a single application. For instance one can perform a send operation from a managed memory buffer and receive it on a buffer allocated directly on the GPU memory.

To enable this feature, please set MV2\_CUDA\_ENABLE\_MANAGED=1 when executing your application.

MVAPICH2-GDR 2.3.7 enhances the intra-node support for managed memory when all GPUs in a node are peer-access. The design takes advantage of CUDA-IPC to boost the performance of data movement operations from/to managed buffers. To enable this feature, please add MV2\_CUDA\_MANAGED\_IPC=1 on your execution command line.

## 12. Enabling Support for InfiniBand hardware UD-Multicast based collectives (NVIDIA GPUs only)

InfiniBand hardware UD-Multicast (IB-MCAST) is a feature for designing highly scalable collective operations. Combining IB-MCAST and other features in MVAPICH2-GDR is extremely helpful for high-performance streaming applications, which are using MPI\_Bcast.

This feature is disabled by default. Basic support for IB-MCAST feature with MVAPICH2-GDR can be enabled by using following parameters. Please refer to [**MVAPICH2 User Guide**](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3.7-userguide.html) for more details.

* MV2\_USE\_MCAST
  + Default: 0
  + Set this to 1, to enable hardware multicast support in collective communication
* MV2\_USE\_RDMA\_CM\_MCAST
  + Default: 0
  + Set this to 1, to enable RDMA-CM-based hardware multicast support in collective communication
* MV2\_MCAST\_NUM\_NODES\_THRESHOLD
  + Default: 8
  + This defines the threshold for enabling multicast support in collective communication. When MV2\_USE\_MCAST is set to 1 and the number of nodes in the job is greater than or equal to the threshold value, it uses multicast support in collective communication.

In addition to the basic support, advanced features can be enabled by using the following parameters:

* MV2\_GPUDIRECT\_MCAST\_PIPELINE
  + Default: 1 (Enabled)
  + To disable the support for efficient pipelined IB-MCAST from NVIDIA GPUs, set to 0.
* MV2\_GPUDIRECT\_MCAST\_RECV\_TYPE
  + Default: 1 (Basic support)
  + To toggle support for IB-MCAST from/to NVIDIA GPUs. For enabling efficient zero-copy design, set to 2.
* MV2\_MCAST\_RELIABILITY\_TYPE
  + Default: 1 (Negative acknowledgement-based scheme)
  + To toggle reliability support for UD-based IB-MCAST from/to NVIDIA GPUs. For enabling RMA-based reliability scheme, set to 2.

## 13. 调优使用参数

Note that MVAPICH2-GDR selects the optimal value for each of the following parameters based on architecture detection.

13.1. 基本用法

Unless explicitly mentioned, all the following environment variables are applicable to both CUDA and ROCm versions of MVAPICH2-GDR

* MV2\_USE\_CUDA
  + Default: 0 (Disabled)
  + To toggle support for communication from NVIDIA GPUs. For enabling, set to 1.
  + Applicable: NVIDIA GPUs only
* MV2\_USE\_ROCM
  + Default: 0 (Disabled)
  + To toggle support for communication from AMD GPUs. For enabling, set to 1.
  + Applicable: AMD GPUs only
* MV2\_USE\_GPUDIRECT\_RDMA
  + Default: 1 (Enabled)
  + The runtime variable *MV2\_USE\_GPUDIRECT* will be deprecated in future. Please use *MV2\_USE\_GPUDIRECT\_RDMA* or *MV2\_USE\_GDR* to enable/disable the GPU Direct RDMA support in MVAPICH2. Please note that disabling this feature only disables the GPU Direct RDMA support. Other features of the GPU Direct family like GPU Direct Peer-to-Peer (P2P) communication are not affected by this parameter.
* MV2\_CUDA\_BLOCK\_SIZE
  + Default: 262144
  + To tune pipelined internode transfers between NVIDIA GPUs. Higher values may help applications that use larger messages and are bandwidth critical.
* MV2\_GPUDIRECT\_LIMIT
  + Default: 8192
  + To tune the hybrid design that uses pipelining and GPUDirect RDMA for maximum performance while overcoming P2P bandwidth bottlenecks seen on modern systems. GPUDirect RDMA is used only for messages with size less than or equal to this limit. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.
* MV2\_USE\_GPUDIRECT\_RECEIVE\_LIMIT
  + Default: 131072
  + To tune the hybrid design that uses pipelining and GPUDirect RDMA for maximum performance while overcoming P2P read bandwidth bottlenecks seen on modern systems. Lower values (16384) may help improve performance on nodes with multiple GPUs and IB adapters. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.
* MV2\_CUDA\_IPC\_THRESHOLD
  + Default: 32768
  + To tune the usage of IPC communication path for intranode configuration. Note that if you have processes sharing the same GPU, please increase this parameter to a higher value like 524288.
* MV2\_CUDA\_USE\_IPC\_BCAST
  + Default: 0 (Disabled)
  + To toggle support for IPC-based intra-node broadcast among NVIDIA GPUs. For enabling, set to 1.

13.2. 其他参数

The following parameters are required in some specific use-cases. Here we will explain these additional parameters and when to use them.

* MV2\_USE\_LAZY\_MEM\_UNREGISTER
  + Default: 1 (Default behavior of IB registration cache)
  + For host buffers created by CUDA/ROCm APIs e.g., cudaMallocHost / hipMallocHost, it is observed that the same virtual address (VA) is returned after few iterations. If this buffer had been registered with IB and cached in IB registration cache, then we have to evict the stale entries upon free’ing. While this is already taken care of in normal scenario where host buffers are allocated with malloc/mmap calls, it becomes a special case where the host buffer is allocated by device APIs. In order to circumvent the invalidation issues, we recomment setting the value of this variable to to 2.

1: $ **export** MV2\_USE\_LAZY\_MEM\_UNREGISTER=2

13.3. Running on OpenPOWER Systems

* MV2\_USE\_GPUDIRECT\_RDMA
  + Default: 1
  + Current generation (POWER8) systems do not have GPUDirect RDMA support so users should disable this support by setting MV2\_USE\_GPUDIRECT\_RDMA=0
  + There are no expected side effects when GPUDIRECT support is disabled.

13.4. GDRCOPY Feature

* MV2\_GDRCOPY\_LIMIT
  + Default: 8192
  + The runtime variable *MV2\_USE\_GPUDIRECT\_GDRCOPY\_LIMIT* will be deprecated in future. Please use *MV2\_GDRCOPY\_LIMIT* to tune the local transfer threshold using gdrcopy module between GPU and CPU for point to point communications. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.
* MV2\_GDRCOPY\_NAIVE\_LIMIT
  + Default: 8192
  + The runtime variable *MV2\_USE\_GPUDIRECT\_GDRCOPY\_NAIVE\_LIMIT* will be deprecated in future. Please use *MV2\_GDRCOPY\_NAIVE\_LIMIT* to tune the local transfer threshold using gdrcopy module between GPU and CPU for collective communications. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.

13.5. Loopback Feature

* MV2\_USE\_GPUDIRECT\_LOOPBACK\_LIMIT
  + Default: 8192
  + To tune the transfer threshold using loopback design for point to point communications. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.
* MV2\_USE\_GPUDIRECT\_LOOPBACK\_NAIVE\_LIMIT
  + Default: 8192
  + To tune the transfer threshold using loopback design for collective communications. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.

13.6. Non-Blocking Collective Support with GDR and Core-Direct

* MV2\_USE\_CORE\_DIRECT
  + Default: 0
  + To turn ON this features please set it to 1
  + To enable efficient support while working around the P2P bottleneck please set to 2
  + To provide a maximal overlap for both CPU and GPU simultaneous computing while avoiding the P2P bottlenecks set to 3
  + Applicable: NVIDIA GPUs

13.7. CPU Binding and Mapping Parameters

When experimenting on nodes with multiple NVIDIA GPUs and InfiniBand adapters, selecting the right NVIDIA GPU and IB adapter at each MPI process can be important to achieve good performance. The following parameters help uses bind processes to different IB HCAs. GPU device selection is expected to be made in the application using CUDA interfaces like cudaSetDevice. For the IB selection we have the below scenarios :

1) Multi-IB and Multi-GPU scenario: on a systems with 2 IBs and 2 GPUs, achieving the best best performance requires the processes to use the GPU closest to the IB. To do so:

* MV2\_PROCESS\_TO\_RAIL\_MAPPING
  + Default: NONE
  + Value Domain: BUNCH, SCATTER, <CUSTOM LIST>
  + When MV2\_RAIL\_SHARING\_POLICY is set to the value FIXED\_MAPPING this variable decides the manner in which the HCAs will be mapped to the rails. The<CUSTOM LIST> is colon(:) separated list with the HCA ranks (e.g. 0:1:1:0) or HCA names specified (e.g. mlx5\_0:mlx5\_1:mlx5\_0:mlx5\_1). For more information on this parameter, refer to following section of MVAPICH2 user guide: <http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3.7-userguide.html#x1-22200011.40>

2) Multi-IB and 1 GPU scenario : for this scenario and in order to take advantage of the Multi-rails support, sending a large message from/to a GPU will take advantage of both IBs. In addition to the MV2\_PROCESS\_TO\_RAIL\_MAPPING parameter, the following parameters can be used:

* MV2\_RAIL\_SHARING\_POLICY
  + Default: ROUND\_ROBIN
  + Value Domain: USE\_FIRST, ROUND\_ROBIN, FIXED\_MAPPING
  + This specifies the policy that will be used to assign HCAs to each of the processes. For more information on this parameter, refer to following section of MVAPICH2 user guide: <http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3.7-userguide.html#x1-22000011.38>
* MV2\_RAIL\_SHARING\_LARGE\_MSG\_THRESHOLD
  + Default: 16K
  + This specifies the message size beyond which striping of messages across all available rails will take place.

It is also important to bind MPI processes as close to the GPU and IB adapter as possible. The following parameter allows you to manually control process-to-core mapping. MVAPICH2-GDR 2.3.7 does automatically the best binding by default, and prints a warning if the user does specify a binding which is not the best mapping.

* MV2\_CPU\_MAPPING
  + Default: Unset
  + This allows users to specify process to CPU (core) mapping. The detailed usage of this parameter is described in [MVAPICH2 user guide](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3.7-userguide.html#x1-19500011.13)

13.8. GPU Datatype Processing Feature

These features are only supported for NVIDIA GPUs for now.

* MV2\_CUDA\_DIRECT\_DT\_THRESHOLD
  + Default: 8
  + To tune the direct transfer scheme using asynchronous CUDA memory copy for datatype packing/unpacking. Direct transfer scheme can avoid the kernel invocation overhead for dense datatypes. It has to be tuned based on the node architecture, the processor, the GPU and the IB card.
* MV2\_CUDA\_KERNEL\_VECTOR\_TIDBLK\_SIZE
  + Default: 1024
  + To tune the thread block size for vector/hvector packing/unpacking kernels. It has to be tuned based on the vector/hvector datatype shape and the GPU.
* MV2\_CUDA\_KERNEL\_VECTOR\_YSIZE
  + Default: 32
  + To tune the x dimension of thread block for vector/hvector packing/unpacking kernels. This value is automatically tuned based on the block length of vector/hvector datatypes. It also can be tuned based on the vector/hvector datatype shape and the GPU.
* MV2\_CUDA\_KERNEL\_SUBARR\_TIDBLK\_SIZE
  + Default: 1024
  + To tune the thread block size for subarray packing/unpacking kernels. It has to be tuned based on the subarray datatype dimension, shape and the GPU.
* MV2\_CUDA\_KERNEL\_SUBARR\_XDIM
  + Default: 8 (3D) /16 (2D) /256 (1D)
  + To tune the x dimension of thread block for subarray packing/unpacking kernels. It has to be tuned based on the subarray datatype dimension, shape and the GPU.
* MV2\_CUDA\_KERNEL\_SUBARR\_YDIM
  + Default: 8 (3D) /32 (2D) /4 (1D)
  + To tune the y dimension of thread block for subarray packing/unpacking kernels. It has to be tuned based on the subarray datatype dimension, shape and the GPU.
* MV2\_CUDA\_KERNEL\_SUBARR\_ZDIM
  + Default: 16 (3D) /1 (2D) /1 (1D)
  + To tune the z dimension of thread block for subarray packing/unpacking kernels. It has to be tuned based on the subarray datatype dimension, shape and the GPU.
* MV2\_CUDA\_KERNEL\_ALL\_XDIM
  + Default: 16
  + To tune the x dimension of thread block for all datatypes except vector/hvector, indexed\_block/hindexed\_block and subarray. It has to be tuned based on the datatype shape and the GPU.
* MV2\_CUDA\_KERNEL\_IDXBLK\_XDIM
  + Default: 1
  + To tune the x dimension of thread block for indexed\_block/hindexed\_block packing/unpacking kernels.It has to be tuned based on the indexed\_block/hindexed\_block datatype shape and the GPU.
* MV2\_USE\_SHM\_CACHE\_DATATYPE
  + Default: 1
  + To toggle support for caching the layout of MPI derived datatype. For disabling, set to 0.
* MV2\_USE\_ZCPY\_DDT
  + Default: 1
  + To toggle support for performing zero-copy datatype procesing between GPUs through PCIe or NVLink. For disabling, set to 0.
* MV2\_CUDA\_USE\_DDT\_KERNEL\_FUSION
  + Default: 1
  + To toggle support for performing GPU-driven datatype procesing with kernel fusion feature for low-latency packing/unpacking. For disabling, set to 0.
* MV2\_CUDA\_USE\_DDT\_ASYNC\_KERNEL
  + Default: 1
  + To toggle support for performing asynchronous GPU-driven datatype procesing. For disabling, set to 0.

## 14. GPU-Aware MPI Primitives

The following GPU aware MPI primitives are available for both CUDA and ROCM stacks as a part of the library:

* Point to Point
  + MPI\_Send, MPI\_Recv, MPI\_Isend, MPI\_Irecv, MPI\_Ssend, MPI\_Rsend, MPI\_Issend, MPI\_Bsend, MPI\_Sendrecv, MPI\_Send\_init, MPI\_Recv\_init, MPI\_Ssend\_init, MPI\_Bsend\_init, MPI\_Rsend\_init
* Collectives
  + MPI\_Bcast, MPI\_Scatter, MPI\_Scatterv, MPI\_Gather, MPI\_Gatherv, MPI\_Reduce, MPI\_Allreduce, MPI\_Allgather, MPI\_Allgatherv, MPI\_Alltoall, MPI\_Alltoallv, MPI\_Scan, MPI\_Exscan, MPI\_Reduce\_scatter, MPI\_Reduce\_scatter\_block
* Non-Blocking Collectives (NBC)
  + MPI\_Ibcast, MPI\_Iscatter, MPI\_Igather, MPI\_Iallgather, MPI\_Ialltoall
* One Sided Communications [[1](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_footnote_1)]

**Allocation Primitives**

* + MPI\_Win\_create, MPI\_Win\_allocate, MPI\_Win\_allocate\_shared, MPI\_Win\_create\_dynamic

**Communication Primitives**

* + MPI\_Get, MPI\_Put, MPI\_Accumulate

**Synchronization Primitives**

* + MPI\_Win\_lock/MPI\_Win\_unlock, MPI\_Win\_lock\_all/MPI\_Win\_unlock\_all, MPI\_Win\_flush, MPI\_Win\_fence, MPI\_Win\_flush\_all, MPI\_Win\_flush\_local, MPI\_Win\_flush\_local\_all, MPI\_Win\_post/MPI\_Win\_start/MPI\_Win\_wait/MPI\_Win\_complete
* Enhanced support of MPI datatypes for GPU
  + MPI\_Type\_vector, MPI\_Type\_hvector, MPI\_Type\_create\_struct, MPI\_Type\_create\_subarray, MPI\_Type\_indexed, MPI\_Type\_hindexed, MPI\_Type\_create\_hindexed\_block

## 15. CUDA and OpenACC Extensions to OMB

The following benchmarks have been extended to evaluate performance of MPI communication from and to buffers on NVIDIA GPU devices.

|  |  |
| --- | --- |
| osu\_bibw | Bidirectional Bandwidth Test |
| osu\_bw | Bandwidth Test |
| osu\_latency | Latency Test |
| osu\_put\_latency | Latency Test for Put |
| osu\_get\_latency | Latency Test for Get |
| osu\_put\_bw | Bandwidth Test for Put |
| osu\_get\_bw | Bandwidth Test for Get |
| osu\_put\_bibw | Bidirectional Bandwidth Test for Put |
| osu\_acc\_latency | Latency Test for Accumulate |
| osu\_cas\_latency | Latency Test for Compare and Swap |
| osu\_fop\_latency | Latency Test for Fetch and Op |
| osu\_allgather | MPI\_Allgather Latency Test |
| osu\_allgatherv | MPI\_Allgatherv Latency Test |
| osu\_allreduce | MPI\_Allreduce Latency Test |
| osu\_alltoall | MPI\_Alltoall Latency Test |
| osu\_alltoallv | MPI\_Alltoallv Latency Test |
| osu\_bcast | MPI\_Bcast Latency Test |
| osu\_gather | MPI\_Gather Latency Test |
| osu\_gatherv | MPI\_Gatherv Latency Test |
| osu\_reduce | MPI\_Reduce Latency Test |
| osu\_reduce\_scatter | MPI\_Reduce\_scatter Latency Test |
| osu\_scatter | MPI\_Scatter Latency Test |
| osu\_scatterv | MPI\_Scatterv Latency Test |
| osu\_iallgather | MPI\_Iallgather Latency and Overlap Test |
| osu\_iallgatherv | MPI\_Iallgatherv Latency and Overlap Test |
| osu\_iallreduce | MPI\_Iallreduce Latency and Overlap est |
| osu\_ialltoall | MPI\_Ialltoall Latency and Overlap Test |
| osu\_ialltoallv | MPI\_Ialltoallv Latency and Overlap Test |
| osu\_ibcast | MPI\_Ibcast Latency and Overlap Test |
| osu\_igather | MPI\_Igather Latency and Overlap Test |
| osu\_igatherv | MPI\_Igatherv Latency and Overlap Test |
| osu\_iscatter | MPI\_Iscatter Latency and Overlap Test |
| osu\_iscatterv | MPI\_Iscatterv Latency and Overlap Test |

Some directions for usage are:

* The CUDA extensions are enabled when the benchmark suite is configured with --enable-cuda option.
* The OpenACC extensions are enabled when --enable-openacc is specified. Whether a process allocates its communication buffers on the GPU device or on the host can be controlled at run-time.
* Each of the pt2pt benchmarks takes two input parameters. The first parameter indicates the location of the buffers at rank 0 and the second parameter indicates the location of the buffers at rank 1. The value of each of these parameters can be either *H* or *D* to indicate if the buffers are to be on the host or on the device respectively. When no parameters are specified, the buffers are allocated on the host.
* The collective benchmarks will use buffers allocated on the device if the -d option is used otherwise the buffers will be allocated on the host.
* The non-blocking collective benchmarks can also use -t for MPI\_Test() calls and -r option for setting the target of dummy computation.

## 16. Managed Memory Extensions to OMB

In addition to the CUDA and OpenACC support, the following benchmarks provides the support for managed memory allocation. To enable this support, set the environment variable mentioned earlier. This will enable support in the MVAPICH runtime. This is currently not supported for AMD GPUs. To enable allocation of memory using the CUDA managed memory API, refer below.

* For pt2pt operations use the *M* flag
* For collective operations use the -d managed flag

16.1. Example running OSU Micro Benchmarks with Managed Memory Support

To run osu\_latency test for measuring internode MPI Send/Recv latency between GPUs, when enabling managed memory allocations on both the sender and receiver.

1: $ export MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ export MV2\_GPUDIRECT\_GDRCOPY\_LIB=/path/to/GDRCOPY/install

3: $ export MV2\_USE\_CUDA=1

4: $ export MV2\_CUDA\_ENABLE\_MANAGED=1

5:

6: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

7: $MV2\_PATH/libexec/mvapich2/get\_local\_rank \

8: $MV2\_PATH/libexec/mvapich2/osu\_latency M M

For collectives, see the example below

1: $ export MV2\_PATH=/opt/mvapich2/gdr/2.3.7/gnu

2: $ export MV2\_GPUDIRECT\_GDRCOPY\_LIB=/path/to/GDRCOPY/install

3: $ export MV2\_USE\_CUDA=1

4: $ export MV2\_CUDA\_ENABLE\_MANAGED=1

5:

6: $ $MV2\_PATH/bin/mpirun\_rsh -export -np 2 hostA hostB \

7: $MV2\_PATH/libexec/mvapich2/get\_local\_rank \

8: $MV2\_PATH/libexec/mvapich2/osu\_bcast -d managed

The following benchmarks have been extended to evaluate performance of MPI Communications to and from buffers allocated with CUDA Managed memory.

|  |  |
| --- | --- |
| osu\_bibw | Bidirectional Bandwidth Test |
| osu\_bw | Bandwidth Test |
| osu\_latency | Latency Test |
| osu\_allgather | MPI\_Allgather Latency Test |
| osu\_allgatherv | MPI\_Allgatherv Latency Test |
| osu\_allreduce | MPI\_Allreduce Latency Test |
| osu\_alltoall | MPI\_Alltoall Latency Test |
| osu\_alltoallv | MPI\_Alltoallv Latency Test |
| osu\_bcast | MPI\_Bcast Latency Test |
| osu\_gather | MPI\_Gather Latency Test |
| osu\_gatherv | MPI\_Gatherv Latency Test |
| osu\_reduce | MPI\_Reduce Latency Test |
| osu\_reduce\_scatter | MPI\_Reduce\_scatter Latency Test |
| osu\_scatter | MPI\_Scatter Latency Test |
| osu\_scatterv | MPI\_Scatterv Latency Test |

## 17. Container Support

MVAPICH2-GDR can be run on the Docker container environments. In this section, we use Docker as the example to show how to set up the network environment for it.

Prerequisites:

* + Docker: <https://www.docker.com/>
  + Nvidia Docker: <https://github.com/NVIDIA/nvidia-docker>

The following script can be used as a best practice to configure Docker container network, please edit it according to your environment

1: # This script should be executed with root privilege

2:

3: # Launch Docker container using nvidia-docker with your image

4: # Please refer to https://github.com/nvidia/nvidia-docker/wiki for more information

5:

6: # Set a hostname and assign an IP address to the launched container, in this example

7: docker\_hostname="docker-cont1"

8: docker\_ip="192.168.3.99"

9:

10: # Configure network for the Docker container

11: ifname="eth0p1"

12: cpid=`nvidia-docker inspect --format '' "$docker\_hostname"`

13:

14: # Create a macvlan interface associated with your physical interface

15: ip link add $ifname link eth0 **type** macvlan mode bridge

16:

17: # Add this interface to the container's network namespace

18: ip link **set** netns $cpid $ifname

19:

20: # Bring up the link

21: nsenter -t $cpid -n ip link **set** $ifname up

22:

23: # And configure the ip address and routing

24: nsenter -t $cpid -n ip addr add $docker\_ip/24 dev $ifname

25:

26: # Start ssh daemon

27: nvidia-docker **exec** $docker\_hostname /usr/sbin/sshd

28:

29: # Now you can attach to the Docker container and then install and run the MVAPICH2-GDR there

[1](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_footnoteref_1). True one sided implementation for transfers from/to GPU