MVAPICH2-X 2.3 User Guide

MVAPICH TEAM

NETWORK-BASED COMPUTING LABORATORY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE OHIO STATE UNIVERSITY

http://mvapich.cse.ohio-state.edu

Copyright (c) 2001-2018
Network-Based Computing Laboratory,
headed by Dr. D. K. Panda.
All rights reserved.

Last revised: September 21, 2018

Contents

1 Overview of the MVAPICH2-X Project **1**

2 Features **3**

3 Download and Installation Instructions **8**
3.1 Example downloading and installing RHEL7 package . . . 8
3.1.1 About installation . . . 8
3.1.2 Installing with local Berkeley UPC translator support . . . 8
3.1.3 Installing CAF with OpenUH Compiler . . . 8

4 Basic Usage Instructions **10**
4.1 Compile Applications . . . 10
4.1.1 Compile using `mpicc` for MPI or MPI+OpenMP Applications . . . 10
4.1.2 Compile using `oshcc` for OpenSHMEM or MPI+OpenSHMEM applications . . . 10
4.1.3 Compile using `upcc` for UPC or MPI+UPC applications . . . 10
4.1.4 Compile using `uhcaf` for CAF and MPI+CAF applications . . . 11
4.1.5 Compile using `upc++` for UPC++ or MPI+UPC++ applications . . . 11
4.2 Run Applications . . . 11
4.2.1 Run using `mpirun_rsh` . . . 12
4.2.2 Run using `oshrun` . . . 13
4.2.3 Run using `upcrun` . . . 13
4.2.4 Run using `cafrun` . . . 13
4.2.5 Running UPC++ applications using `mpirun_rsh` . . . 14
4.2.6 Run using `Hydra (mpiexec)` . . . 14

5 Advanced MPI Usage Instructions **14**
5.1 Support for User Mode Memory Registration (UMR) . . . 14
5.2 Support for Dynamic Connected Transport . . . 14
5.3 Support for Core-Direct Based Non-Blocking Collectives . . . 15
5.4 Support for OSU InfiniBand Network Analysis and Monitoring (OSU INAM) Tool . . . 16
5.5 Support for Shared Address Space based MPI Communication Using XPMEM . . . 17
5.6 Support for Efficient Asynchronous Communication Progress . . . 17
5.7 Running Collectives with Hardware based SHArP support . . . 18

6 Hybrid (MPI+PGAS) Applications **19**
6.1 MPI+OpenSHMEM Example . . . 19
6.2 MPI+UPC Example . . . 20
6.3 MPI+UPC++ Example . . . 22

7 OSU PGAS Benchmarks **24**
7.1 OSU OpenSHMEM Benchmarks . . . 24
7.2 OSU UPC Benchmarks . . . 25
7.3 OSU UPC++ Benchmarks . . . 26

8 Runtime Parameters **28**

8.1 Runtime Parameters for Dynamic Connected Transport 28
8.1.1 MV2_USE_DC 28
8.1.2 MV2_DC_KEY 28
8.1.3 MV2_NUM_DC_TGT 28
8.1.4 MV2_SMALL_MSG_DC_POOL 28
8.1.5 MV2_LARGE_MSG_DC_POOL 29
8.2 Runtime Parameters for User Mode Memory Registration 29
8.2.1 MV2_USE_UMR 29
8.2.2 MV2_NUM_UMRS 29
8.3 Core-Direct Specific Runtime Parameters 29
8.3.1 MV2_USE_CORE_DIRECT 29
8.3.2 MV2_USE_CORE_DIRECT_TUNING 30
8.3.3 MV2_USE_CD_IALLGATHER 30
8.3.4 MV2_USE_CD_IALLGATHERV 30
8.3.5 MV2_USE_CD_IALLTOALL 30
8.3.6 MV2_USE_CD_IALLTOALLV 30
8.3.7 MV2_USE_CD_IALLTOALLW 31
8.3.8 MV2_USE_CD_IBARRIER 31
8.3.9 MV2_USE_CD_IBCAST 31
8.3.10 MV2_USE_CD_IGATHER 31
8.3.11 MV2_USE_CD_IGATHERV 31
8.3.12 MV2_USE_CD_ISCATTER 32
8.3.13 MV2_USE_CD_ISCATTERV 32
8.4 Runtime Parameters for On Demand Paging 32
8.4.1 MV2_USE_ODP 32
8.4.2 MV2_USE_ODP_PREFETCH 32
8.5 CMA Collective Specific Runtime Parameters 32
8.5.1 MV2_USE_CMA_COLL 33
8.5.2 MV2_CMA_COLL_THRESHOLD 33
8.5.3 MV2_USE_CMA_COLL_ALLGATHER 33
8.5.4 MV2_USE_CMA_COLL_ALLTOALL 33
8.5.5 MV2_USE_CMA_COLL_GATHER 33
8.5.6 MV2_USE_CMA_COLL_SCATTER 34
8.6 UPC Runtime Parameters 34
8.6.1 UPC_SHARED_HEAP_SIZE 34
8.7 OpenSHMEM Runtime Parameters 34
8.7.1 OOSHM_USE_SHARED_MEM 34
8.7.2 OOSHM_SYMMETRIC_HEAP_SIZE 34
8.7.3 OSHM_USE_CMA 34
8.8 OSU INAM Specific Runtime Parameters 35
8.8.1 MV2_TOOL_INFO_FILE_PATH 35
8.8.2 MV2_TOOL_QPN 35
8.8.3 MV2_TOOL_LID 35
8.8.4 MV2_TOOL_REPORT_CPU_UTIL 35
8.8.5 MV2_TOOL_REPORT_MEM_UTIL 35
8.8.6 MV2_TOOL_REPORT_IO_UTIL 36

8.8.7 MV2_TOOL_REPORT_COMM_GRID . . . 36
8.8.8 MV2_TOOL_COUNTER_INTERVAL . . . 36
8.9 Hierarchical Multi-Leader Collectives Runtime Parameters . . . 36
8.9.1 MV2_ENABLE_DPML_COLL . . . 36
8.10 XPMEM based Point-to-point Communication Runtime Parameters . . . 37
8.10.1 MV2_SMP_USE_XPMEM . . . 37
8.11 Shared Address Space based MPI Collectives Runtime Parameters . . . 37
8.11.1 MV2_USE_XPMEM_COLL . . . 37
8.11.2 MV2_XPMEM_COLL_THRESHOLD . . . 37
8.12 Runtime Parameters for Asynchronous Communication Progress . . . 38
8.12.1 MV2_ASYNC_PROGRESS . . . 38
8.12.2 MV2_OPTIMIZED_ASYNC_PROGRESS . . . 38
8.13 Runtime Parameters for Collectives with Hardware based SHArP support . . . 38
8.13.1 MV2_ENABLE_SHARP . . . 38
8.13.2 MV2_SHARP_HCA_NAME . . . 38
8.13.3 MV2_SHARP_PORT . . . 39

9 FAQ and Troubleshooting with MVAPICH2-X 40
9.1 General Questions and Troubleshooting . . . 40
9.1.1 Compilation Errors with `upcc` . . . 40
9.1.2 Unresponsive `upcc` . . . 40
9.1.3 Shared memory limit for OpenSHMEM / MPI+OpenSHMEM programs . . . 40
9.1.4 Collective scratch size in UPC++ . . . 41
9.1.5 Install MVAPICH2-X to a specific location . . . 41
9.1.6 XPMEM based Collectives Performance Issue . . . 41

1 Overview of the MVAPICH2-X Project

Message Passing Interface (MPI) has been the most popular programming model for developing parallel scientific applications. Partitioned Global Address Space (PGAS) programming models are an attractive alternative for designing applications with irregular communication patterns. They improve programmability by providing a shared memory abstraction while exposing locality control required for performance. It is widely believed that hybrid programming model (MPI+X, where X is a PGAS model) is optimal for many scientific computing problems, especially for exascale computing.

MVAPICH2-X provides advanced MPI features/support (such as User Mode Memory Registration (UMR), On-Demand Paging (ODP), Dynamic Connected Transport (DC), Core-Direct, SHARP, and XPMEM). It also provides support for the OSU InfiniBand Network Analysis and Monitoring (OSU INAM) Tool.

It also provides a unified high-performance runtime that supports both MPI and PGAS programming models on InfiniBand clusters. It enables developers to port parts of large MPI applications that are suited for PGAS programming model. This minimizes the development overheads that have been a huge deterrent in porting MPI applications to use PGAS models. The unified runtime also delivers superior performance compared to using separate MPI and PGAS libraries by optimizing use of network and memory resources. The DCT support is also available for the PGAS models.

MVAPICH2-X supports Unified Parallel C (UPC) OpenSHMEM, CAF, and UPC++ as PGAS models. It can be used to run pure MPI, MPI+OpenMP, pure PGAS (UPC/OpenSHMEM/CAF/UPC++) as well as hybrid MPI(+OpenMP) + PGAS applications. MVAPICH2-X derives from the popular MVAPICH2 library and inherits many of its features for performance and scalability of MPI communication. It takes advantage of the RDMA features offered by the InfiniBand interconnect to support UPC/OpenSHMEM/CAF/UPC++ data transfer and OpenSHMEM atomic operations. It also provides a high-performance shared memory channel for multi-core InfiniBand clusters.

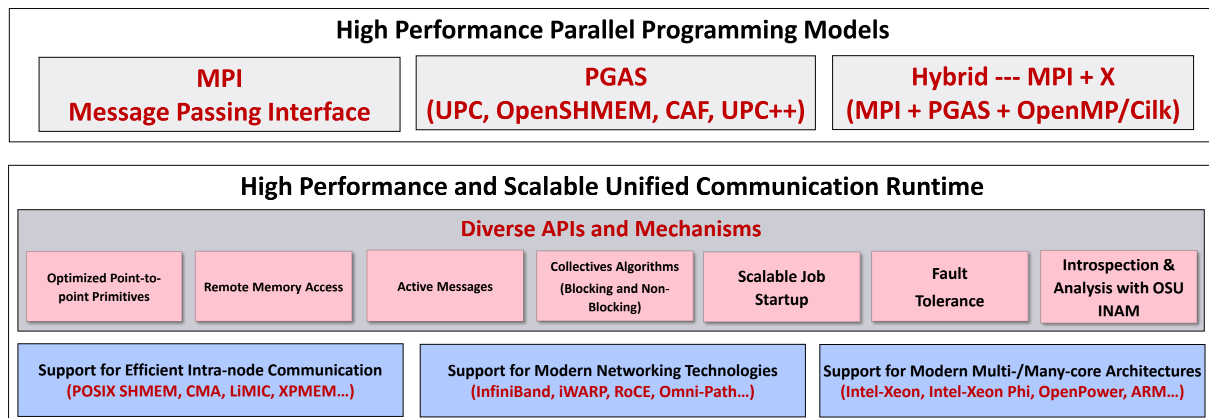

Figure 1: MVAPICH2-X Architecture

The MPI implementation of MVAPICH2-X is based on MVAPICH2, which supports MPI-3 features. The UPC implementation is UPC Language Specification 1.2 standard compliant and is based on Berkeley UPC 2.20.2 . OpenSHMEM implementation is OpenSHMEM 1.3 standard compliant and is based

on OpenSHMEM Reference Implementation 1.3 . CAF implementation is Coarray Fortran Fortran 2015 standard compliant and is based on UH CAF Reference Implementation 3.0.39 .

The current release supports InfiniBand transport interface (inter-node), and Shared Memory Interface (intra-node). The overall architecture of MVAPICH2-X is shown in the Figure 1.

This document contains necessary information for users to download, install, test, use, tune and troubleshoot MVAPICH2-X 2.3. We continuously fix bugs and update this document as per user feedback. Therefore, we strongly encourage you to refer to our web page for updates.

2 Features

MVAPICH2-X supports pure MPI programs, MPI+OpenMP programs, UPC programs, OpenSHMEM programs, CAF programs, UPC++ programs, as well as hybrid MPI(+OpenMP) + PGAS(UPC / OpenSHMEM / CAF / UPC++) programs. Current version of MVAPICH2-X 2.3 supports UPC, OpenSHMEM, CAF and UPC++ as the PGAS model. High-level features of MVAPICH2-X 2.3 are listed below. New features compared to 2.2 are indicated as (NEW).

MPI Features

- Support for MPI-3 features
- (NEW) Based on MVAPICH2 2.3 (OFA-IB-CH3 interface). MPI programs can take advantage of all the features enabled by default in OFA-IB-CH3 interface of MVAPICH2 2.3
- (NEW) Support for ARM architecture
- (NEW) Collective tuning for ARM architecture
- (NEW) Collective tuning for Intel Skylake architecture
- (NEW) Support for Efficient Asynchronous Communication Progress
- Support for Omni-Path architecture
  - Introduction of a new PSM2 channel for Omni-Path
- Support for OpenPower architecture
  - (NEW) Improve performance for Intra- and Inter-node communication
  - (NEW) Optimized inter-node and intra-node communication
  - High performance two-sided communication scalable to multi-thousand nodes
  - Optimized collective communication operations
    - Shared-memory optimized algorithms for barrier, broadcast, reduce and allreduce operations
    - Optimized two-level designs for scatter and gather operations
    - Improved implementation of allgather, alltoall operations
  - High-performance and scalable support for one-sided communication
    - Direct RDMA based designs for one-sided communication
    - Shared memory backed Windows for One-Sided communication
    - Support for truly passive locking for intra-node RMA in shared memory backed windows
  - Multi-threading support
    - Enhanced support for multi-threaded MPI applications

(NEW) MPI Advanced Features

- (NEW) Support Data Partitioning-based Multi-Leader Design (DPML) for MPI collectives (OFA-IB-CH3, PSM-CH3, and PSM2-CH3 interfaces)
- (NEW) Support Contention Aware Kernel-Assisted MPI collectives (OFA-IB-CH3, PSM-CH3, and PSM2-CH3 interfaces)
- (NEW) Support for Shared Address Space based MPI Communication Using XPMEM
  - Support for pt-to-pt communication
  - Support for Reduce and Allreduce collectives
- Support for Dynamically Connected (DC) transport protocol
  - Support for pt-to-pt, RMA and collectives
- Support for Hybrid mode with RC/DC/UD/XRC
- Support User Mode Memory Registration (UMR) for
- Efficient support for On Demand Paging (ODP) feature of Mellanox for point-to-point and RMA operations
- Support for Core-Direct based Non-blocking collectives
  - Support available for Ibcast, Ibarrier, Iscatter, Iscatterv, Igather, Igatherv, Ialltoall, Ialltoallv, Ialltoallw, Iallgather and Iallgatherv
- (NEW) Support for Efficient Asynchronous Communication Progress

Unified Parallel C (UPC) Features

- UPC Language Specification 1.2 standard compliance
- Based on Berkeley UPC v2.20.2 (contains changes/additions in preparation for UPC 1.3 specification)
- Support for OpenPower architecture
- Support for Intel Knights Landing architecture
  - Optimized inter-node and intra-node communication
- Optimized RDMA-based implementation of UPC data movement routines
- Improved UPC memput design for small/medium size messages
- Support for GNU UPC translator
- Optimized UPC collectives (Improved performance for upc_all_broadcast, upc_all_scatter, upc_all_gather, upc_all_gather_all, and upc_all_exchange)
- Support for RoCE (v1 and v2)

- Support for Dynamically Connected (DC) transport protocol

OpenSHMEM Features

- (NEW) Based on OpenSHMEM v1.3 reference implementation
- (NEW) Support Non-Blocking remote memory access routines
- OpenSHMEM 1.3 standard compliance
- Support for OpenPower architecture
- Support for Intel Knights Landing architecture
  - Optimized inter-node and intra-node communication
- Based on OpenSHMEM reference implementation v1.0h
- Support for on-demand establishment of connections
- Improved job start up and memory footprint
- Optimized RDMA-based implementation of OpenSHMEM data movement routines
- Support for OpenSHMEM 'shmem_ptr' functionality
- Support for RoCE (v1 and v2)
- Support for Dynamically Connected (DC) transport protocol
- Efficient implementation of OpenSHMEM atomics using RDMA atomics
- Optimized OpenSHMEM put routines for small/medium message sizes
- Optimized OpenSHMEM collectives (Improved performance for shmem_collect, shmem_fcollect, shmem_barrier, shmem_reduce and shmem_broadcast)
- Optimized 'shmalloc' routine
- Improved intra-node communication performance using shared memory and CMA designs

CAF Features

- Based on University of Houston CAF implementation
- Efficient point-point read/write operations
- Efficient CO_REDUCE and CO_BROADCAST collective operations
- Support for RoCE (v1 and v2)
- Support for Dynamically Connected (DC) transport protocol

- Support for Intel Knights Landing architecture

UPC++ Features

- Based on Berkeley UPC++ 0.1
- Support for OpenPower architecture
- Support for Intel Knights Landing architecture
- Asynchronous task based execution
- Multi-dimensional arrays library
- Introduce UPC++ level support for new scatter collective operation (upcxx_scatter)
- Optimized UPC collectives (improved performance for upcxx_reduce, upcxx_bcast, upcxx_gather, upcxx_allgather, upcxx_alltoall)
- Support for RoCE (v1 and v2)
- Support for Dynamically Connected (DC) transport protocol

Hybrid Program Features

- Supports hybrid programming using MPI(+OpenMP), MPI(+OpenMP)+UPC, MPI(+UPC++), MPI(+OpenMP)+OpenSHMEM and MPI(+OpenMP)+CAF
- Support for OpenPower architecture
- Support for Intel Knights Landing architecture for MPI+PGAS applications
  - Optimized inter-node and intra-node communication
- Compliance to MPI-3, UPC 1.2, OpenSHMEM 1.3 and CAF Fortran 2015 standards
- Optimized network resource utilization through the unified communication runtime
- Efficient deadlock-free progress of MPI and UPC/OpenSHMEM/CAF/UPC++ calls

Unified Runtime Features

- (NEW) Based on MVAPICH2 2.3 (OFA-IB-CH3 interface). All the runtime features enabled by default in OFA-IB-CH3 interface of MVAPICH2 2.3 are available in MVAPICH2-X 2.3rc1. MPI, UPC, OpenSHMEM, CAF, UPC++ and Hybrid programs benefit from its features listed below
  - Scalable inter-node communication with highest performance and reduced memory usage
    * Integrated RC/XRC design to get best performance on large-scale systems with reduced/ constant memory footprint

    * RDMA Fast Path connections for efficient small message communication
    * Shared Receive Queue (SRQ) with flow control to significantly reduce memory footprint of the library.
    * AVL tree-based resource-aware registration cache
    * Automatic tuning based on network adapter and host architecture
  - (NEW) The advanced MPI features listed in Section "MPI Advanced Features" are available with the unified runtime
  - Optimized intra-node communication support by taking advantage of shared-memory communication
    * Efficient Buffer Organization for Memory Scalability of Intra-node Communication
    * Automatic intra-node communication parameter tuning based on platform
  - Flexible CPU binding capabilities
    * (NEW) Portable Hardware Locality (hwloc v1.11.9) support for defining CPU affinity
    * Efficient CPU binding policies (bunch and scatter patterns, socket and numanode granularity) to specify CPU binding per job for modern multi-core platforms
    * Allow user-defined flexible processor affinity
  - Two modes of communication progress
    * Polling
    * Blocking (enables running multiple processes/processor)

- Flexible process manager support
  - Support for mpirun_rsh, hydra, upcrun and oshrun process managers

Support for OSU InfiniBand Network Analysis and Management (OSU INAM) Tool v0.9.3

- Capability to profile and report process to node communication matrix for MPI processes at user specified granularity in conjunction with OSU INAM
- Capability to classify data flowing over a network link at job level and process level granularity in conjunction with OSU INAM
- Capability to profile and report node-level, job-level and process-level activities for MPI communication in conjunction with OSU INAM (pt-to-pt, collectives and RMA) at user specified granularity
- Capability to profile and report the following parameters of MPI processes at node-level, job-level and process-level at user specified granularity in conjunction with OSU INAM
  - CPU Utilization
  - Memory Utilization
  - Inter-node communication buffer usage for RC transport
  - Inter-node communication buffer usage for UD transport

3 Download and Installation Instructions

The MVAPICH2-X package can be downloaded from here. Select the link for your distro. All MVAPICH2-X RPMs are relocatable.

3.1 Example downloading and installing RHEL7 package

Below are the steps to download MVAPICH2-X RPMs for RHEL7:

```
$ wget http://mvapich.cse.ohio-state.edu/download/mvapich/mv2x/mvapich2-x-mo
$ tar mvapich2-x-mofed3.4-gnu4.8.5-2.3rc1-1.el7.tgz
$ cd mvapich2-x-mofed3.4-gnu4.8.5-2.3rc1-1.el7
$ ./install.sh <distro> <pkgtype> <pkg_type>
```

3.1.1 About installation

Running the `install.sh` script will install the software in /opt/mvapich2-x. The /opt/mvapich2-x/ directory contains the software built using gcc distributed with RHEL7 and RHEL6. The install.sh script runs the near equivalent of the following command:

```
rpm -Uvh --nodeps *.rpm
```

This will upgrade any prior versions of MVAPICH2-X that may be present. These RPMs are relocatable and advanced users may skip the install.sh script to directly use alternate commands to install the desired RPMs.

3.1.2 Installing with local Berkeley UPC translator support

By default, MVAPICH2-X UPC uses the online UPC-to-C translator as the Berkeley UPC does. If your install environment cannot access the Internet, `upcc` will not work. In this situation, a local translator should be installed. The local Berkeley UPC-to-C translator can be downloaded from http://upc.lbl.gov/download/. After installing it, you should edit the `upcc` configure file (`/opt/mvapich2-x/gnu/etc/upcc.conf` or `$HOME/.upccrc`), and set the `translator` option to be the path of the local translator (e.g. `/usr/local/berkeley_upc_translator-<VERSION>/targ`).

3.1.3 Installing CAF with OpenUH Compiler

The CAF implementation of MVAPICH2-X is based on the OpenUH CAF compiler. Thus an installation of OpenUH compiler is needed. Here are the detailed steps to build CAF support in MVAPICH2-X:

Installing OpenUH Compiler

- `$ mkdir openuh-install; cd openuh-install`

- `$ wget http://web.cs.uh.edu/~openuh/download/packages/openuh-3.0.39-x86_64-bin.tar.bz2`
- `$ tar xjf openuh-3.0.39-x86_64-bin.tar.bz2`
- Export the PATH of OpenUH (/openuh-install/openuh-3.0.39/bin) into the environment.

Installing MVAPICH2-X CAF

- Install the MVAPICH2-X RPMs (simply use install.sh, and it will be installed in /opt/mvapich2-x)

Copy the libcaf directory from MVAPICH2-X into OpenUH

- `$ cp -a /opt/mvapich2-x/gnu/lib64/gcc-lib /openuh-install/openuh-3.0.39/lib`

Please email us at `mvapich-help@cse.ohio-state.edu` if your distro does not appear on the list or if you experience any trouble installing the package on your system.

4 Basic Usage Instructions

4.1 Compile Applications

MVAPICH2-X supports MPI applications, PGAS (OpenSHMEM, UPC, CAF, UPC++) applications and hybrid (MPI+ OpenSHMEM, MPI+UPC, MPI+CAF or MPI+UPC++) applications. User should choose the corresponding compilers according to the applications. These compilers (`oshcc`, `upcc`, `uhcaf`, `upc++` and `mpicc`) can be found under `<MVAPICH2-X_INSTALL>`
`/bin` folder.

4.1.1 Compile using mpicc for MPI or MPI+OpenMP Applications

Please use `mpicc` for compiling MPI and MPI+OpenMP applications. Below are examples to build MPI applications using `mpicc`:

```
$ mpicc -o test test.c
```

This command compiles `test.c` program into binary execution file `test` by `mpicc`.

```
$ mpicc -fopenmp -o hybrid mpi_openmp_hybrid.c
```

This command compiles a MPI+OpenMP program `mpi_openmp_hybrid.c` into binary execution file `hybrid` by `mpicc`, when MVAPICH2-X is built with GCC compiler. For Intel compilers, use `-openmp` instead of `-fopenmp`; For PGI compilers, use `-mp` instead of `-fopenmp`.

4.1.2 Compile using oshcc for OpenSHMEM or MPI+OpenSHMEM applications

Below is an example to build an MPI, an OpenSHMEM or a hybrid application using `oshcc`:

```
$ oshcc -o test test.c
```

This command compiles `test.c` program into binary execution file `test` by `oshcc`.

For MPI+OpenMP hybrid programs, add compile flags `-fopenmp`, `-openmp` or `-mp` according to different compilers, as mentioned in `mpicc` usage examples.

4.1.3 Compile using upcc for UPC or MPI+UPC applications

Below is an example to build a UPC or a hybrid MPI+UPC application using `upcc`:

```
$ upcc -o test test.c
```

This command compiles `test.c` program into binary execution file `test` by `upcc`.

Note: (1) The UPC compiler generates the following warning if MPI symbols are found in source code.
`upcc: warning: 'MPI_*' symbols seen at link time: should you be using '--uses-mpi'` This warning message can be safely ignored.

(2) `upcc` requires a C compiler as the back-end, whose version should be as same as the compiler used by MVAPICH2-X libraries. Take the MVAPICH2-X RHEL7 GCC RPMs for example, the C compiler should be GCC 4.8.5. You need to install this version of GCC before using `upcc`.

4.1.4 Compile using `uhcaf` for CAF and MPI+CAF applications

Below is an example to build an MPI, a CAF or a hybrid application using `uhcaf`:

Download the UH test example

- `$ wget http://web.cs.uh.edu/ openuh/download/packages/caf-runtime-3.0.39-src.tar.bz2`
- `$ tar xjf caf-runtime-3.0.39-src.tar.bz2`
- `$ cd caf-runtime-3.0.39/regression-tests/cases/singles/should-pass`

Compilation using uhcaf

To take advantage of MVAPICH2-X, user needs to specify the MVAPICH2-X as the conduit during the compilation.

- `$ uhcaf --layer=gasnet-mvapich2x -o event_test event_test.caf`

4.1.5 Compile using `upc++` for UPC++ or MPI+UPC++ applications

Below is an example to build a UPC++ or a hybrid MPI+UPC++ application using `upc++`:

```
$ upc++ -o test test.cpp
```

This command compiles `test.cpp` program into binary execution file `test` by `upc++`.

(2) In order to use complete set of features provided by UPC++, a C++ compiler is required as the back-end, whose version should be as same as the compiler used by MVAPICH2-X libraries. Take the MVAPICH2-X RHEL7 GCC RPMs for example, the C compiler should be GCC 4.8.5. You need to install this version of GCC before using `upc++`.

4.2 Run Applications

This section provides instructions on how to run applications with MVAPICH2. Please note that on new multi-core architectures, process-to-core placement has an impact on performance. MVAPICH2-X inherits its process-to-core binding capabilities from MVAPICH2. Please refer to (MVAPICH2 User Guide) for process mapping options on multi-core nodes.

4.2.1 Run using `mpirun_rsh`

The MVAPICH team suggests users using this mode of job start-up. `mpirun_rsh` *provides fast and scalable job start-up. It scales to multi-thousand node clusters. It can be use to launch MPI, OpenSHMEM, UPC, CAF and hybrid applications.*

Prerequisites:

- Either `ssh` or `rsh` should be enabled between the front nodes and the computing nodes. In addition to this setup, you should be able to login to the remote nodes without any password prompts.
- All host names should resolve to the same IP address on all machines. For instance, if a machine's host names resolves to 127.0.0.1 due to the default /etc/hosts on some Linux distributions it leads to incorrect behavior of the library.

Jobs can be launched using `mpirun_rsh` by specifying the target nodes as part of the command as shown below:

```
$ mpirun_rsh -np 4 n0 n0 n1 n1 ./test
```

This command launches `test` on nodes n0 and n1, two processes per node. By default `ssh` is used.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 ./test
```

This command launches `test` on nodes n0 and n1, two processes per each node using `rsh` instead of `ssh`. The target nodes can also be specified using a hostfile.

```
$ mpirun_rsh -np 4 -hostfile hosts ./test
```

The list of target nodes must be provided in the file `hosts` one per line. MPI or OpenSHMEM ranks are assigned in order of the hosts listed in the hosts file or in the order they are passed to mpirun_rsh. i.e., if the nodes are listed as n0 n1 n0 n1, then n0 will have two processes, rank 0 and rank 2; whereas n1 will have rank 1 and 3. This rank distribution is known as "cyclic". If the nodes are listed as n0 n0 n1 n1, then n0 will have ranks 0 and 1; whereas n1 will have ranks 2 and 3. This rank distribution is known as "block".

The mpirun_rsh hostfile format allows users to specify a multiplier to reduce redundancy. It also allows users to specify the HCA to be used for communication. The multiplier allows you to save typing by allowing you to specify blocked distribution of MPI ranks using one line per hostname. The HCA specification allows you to force an MPI rank to use a particular HCA. The optional components are delimited by a ':'. Comments and empty lines are also allowed. Comments start with '#' and continue to the next newline. Below are few examples of hostfile formats:

```
$ cat hosts
# sample hostfile for mpirun_rsh
host1          # rank 0 will be placed on host1
host2:2        # rank 1 and 2 will be placed on host 2
host3:hca1     # rank 3 will be on host3 and will use hca1
host4:4:hca2   # ranks 4 through 7 will be on host4 and use hca2
```

```
# if the number of processes specified for this job is greater than 8
# then the additional ranks will be assigned to the hosts in a cyclic
# fashion.  For example, rank 8 will be on host1 and ranks 9 and 10
# will be on host2.
```

Many parameters of the MPI library can be configured at run-time using environmental variables. In order to pass any environment variable to the application, simply put the variable names and values just before the executable name, like in the following example:

```
$ mpirun_rsh -np 4 -hostfile hosts ENV1=value ENV2=value ./test
```

Note that the environmental variables should be put immediately before the executable. Alternatively, you may also place environmental variables in your shell environment (e.g. `.bashrc`). These will be automatically picked up when the application starts executing.

4.2.2 Run using `oshrun`

MVAPICH2-X provides oshrun and can be used to launch applications as shown below.

```
$ oshrun -np 2 ./test
```

This command launches two processes of `test` on the localhost. A list of target nodes where the processes should be launched can be provided in a hostfile and can be used as shown below. The oshrun hostfile can be in one of the two formats outlined for mpirun_rsh earlier in this document.

```
$ oshrun -f hosts -np 2 ./test
```

4.2.3 Run using `upcrun`

MVAPICH2-X provides `upcrun` to launch UPC and MPI+UPC applications. To use upcrun, we suggest users to set the following environment:

```
$ export MPIRUN_CMD='<path-to-MVAPICH2-X-install>/bin/mpirun_rsh
-np %N -hostfile hosts %P %A'
```

A list of target nodes where the processes should be launched can be provided in the hostfile named as “hosts”. The hostfile “hosts” should follow the same format for `mpirun_rsh`, as described in Section 4.2.1. Then `upcrun` can be used to launch applications as shown below.

```
$ upcrun -n 2 ./test
```

4.2.4 Run using `cafrun`

Similar to UPC and OpenSHMEM, to run a CAF application, we can use *cafrun* or *mpirun_rsh*:

- Export the PATH and LD_LIBRARY_PATH of the GNU version of MVAPICH2-X (/opt/mvapich2-x/gnu/bin, /opt/mvapich2-x/gnu/lib64) into the environment.

- `$ export UHCAF_LAUNCHER_OPTS="-hostfile hosts"`
- `$ cafrun -n 16 -v ./event_test`

4.2.5 Running UPC++ applications using mpirun_rsh

To run a UPC++ application we need *mpirun_rsh*:

- Export the PATH and LD_LIBRARY_PATH of the GNU version of MVAPICH2-X (/opt/mvapich2-x/gnu/bin, /opt/mvapich2-x/gnu/lib64) into the environment.
- `$ mpirun_rsh -n 16 -hostfile <hosts> ./hello`

4.2.6 Run using Hydra (mpiexec)

MVAPICH2-X also distributes the Hydra process manager along with with mpirun_rsh. Hydra can be used either by using `mpiexec` or `mpiexec.hydra`. The following is an example of running a program using it:

```
$ mpiexec -f hosts -n 2 ./test
```

This process manager has many features. Please refer to the following web page for more details.

http://wiki.mcs.anl.gov/mpich2/index.php/Using_the_Hydra_Process_Manager

5 Advanced MPI Usage Instructions

5.1 Support for User Mode Memory Registration (UMR)

Support for the User Mode Memory Registration of InfiniBand is available for Mellanox ConnectIB (Dual-FDR) and ConnectX-4 (EDR) adapters. This features requires Mellanox OFED version 2.4 (or higher) and supported IB HCAs listed above. Note that users should be using the appropriate version of the MVAPICH2-X RPM built with the support for advanced features to use this. Please refer to Section 8.2 of the userguide for a detailed description of the UMR related runtime parameters.

This command launches `test` on nodes n0 and n1, two processes per node with support for User Mode Memory Registration.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 MV2_USE_UMR=1
./test
```

5.2 Support for Dynamic Connected Transport

Support for the Dynamic Connected transport of InfiniBand is available for Mellanox ConnectIB (Dual-FDR) and ConnectX-4 (EDR) adapters. This features requires Mellanox OFED version 2.4 (or higher) and

supported IB HCAs listed above. It also automatically enables the Shared Receive Queue (SRQ) feature available in MVAPICH2-X. Note that users should be using the appropriate version of the MVAPICH2-X RPM built with the support for advanced features to use this. Please refer to Section 8.1 of the userguide for a detailed description of the DC related runtime parameters.

This command launches `test` on nodes n0 and n1, two processes per node with support for Dynamic Connected Transport.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 MV2_USE_DC=1
./test
```

This command launches `test` on nodes n0 and n1, two processes per node with support for Dynamic Connected Transport with different number of DC initiator objects for small messages and large messages.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 MV2_USE_DC=1
MV2_SMALL_MSG_DC_POOL=10 MV2_LARGE_MSG_DC_POOL=10
./test
```

5.3 Support for Core-Direct Based Non-Blocking Collectives

The Mellanox ConnectX and ConnectIB series of InfiniBand HCAs provides support for offloading entire collective communication operations. These features are exposed to the user through the interfaces provided in the latest Mellanox OFED drivers. MVAPICH2-X takes advantage of such capabilities to provide hardware based offloading support for MPI-3 non-blocking collective operations. This allows for better computation and communication overlap for MPI and hybrid MPI+PGAS applications. MVAPICH2-X 2.3rc1 offers Core-Direct based support for the following collective operations - MPI_Ibcast, MPI_Iscatter, MPI_Iscatterv, MPI_Igather, MPI_Igatherv, MPI_Iallgather, MPI_Iallgatherv, MPI_Ialltoall, MPI_Ialltoallv, MPI_Ialltoallw, and MPI_Ibarrier. Note that users should be using the appropriate version of the MVAPICH2-X RPM built with the support for advanced features to use this. Please refer to Section 8.9 of the userguide for a detailed description of the Core-Direct related runtime parameters.

The Core-Direct feature and can be enabled or disabled globally by the use of the environment variable *MV2_USE_CORE_DIRECT*.

This command launches `test` on nodes n0 and n1, two processes per node with support for Core-Direct based non-blocking collectives.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1
MV2_USE_CORE_DIRECT=1 ./test
```

By default, when Core-Direct capabilities are turned on using the above variable, all supported non-blocking collectives listed above leverage the feature. To specifically toggle Core-Direct capabilities on a per collective basis, the following environment variables may be used:

```
MV2_USE_CD_IBCAST=0         #Disables Core-Direct support for MPI_Ibcast
MV2_USE_CD_ISCATTER=0       #Disables Core-Direct support for MPI_Iscatter
MV2_USE_CD_ISCATTERV=0      #Disables Core-Direct support for MPI_Iscatterv
MV2_USE_CD_IGATHER=0        #Disables Core-Direct support for MPI_Igather
MV2_USE_CD_IGATHERV=0       #Disables Core-Direct support for MPI_Igatherv
MV2_USE_CD_IALLGATHER=0     #Disables Core-Direct support for MPI_Iallgather
```

```
MV2_USE_CD_IALLGATHERV=0   #Disables Core-Direct support for MPI_Iallgatherv
MV2_USE_CD_IALLTOALL=0     #Disables Core-Direct support for MPI_Ialltoall
MV2_USE_CD_IALLTOALLV=0    #Disables Core-Direct support for MPI_Ialltoallv
MV2_USE_CD_IALLTOALLW=0    #Disables Core-Direct support for MPI_Ialltoallw
MV2_USE_CD_IBARRIER=0      #Disables Core-Direct support for MPI_Ibarrier
```

This command launches `test` on nodes n0 and n1, two processes per node with Core-Direct based non-blocking collectives support for all non-blocking collectives listed above except non-blocking broadcast.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1
MV2_USE_CORE_DIRECT=1 MV2_USE_CD_IBCAST=0 ./test
```

5.4 Support for OSU InfiniBand Network Analysis and Monitoring (OSU INAM) Tool

The OSU InfiniBand Network Analysis and Monitoring tool - *OSU INAM* monitors IB clusters in real time by querying various subnet management entities in the network. It is also capable of interacting with the MVAPICH2-X software stack to gain insights into the communication pattern of the application and classify the data transferred into Point-to-Point, Collective and Remote Memory Access (RMA). OSU INAM can also remotely monitoring the CPU utilization of MPI processes in conjunction with MVAPICH2-X. Note that users should be using the appropriate version of the MVAPICH2-X RPM built with the support for advanced features to use this. In this section, we detail how one should enable MVAPICH2-X to work in conjunction with OSU INAM.

Please note that MVAPICH2-X must be launched with support for on-demand connection management when running in conjunction with OSU INAM. One can achieve this by setting the MV2_ON_DEMAND_THRESHOLD environment variable to a value less than the number of processes in the job.

Please refer to the *Tools* page in the MVAPICH website (http://mvapich.cse.ohio-state.edu/tools/osu-inam) for more information on how to download, install and run OSU INAM. Please refer to Section 8.8 of the userguide for a detailed description of the OSU INAM related runtime parameters.

This command launches `test` on nodes n0 and n1, two processes per node with support for sending the process and node level information to the OSU INAM daemon.

```
$ mpirun_rsh -rsh -np 4 n0 n0
n1 n1 MV2_ON_DEMAND_THRESHOLD=1
MV2_TOOL_INFO_FILE_PATH=/opt/inam/.mv2-tool-mvapich2.conf
./test
```

```
$ cat /opt/inam/.mv2-tool-mvapich2.conf
MV2_TOOL_QPN=473                 #UD QPN at which OSU INAM is listening.
MV2_TOOL_LID=208                 #LID at which OSU INAM is listening.
MV2_TOOL_COUNTER_INTERVAL=30 #Specifies whether MVAPICH2-X should report
                                 #process level CPU utilization information.
MV2_TOOL_REPORT_CPU_UTIL=1       #The interval at which MVAPICH2-X should
                                 #report node, job and process level information.
```

5.5 Support for Shared Address Space based MPI Communication Using XPMEM

MVAPICH2-X supports shared address-space based efficient MPI intra-node communication. This feature requires XPMEM `https://github.com/hjelmn/xpmem` kernel module. Currently, it provides support for efficient intra-node rendezvous communication and zero-copy MPI reduction operations. Note that the users should be using the appropriate version of the MVAPICH2-X RPM built with the support for XPMEM to use this. Please refer to Section 8.10.1 and Section 8.11.1 of the userguide for a detailed description of the XPMEM based point-to-point and collective communication support in MVAPICH2-X.

XPMEM kernels module can be installed by following the instructions given below:

```
1. $ ./configure --prefix=/opt/xpmem \
            --with-default-prefix=/opt/xpmem \
            --with-module=/opt/xpmem/share/modules/xpmem
2. $ make -j 8
3. $ sudo make install
4. $ sudo insmod /opt/xpmem/lib/module/xpmem.ko ; \
       sleep 1 ; \
       sudo chmod 666 /dev/xpmem
5. $ make check

6. After installing in /opt, just load the module (step-4 only) on the
desired set of nodes.
```

The following command launches `osu_latency` with two MPI processes on a single node and uses XPMEM based rendezvous communication mechanism.

```
$ mpirun_rsh -np 2 n0 n1 MV2_SMP_USE_XPMEM=1 ./osu_latency
```

The following command launches `osu_reduce` on nodes n0 and n1, two processes per node with support for XPMEM based MPI_Reduce for message sizes greater than 4KB.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 MV2_USE_XPMEM_COLL=1
MV2_XPMEM_COLL_THRESHOLD=4096 ./osu_reduce
```

Note: XPMEM has a known issue where a remote memory segment that is “attached” using XPMEM, can not be registered with InfiniBand device. In some cases, if severe performance degradation is observed when using XPMEM based collectives, additionally setting following two parameters can circumvent the issue.

```
MV2_IBA_EAGER_THRESHOLD=262144 MV2_VBUF_TOTAL_SIZE=262144
```

5.6 Support for Efficient Asynchronous Communication Progress

MVAPICH2-X provides an optimized asynchronous progress to overlap computation and communication in HPC applications. This design is recommended over thread-based asynchronous progress design in MPICH. The optimized asynchronous progress design enables applications to use all available cores for compute without sparing dedicated cores for asynchronous progress threads. Applications which use large (greater

than eager threshold) non-blocking messages are suggested to use asynchronous progress design to gain better overlap of computation and communication.

The optimized version of asynchronous communication progress design is enabled by using environment variable MV2_OPTIMIZED_ASYNC_PROGRESS=1. Note that the optimized support needs the user to enable basic asynchronous progress support by setting MV2_ASYNC_PROGRESS=1.

This command launches test on nodes n0 and n1, two processes per node with support for Asynchronous Progress design.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 MV2_ASYNC_PROGRESS=1
MV2_OPTIMIZED_ASYNC_PROGRESS=1 MV2_CPU_BINDING_POLICY=hybrid ./test
```

5.7 Running Collectives with Hardware based SHArP support

In MVAPICH2, support for SHArP-based collectives has been enabled for MPI applications running over OFA-IB-CH3 interface. Currently, this support is available for the following collective operations:

- MPI_Allreduce

This feature is turned off by default at runtime and can be turned on at runtime by using parameter MV2_ENABLE_SHARP=1 (8.13.1).

Note that the various SHArP related daemons (SHArP Aggregation Manager - *sharpam* and the local SHArP daemon - *sharpd* must be installed, setup and running for SHArP support in MVAPICH2 to work properly. This can be verified by running the `sharp_hello` program available in the “bin” sub-folder of the SHArP installation directory.

When using HPCX v1.7, we recommend setting SHARP_COLL_ENABLE_GROUP_TRIM=0 environment variable. Note that if you are using `mpirun_rsh` you need to add `-export` option to make sure that environment variable is exported correctly. This environment variable is a part of SHArP library. Please refer to SHArP userguide for more information.

6 Hybrid (MPI+PGAS) Applications

MVAPICH2-X supports hybrid programming models. Applications can be written using both MPI and PGAS constructs. Rather than using a separate runtime for each of the programming models, MVAPICH2-X supports hybrid programming using a unified runtime and thus provides better resource utilization and superior performance.

6.1 MPI+OpenSHMEM Example

A simple example of Hybrid MPI+OpenSHMEM program is shown below. It uses both MPI and OpenSHMEM constructs to print the sum of ranks of each processes.

```
1    #include <stdio.h>
2    #include <shmem.h>
3    #include <mpi.h>
4
5    static int sum = 0;
6    int main(int c, char *argv[])
7    {
8        int rank, size;
9
10       /* SHMEM init */
11       start_pes(0);
12
13       /* get rank and size */
14       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
15       MPI_Comm_size(MPI_COMM_WORLD, &size);
16
17       /* SHMEM barrier */
18       shmem_barrier_all();
19
20       /* fetch-and-add at root */
21       shmem_int_fadd(&sum, rank, 0);
22
23       /* MPI barrier */
24       MPI_Barrier(MPI_COMM_WORLD);
25
26       /* root broadcasts sum */
27       MPI_Bcast(&sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
28
29       /* print sum */
30       fprintf(stderr, "(%d): Sum: %d\n", rank, sum);
31
32       shmem_barrier_all();
```

```
33      return 0;
34  }
```

start_pes in line 10 initializes the runtime for MPI and OpenSHMEM communication. An explicit call to MPI_Init is not required. The program uses MPI calls MPI_Comm_rank and MPI_Comm_size to get process rank and size, respectively (lines 14-15). MVAPICH2-X assigns same rank for MPI and PGAS model. Thus, alternatively the OpenSHMEM constructs _my_pe and _num_pes can also be used to get rank and size, respectively. In line 17, every process does a barrier using OpenSHMEM construct shmem_barrier_all.

After this, every process does a fetch-and-add of the rank to the variable sum in process 0. The sample program uses OpenSHMEM construct shmem_int_fadd (line 21) for this. Following the fetch-and-add, every process does a barrier using MPI_Barrier (line 24). Process 0 then broadcasts sum to all processes using MPI_Bcast (line 27). Finally, all processes print the variable sum. Explicit MPI_Finalize is not required.

The program outputs the following for a four-process run:

```
$$> mpirun_rsh -np 4 -hostfile ./hostfile ./hybrid_mpi_shmem

(0): Sum: 6
(1): Sum: 6
(2): Sum: 6
(3): Sum: 6
```

The above sample hybrid program is available at <MVAPICH2-X_INSTALL>/<gnu|intel>/share/examples/hybrid_mpi_shmem.c

6.2 MPI+UPC Example

A simple example of Hybrid MPI+UPC program is shown below. Similarly to the previous example, it uses both MPI and UPC constructs to print the sum of ranks of each UPC thread.

```
 1 #include <stdio.h>
 2 #include <upc.h>
 3 #include <mpi.h>
 4
 5 shared [1] int A[THREADS];
 6 int main() {
 7     int sum = 0;
 8     int rank, size;
 9
10     /* get MPI rank and size */
11     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
12     MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
13
14     /* UPC barrier */
15     upc_barrier;
16
17     /* initialize global array */
18     A[MYTHREAD] = rank;
19     int *local = ((int *)(&A[MYTHREAD]));
20
21     /* MPI barrier */
22     MPI_Barrier(MPI_COMM_WORLD);
23
24     /* sum up the value with each UPC thread */
25     MPI_Allreduce(local, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
26
27     /* print sum */
28     if (MYTHREAD == 0)
29         fprintf(stderr, "(%d): Sum: %d\n", rank, sum);
30
31     upc_barrier;
32
33     return 0;
34 }
```

An explicit call to `MPI_Init` is not required. The program uses MPI calls `MPI_Comm_rank` and `MPI_Comm_size` to get process rank and size, respectively (lines 11-12). MVAPICH2-X assigns same rank for MPI and PGAS model. Thus, MYTHREAD and THREADS contains the UPC thread rank and UPC thread size respectively, which is equal to the return value of `MPI_Comm_rank` and `MPI_Comm_size`. In line 15, every UPC thread does a barrier using UPC construct `upc_barrier`.

After this, every UPC thread set its MPI rank to one element of a global shared memory array A and this element `A[MYTHREAD]` has affinity with the UPC thread who set the value of it (line 18). Then a local pointer need to be set to the global shared array element for MPI collective functions. Then every UPC thread does a barrier using `MPI_Barrier` (line 22). After the barrier, `MPI_Allreduce` is called (line 25) to sum up all the rank values and return the results to every UPC thread, in `sum` variable. Finally, all processes print the variable `sum`. Explicit `MPI_Finalize` is not required.

The program can be compiled using `upcc`:

```
$$> upcc hybrid_mpi_upc.c -o hybrid_mpi_upc
```

The program outputs the following for a four-process run:

```
$$> mpirun_rsh -n 4 -hostfile hosts ./hybrid_mpi_upc
(0): Sum: 6
(3): Sum: 6
(1): Sum: 6
(2): Sum: 6
```

The above sample hybrid program is available at `<MVAPICH2-X_INSTALL>/<gnu|intel>/share/examples/hybrid_mpi_upc.c`

6.3 MPI+UPC++ Example

A simple example of Hybrid MPI+UPC++ program is shown below. Similarly to the previous example, it uses both MPI and UPC++ constructs to reduce the sum of ranks of each UPC++ thread on all ranks. The sum will be reduced on all ranks but printed only on thread 0.

```
 1 #include <upcxx.h>
 2 #include <mpi.h>
 3
 4 using namespace upcxx;
 5
 6 int main (int argc, char** argv)
 7 {
 8    /* UPC++ init */
 9    init(&argc, &argv);
10
11    shared_array<int> A(ranks());
12    int sum = 0;
13    int rank, size;
14
15    /* get rank and size */
16    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
17    MPI_Comm_size(MPI_COMM_WORLD, &size);
18
19    /* UPC++ barrier all */
20    barrier();
21
22    /* initialize global array */
23    A[myrank()] = rank;
24
25    /* cast value from shared array to local pointer */
26    int *local = ((int *) (&A[myrank()]));
27
28    /* MPI Barrier */
29    MPI_Barrier(MPI_COMM_WORLD);
30
31    /* sum up the value with each UPC++ thread */
32    MPI_Allreduce(local, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
33
34    /* print sum */
35    if (myrank() == 0)
36        std::cout << "Rank(" << rank << "): Sum: "<< sum << std::endl;
```

```
37
38     /* UPC++ finalize */
39     finalize();
40
41     return 0;
42 }
```

An explicit call to MPI_Init is not required. However, a call to UPC++'s init is required as in line 9. The program uses MPI calls MPI_Comm_rank and MPI_Comm_size to get process rank and size, respectively (lines 16-17). MVAPICH2-X assigns same rank for MPI and PGAS model. Thus, myrank() and ranks() contains the UPC++ thread rank and UPC++ thread size respectively, which is equal to the return value of MPI_Comm_rank and MPI_Comm_size. In line 20, every UPC++ thread does a barrier by explicitly calling UPC++ function barrier().

After this, every UPC++ thread sets its MPI rank to one element of a global shared memory array A and this element A[myrank()] has affinity with the UPC++ thread who set the value of it (line 23). Then a local pointer need to be set to the global shared array element for MPI collective functions. Then every UPC++ thread does a barrier using MPI_Barrier (line 29). After the barrier, MPI_Allreduce is called (line 32) to sum up all the rank values and return the results to every UPC++ thread, in sum variable. Finally, all processes print the variable sum. Explicit MPI_Finalize is not required. However, UPC++'s finalize() function is called in the end for graceful termination of UPC++ application (line 39).

The program can be compiled using upc++:

```
$$> upc++ hybrid_mpi_upcxx.cpp -o hybrid_mpi_upcxx
```

The program outputs the following for a four-process run:

```
$$> mpirun_rsh -n 4 -hostfile hosts ./hybrid_mpi_upcxx
(0): Sum: 6
```

The above sample hybrid program is available at <MVAPICH2-X_INSTALL>/<gnu|intel>/share /examples/hybrid_mpi_upcxx.c

7 OSU PGAS Benchmarks

7.1 OSU OpenSHMEM Benchmarks

We have extended the OSU Micro Benchmark (OMB) suite with tests to measure performance of OpenSHMEM operations. OSU Micro Benchmarks (OMB-5.4.3) have OpenSHMEM data movement and atomic operation benchmarks. The complete benchmark suite is available along with MVAPICH2-X binary package, in the folder: `<MVAPICH2-X_INSTALL>/libexec/osu-micro-benchmarks`. A brief description for each of the newly added benchmarks is provided below.

Put Latency (osu_oshm_put):

This benchmark measures latency of a shmem_putmem operation for different data sizes. The user is required to select whether the communication buffers should be allocated in global memory or heap memory, through a parameter. The test requires exactly two PEs. PE 0 issues shmem_putmem to write data at PE 1 and then calls shmem_quiet. This is repeated for a fixed number of iterations, depending on the data size. The average latency per iteration is reported. A few warm-up iterations are run without timing to ignore any start-up overheads. Both PEs call shmem_barrier_all after the test for each message size.

Get Latency (osu_oshm_get):

This benchmark is similar to the one above except that PE 0 does a shmem_getmem operation to read data from PE 1 in each iteration. The average latency per iteration is reported.

Put Operation Rate (osu_oshm_put_mr):

This benchmark measures the aggregate uni-directional operation rate of OpenSHMEM Put between pairs of PEs, for different data sizes. The user should select for communication buffers to be in global memory and heap memory as with the earlier benchmarks. This test requires number of PEs to be even. The PEs are paired with PE 0 pairing with PE n/2 and so on, where n is the total number of PEs. The first PE in each pair issues back-to-back shmem_putmem operations to its peer PE. The total time for the put operations is measured and operation rate per second is reported. All PEs call shmem_barrier_all after the test for each message size.

Atomics Latency (osu_oshm_atomics):

This benchmark measures the performance of atomic fetch-and-operate and atomic operate routines supported in OpenSHMEM for the integer datatype. The buffers can be selected to be in heap memory or global memory. The PEs are paired like in the case of Put Operation Rate benchmark and the first PE in each pair issues back-to-back atomic operations of a type to its peer PE. The average latency per atomic operation and the aggregate operation rate are reported. This is repeated for each of fadd, finc, add, inc, cswap and swap routines.

Collective Latency Tests:

OSU Microbenchmarks consists of the following collective latency tests:

The latest OMB Version includes the following benchmarks for various OpenSHMEM collective operations (shmem_collect, shmem_fcollect shmem_broadcast, shmem_reduce and shmem_barrier).

- osu_oshm_collect - OpenSHMEM Collect Latency Test
- osu_oshm_fcollect - OpenSHMEM FCollect Latency Test
- osu_oshm_broadcast - OpenSHMEM Broadcast Latency Test
- osu_oshm_reduce - OpenSHMEM Reduce Latency Test
- osu_oshm_barrier - OpenSHMEM Barrier Latency Test

These benchmarks work in the following manner. Suppose users run the osu_oshm_broadcast benchmark with N processes, the benchmark measures the min, max and the average latency of the shmem_broadcast operation across N processes, for various message lengths, over a number of iterations. In the default version, these benchmarks report average latency for each message length. Additionally, the benchmarks the following options:

- "-f" can be used to report additional statistics of the benchmark, such as min and max latencies and the number of iterations
- "-m" option can be used to set the maximum message length to be used in a benchmark. In the default version, the benchmarks report the latencies for up to 1MB message lengths
- "-i" can be used to set the number of iterations to run for each message length

7.2 OSU UPC Benchmarks

OSU Microbenchmarks extensions include UPC benchmarks also. Current version (OMB-5.4.3) has benchmarks for `upc_memput` and `upc_memget`. The complete benchmark suite is available along with MVAPICH2-X binary package, in the folder: `<MVAPICH2-X_INSTALL>/libexec/osu-micro-benchmarks`. A brief description for each of the benchmarks is provided below.

Put Latency (osu_upc_memput):

This benchmark measures the latency of `upc_put` operation between multiple UPC threads. In this benchmark, UPC threads with ranks less than `(THREADS/2)` issue `upc_memput` operations to peer UPC threads. Peer threads are identified as `(MYTHREAD+THREADS/2)`. This is repeated for a fixed number of iterations, for varying data sizes. The average latency per iteration is reported. A few warm-up iterations are run without timing to ignore any start-up overheads. All UPC threads call `upc_barrier` after the test for each message size.

Get Latency (osu_upc_memget):

This benchmark is similar as the `osu_upc_put` benchmark that is described above. The difference is that the shared string handling function is `upc_memget`. The average get operation latency per iteration is reported.

Collective Latency Tests:

OSU Microbenchmarks consists of the following collective latency tests:

The latest OMB Version includes the following benchmarks for various UPC collective operations (upc_all_barrier, upc_all_broadcast, upc_all_exchange, upc_all_gather, upc_all_gather_all, upc_all_reduce, and upc_all_scatter).

- osu_upc_all_barrier - UPC Barrier Latency Test
- osu_upc_all_broadcast - UPC Broadcast Latency Test
- osu_upc_all_exchange - UPC Exchange Latency Test
- osu_upc_all_gather_all - UPC GatherAll Latency Test
- osu_upc_all_gather - UPC Gather Latency Test
- osu_upc_all_reduce - UPC Reduce Latency Test
- osu_upc_all_scatter - UPC Scatter Latency Test

These benchmarks work in the following manner. Suppose users run the osu_upc_all_broadcast with N processes, the benchmark measures the min, max and the average latency of the upc_all_broad-cast operation across N processes, for various message lengths, over a number of iterations. In the default version, these benchmarks report average latency for each message length. Additionally, the benchmarks the following options:

- "-f" can be used to report additional statistics of the benchmark, such as min and max latencies and the number of iterations
- "-m" option can be used to set the maximum message length to be used in a benchmark. In the default version, the benchmarks report the latencies for up to 1MB message lengths
- "-i" can be used to set the number of iterations to run for each message length

7.3 OSU UPC++ Benchmarks

In order to provide performance measurement of UPC++ operations, we have also extended OSU Microbenchmarks to include UPC++ based point-to-point and collectives benchmarks. These are included in current version of OMB (OMB-5.4.3). The point-to-point benchmarks include `upcxx_async_copy_put` and `upcxx_async_copy_get`. The complete benchmark suite is available along with MVAPICH2-X binary package, in the folder: `<MVAPICH2-X_INSTALL>/libexec/osu-micro-benchmarks`. A brief description for each of the benchmarks is provided below.

Put Latency (osu_upcxx_async_copy_put):

This benchmark measures the latency of `async_copy` (`memput`) operation between multiple UPC++ threads. In this benchmark, UPC++ threads with ranks less than `(ranks()/2)` copy data 'from' their local memory 'to' their peer thread's memory using `async_copy` operation. By changing the `src` and `dst` buffers in `async_copy`, we can mimic the behavior of `upc_memput` and `upc_memget`. Peer threads are identified as `(myrank()+ranks()/2)`. This is repeated for a fixed number of iterations, for varying data sizes. The average latency per iteration is reported. A few warm-up iterations are run without timing to ignore any start-up overheads. All UPC++ threads call `barrier()` function after the test for each message size.

Get Latency (osu_upcxx_async_copy_get):

Similar to `osu_upcxx_async_copy_put`, this benchmark mimics the behavior of `upc_memget` and measures the latency of `async_copy` (`memget`) operation between multiple UPC++ threads. The only difference is that the `src` and `dst` buffers in `async_copy` are swapped. In this benchmark, UPC++ threads with ranks less than `(ranks()/2)` copy data 'from' their peer thread's memory 'to' their local memory using `async_copy` operation. The rest of the details are same as discussed above. The average get operation latency per iteration is reported.

Collective Latency Tests:

OSU Microbenchmarks consists of the following collective latency tests:

The latest OMB Version includes the following benchmarks for various UPC++ collective operations (upcxx_reduce, upcxx_bcast, upcxx_gather, upcxx_allgather, upcxx_alltoall, upcxx_scatter).

- osu_upcxx_bcast - UPC++ Broadcast Latency Test
- osu_upcxx_reduce - UPC++ Reduce Latency Test
- osu_upcxx_allgather - UPC++ Allgather Latency Test
- osu_upcxx_gather - UPC++ Gather Latency Test
- osu_upcxx_scatter - UPC++ Scatter Latency Test
- osu_upcxx_alltoall - UPC++ AlltoAll (exchange) Latency Test

These benchmarks work in the following manner. Suppose users run the osu_upcxx_bcast with N processes, the benchmark measures the min, max and the average latency of the upcxx_bcast operation across N processes, for various message lengths, over a number of iterations. In the default version, these benchmarks report average latency for each message length. Additionally, the benchmarks the following options:

- "-f" can be used to report additional statistics of the benchmark, such as min and max latencies and the number of iterations
- "-m" option can be used to set the maximum message length to be used in a benchmark. In the default version, the benchmarks report the latencies for up to 1MB message lengths
- "-i" can be used to set the number of iterations to run for each message length

8 Runtime Parameters

MVAPICH2-X supports all the runtime parameters of MVAPICH2 (OFA-IB-CH3). A comprehensive list of all runtime parameters of MVAPICH2 2.3 can be found in User Guide. Runtime parameters specific to MVAPICH2-X are listed below.

8.1 Runtime Parameters for Dynamic Connected Transport

MVAPICH2-X features support for the Dynamic Connected (DC) transport protocol from Mellanox. In this section, we specify some of the runtime parameters that control these advanced features.

8.1.1 MV2_USE_DC

- Class: Run time
- Default: 0 (Unset)

Enable the use of the Dynamic Connected (DC) InfiniBand transport.

8.1.2 MV2_DC_KEY

- Class: Run time
- Default: 0

This parameter must be same across all processes that wish to communicate with each other in a job.

8.1.3 MV2_NUM_DC_TGT

- Class: Run time
- Default: 1

Controls the number of DC receive communication objects. Please note that we have extensively tuned this parameter based on job size and communication characteristics.

8.1.4 MV2_SMALL_MSG_DC_POOL

- Class: Run time
- Default: 8

Controls the number of DC send communication objects used for transmitting small messages. Please note that we have extensively tuned this parameter based on job size and communication characteristics.

8.1.5 MV2_LARGE_MSG_DC_POOL

- Class: Run time
- Default: 8

Controls the number of DC send communication objects used for transmitting large messages. Please note that we have extensively tuned this parameter based on job size and communication characteristics.

8.2 Runtime Parameters for User Mode Memory Registration

MVAPICH2-X provides support for the User Mode Memory Registration (UMR) feature from Mellanox. In this section, we specify some of the runtime parameters that control these advanced features.

8.2.1 MV2_USE_UMR

- Class: Run time
- Default: 0 (Unset)

Enable the use of User Mode Memory Registration (UMR) for high performance datatype based communication.

8.2.2 MV2_NUM_UMRS

- Class: Run time
- Default: 256

Controls the number of pre-created UMRs for non-contiguous data transfer.

8.3 Core-Direct Specific Runtime Parameters

MVAPICH2-X features support for the Core-Direct (CD) collective offload interface from Mellanox. In this section, we specify some of the runtime parameters that control these advanced features.

8.3.1 MV2_USE_CORE_DIRECT

- Class: Run time
- Default: 0 (Unset)

Enables core-direct support for non-blocking collectives.

8.3.2 MV2_USE_CORE_DIRECT_TUNING

- Class: Run time
- Default: 1 (Set)

Enables tuned version of core-direct support for non-blocking collectives that prioritizes overlap and latency based on message size.

8.3.3 MV2_USE_CD_IALLGATHER

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Allgather collective.

8.3.4 MV2_USE_CD_IALLGATHERV

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Allgatherv collective.

8.3.5 MV2_USE_CD_IALLTOALL

- Class: Run time
- Default: 1 (Unset)

Enables core-direct support for non-blocking Alltoall collective.

8.3.6 MV2_USE_CD_IALLTOALLV

- Class: Run time
- Default: 1 (Unset)

Enables core-direct support for non-blocking Alltoallv collective.

8.3.7 MV2_USE_CD_IALLTOALLW

- Class: Run time
- Default: 1 (Unset)

Enables core-direct support for non-blocking Alltoallw collective.

8.3.8 MV2_USE_CD_IBARRIER

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Barrier collective.

8.3.9 MV2_USE_CD_IBCAST

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Broadcast collective.

8.3.10 MV2_USE_CD_IGATHER

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Gather collective.

8.3.11 MV2_USE_CD_IGATHERV

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Gatherv collective.

8.3.12 MV2_USE_CD_ISCATTER

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Scatter collective.

8.3.13 MV2_USE_CD_ISCATTERV

- Class: Run time
- Default: 1 (Set)

Enables core-direct support for non-blocking Scatterv collective.

8.4 Runtime Parameters for On Demand Paging

MVAPICH2-X features support for the On Demand Paging (ODP) feature from Mellanox. In this section, we specify some of the runtime parameters that control these advanced features.

8.4.1 MV2_USE_ODP

- Class: Run time
- Default: 0 (Unset)

Enable the use of On-Demand Paging for inter-node communication

8.4.2 MV2_USE_ODP_PREFETCH

- Class: Run time
- Default: 1

Enable verbs-level prefetch operation to speed-up On-demand Paging based inter-node communication

8.5 CMA Collective Specific Runtime Parameters

MVAPICH2-X features support for contention-aware, kernel-assisted blocking collectives using Cross Memory Attach (CMA). These designs are applicable to the following interfaces: OFA-IB-CH3, OFA-IB-RoCE, PSM-CH3, and PSM2-CH3. In this section, we specify some of the runtime parameters that control this feature.

8.5.1 MV2_USE_CMA_COLL

- Class: Run time
- Default: 1 (Set)

Enables support for CMA collectives.

8.5.2 MV2_CMA_COLL_THRESHOLD

- Class: Run time
- Default: Architecture Dependent

Specifies the message size above which CMA collectives are used.

8.5.3 MV2_USE_CMA_COLL_ALLGATHER

- Class: Run time
- Default: 1 (Set)

Enables CMA collective support for Allgather.

8.5.4 MV2_USE_CMA_COLL_ALLTOALL

- Class: Run time
- Default: 1 (Set)

Enables CMA collective support for Alltoall.

8.5.5 MV2_USE_CMA_COLL_GATHER

- Class: Run time
- Default: 1 (Set)

Enables CMA collective support for Gather.

8.5.6 MV2_USE_CMA_COLL_SCATTER

- Class: Run time
- Default: 1 (Set)

Enables CMA collective support for Scatter.

8.6 UPC Runtime Parameters

8.6.1 UPC_SHARED_HEAP_SIZE

- Class: Run time
- Default: 64M

Set UPC Shared Heap Size

8.7 OpenSHMEM Runtime Parameters

8.7.1 OOSHM_USE_SHARED_MEM

- Class: Run time
- Default: 1

Enable/Disable shared memory scheme for intra-node communication.

8.7.2 OOSHM_SYMMETRIC_HEAP_SIZE

- Class: Run time
- Default: 512M

Set OpenSHMEM Symmetric Heap Size

8.7.3 OSHM_USE_CMA

- Class: Run time
- Default: 1

Enable/Disable CMA based intra-node communication design

8.8 OSU INAM Specific Runtime Parameters

8.8.1 MV2_TOOL_INFO_FILE_PATH

- Class: Run time
- Default: ".mv2-tool-mvapich2.conf"

Specifies the path to the file containing the runtime parameters to enable node level, job level and process level data collection for MVAPICH2-X. This file contains the following parameters MV2_TOOL_QPN, MV2_TOOL_LID, MV2_TOOL_REPORT_CPU_UTIL and MV2_TOOL_COUNTER_INTERVAL.

8.8.2 MV2_TOOL_QPN

- Class: Run time
- Default: 0 (Unset)

Specifies the UD QPN at which OSU INAM is listening.

8.8.3 MV2_TOOL_LID

- Class: Run time
- Default: 0 (Unset)

Specifies the IB LID at which OSU INAM is listening.

8.8.4 MV2_TOOL_REPORT_CPU_UTIL

- Class: Run time
- Default: 1 (Set)

Specifies whether MVAPICH2-X should report process level CPU utilization information.

8.8.5 MV2_TOOL_REPORT_MEM_UTIL

- Class: Run time
- Default: 1 (Set)

Specifies whether MVAPICH2-X should report process level memory utilization information.

8.8.6 MV2_TOOL_REPORT_IO_UTIL

- Class: Run time
- Default: 1 (Set)

Specifies whether MVAPICH2-X should report process level I/O utilization information.

8.8.7 MV2_TOOL_REPORT_COMM_GRID

- Class: Run time
- Default: 1 (Set)

Specifies whether MVAPICH2-X should report process to node communication information. This parameter only takes effect for multi-node runs.

8.8.8 MV2_TOOL_COUNTER_INTERVAL

- Class: Run time
- Default: 30

Specifies the interval at which MVAPICH2-X should report node, job and process level information.

8.9 Hierarchical Multi-Leader Collectives Runtime Parameters

MVAPICH2-X features support for the high-performance hierarchical multi-leader collectives designated for multi/many-core processors with Omni-Path or Infiniband network architectures. Currently, MPI_Allreduce collective operation is supported with this feature. In this section, we specify the runtime parameter that controls this advanced feature.

8.9.1 MV2_ENABLE_DPML_COLL

- Class: Run time
- Default: 1 (Set)

Enables support for hierarchical multi-leader collectives. This feature can be disabled by setting this parameter to 0.

8.10 XPMEM based Point-to-point Communication Runtime Parameters

MVAPICH2-X provides support for XPMEM based intra-node communication channel to achieve direct load/store based inter-process communication in MPI. This feature is designated for multi/many-core processors with Infiniband network architectures. In this section, we specify the runtime parameter that controls this advanced feature.

8.10.1 MV2_SMP_USE_XPMEM

- Class: Run time
- Default: 1 (Set)

Enables support for XPMEM based intra-node communication. This feature can be disabled by setting this parameter to 0.

8.11 Shared Address Space based MPI Collectives Runtime Parameters

MVAPICH2-X features support for the high-performance shared address-space based multi-leader collectives targeted for modern multi/many-core processors with Infiniband network architectures. Currently, MPI_Reduce and MPI_Allreduce collective operations are supported with this feature. In this section, we specify the runtime parameter that controls this advanced feature.

8.11.1 MV2_USE_XPMEM_COLL

- Class: Run time
- Default: 1 (Set)

Enables support for shared address-space (XPMEM) based multi-leader reduction collectives. This feature can be disabled by setting this parameter to 0.

8.11.2 MV2_XPMEM_COLL_THRESHOLD

- Class: Run time
- Default: Eager message size

Threshold in bytes after which XPMEM based reduction collectives are used. The default value is based on the rendezvous switch-over message size on a given architecture.

8.12 Runtime Parameters for Asynchronous Communication Progress

MVAPICH2-X features support of Asynchronous Communication Progress. In this section, we specify some of the runtime parameters that control these advanced features.

8.12.1 MV2_ASYNC_PROGRESS

- Class: Run time
- Default: 0 (Unset)

Enables use of the basic asynchronous communication progress scheme.

8.12.2 MV2_OPTIMIZED_ASYNC_PROGRESS

- Class: Run time
- Default: 0 (Unset)

Enables use of optimized asynchronous communication progress scheme.

8.13 Runtime Parameters for Collectives with Hardware based SHArP support

MVAPICH2-X supports SHArP-based collectives for MPI applications running over OFA-IB-CH3 interface. In this section, we specify some of the runtime parameters that control these advanced features.

8.13.1 MV2_ENABLE_SHARP

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

Set this to 1, to enable hardware SHArP support in collective communication

8.13.2 MV2_SHARP_HCA_NAME

- Class: Run time
- Default: unset
- Applicable interface(s): OFA-IB-CH3

By default, this is set by the MVAPICH2 library. However, you can explicitly set the HCA name which is realized by the SHArP library.

8.13.3 MV2_SHARP_PORT

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3

By default, this is set by the MVAPICH2 library. However, you can explicitly set the HCA port which is realized by the SHArP library.

9 FAQ and Troubleshooting with MVAPICH2-X

Based on our experience and feedback we have received from our users, here we include some of the problems a user may experience and the steps to resolve them. If you are experiencing any other problem, please feel free to contact us by sending an email to mvapich-help@cse.ohio-state.edu.

9.1 General Questions and Troubleshooting

9.1.1 Compilation Errors with `upcc`

Current version of `upcc` available with MVAPICH2-X package gives a compilation error if the gcc version is not 4.4.7. Please install gcc version 4.4.7 to fix this.

9.1.2 Unresponsive `upcc`

By default, `upcc` compiler driver will transparently use Berkeley's HTTP-based public UPC-to-C translator during compilation. If your system is behind a firewall or not connected to Internet, `upcc` can become unresponsive. This can be solved by using a local installation of UPC translator, which can be downloaded from here.

The translator can be compiled and installed using the following commands:

```
$ make
$ make install PREFIX=<translator-install-path>
```

After this, `upcc` can be instructed to use this translator.

```
$ upcc -translator=<translator-install-path> hello.upc -o hello
```

9.1.3 Shared memory limit for OpenSHMEM / MPI+OpenSHMEM programs

By default, the symmetric heap in OpenSHMEM is allocated in shared memory. The maximum amount of shared memory in a node is limited by the memory available in `/dev/shm`. Usually the default system configuration for `/dev/shm` is 50% of main memory. Thus, programs which specify heap size larger than the total available memory in `/dev/shm` will give an error. For example, if the shared memory limit is 8 GB, the combined symmetric heap size of all intra-node processes shall not exceed 8 GB.

Users can change the available shared memory by remounting `/dev/shm` with the desired limit. Alternatively, users can control the heap size using `OOSHM_SYMMETRIC_HEAP_SIZE` (Section 8.7.2) or disable shared memory by setting `OOSHM_USE_SHARED_MEM=0` (Section 8.7.1). Please be aware that setting a very large shared memory limit or disabling shared memory will have a performance impact.

9.1.4 Collective scratch size in UPC++

While running UPC++ collectives which require high memory footprints the users may encounter errors stating scratch size limit reached. You can increase the scratch size by exporting following environment variable.

`export GASNET_COLL_SCRATCH_SIZE=256M` (or higher).

9.1.5 Install MVAPICH2-X to a specific location

MVAPICH2-X RPMs are relocatable. Please use the `--prefix` option during RPM installation for installing MVAPICH2-X into a specific location. An example is shown below:

```
$ rpm -Uvh --prefix <specific-location>
mvapich2-x_gnu-2.2-0.3.rc1.el7.centos.x86_64.rpm
openshmem-osu_gnu-2.2-0.2.rc1.el7.centos.x86_64.rpm
berkeley_upc-osu_gnu-2.2-0.2.rc1.el7.centos.x86_64.rpm
berkeley_upcxx-osu_gnu-2.2-0.1.rc1.el7.centos.x86_64.rpm
osu-micro-benchmarks_gnu-5.3-1.el7.centos.x86_64.rpm
```

9.1.6 XPMEM based Collectives Performance Issue

Note: XPMEM has a known issue where a memory region that is attached using XPMEM, can not be registered with InfiniBand device. In some cases, if severe performance degradation is observed when using XPMEM based collectives, additionally setting following two parameters can circumvent the performance degradation.

`MV2_IBA_EAGER_THRESHOLD=262144 MV2_VBUF_TOTAL_SIZE=262144`