## Contents

1 Overview of the MVAPICH2-X Project 1

2 Features 3

3 Download and Installation Instructions 6
   3.1 Example downloading RHEL6 package  6
   3.2 Example installing RHEL6 package  6
   3.2.1 Installing with local Berkeley UPC translator support  7
   3.2.2 Installing with GNU UPC translator support  7
   3.2.3 Installing CAF with OpenUH Compiler  7

4 Basic Usage Instructions 9
   4.1 Compile Applications  9
   4.1.1 Compile using `mpicc` for MPI or MPI+OpenMP Applications  9
   4.1.2 Compile using `oshcc` for OpenSHMEM or MPI+OpenSHMEM applications  9
   4.1.3 Compile using `upcc` for UPC or MPI+UPC applications  9
   4.1.4 Compile using `uhcaf` for CAF and MPI+CAF applications  10
   4.2 Run Applications  10
   4.2.1 Run using `mpirun_rsh`  10
   4.2.2 Run using `oshrun`  12
   4.2.3 Run using `upcrun`  12
   4.2.4 Run using `cafrun`  12
   4.2.5 Run using Hydra (mpiexec)  12

5 Hybrid (MPI+PGAS) Applications 13
   5.1 MPI+OpenSHMEM Example  13
   5.2 MPI+UPC Example  14

6 OSU PGAS Benchmarks 17
   6.1 OSU OpenSHMEM Benchmarks  17
   6.2 OSU UPC Benchmarks  18

7 Runtime Parameters 20
   7.1 UPC Runtime Parameters  20
   7.1.1 `UPC_SHARED_HEAP_SIZE`  20
   7.2 OpenSHMEM Runtime Parameters  20
   7.2.1 `OOSHMEM_USE_SHARED_MEM`  20
   7.2.2 `OOSHMEM_SYMMETRIC_HEAP_SIZE`  20
   7.2.3 `OOSHMEM_USE_CMA`  20

8 FAQ and Troubleshooting with MVAPICH2-X 21
   8.1 General Questions and Troubleshooting  21
   8.1.1 Compilation Errors with `upcc`  21
   8.1.2 Unresponsive `upcc`  21
   8.1.3 Shared memory limit for OpenSHMEM / MPI+OpenSHMEM programs  21
8.1.4 Install MVAPICH2-X to a specific location . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
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 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.

MVAPICH2-X supports Unified Parallel C (UPC) OpenSHMEM, and CAF as PGAS models. It can be used to run pure MPI, MPI+OpenMP, pure PGAS (UPC/OpenSHMEM/CAF) 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 data transfer and OpenSHMEM atomic operations. It also provides a high-performance shared memory channel for multi-core InfiniBand clusters.

![MVAPICH2-X Architecture](http://mvapich.cse.ohio-state.edu/)

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 2.20.0. OpenSHMEM implementation is OpenSHMEM 1.0 standard compliant and is based on OpenSHMEM Reference Implementation 1.0h. 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 2.1. 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, as well as hybrid MPI(+OpenMP) + PGAS(UPC/OpenSHMEM/CAF) programs. Current version of MVAPICH2-X 2.1 supports UPC, OpenSHMEM and CAF as the PGAS model. High-level features of MVAPICH2-X 2.1 are listed below.

MPI Features

- Support for MPI-3 features
- (NEW) Based on MVAPICH2 2.1rc2 (OFA-IB-CH3 interface). MPI programs can take advantage of all the features enabled by default in OFA-IB-CH3 interface of MVAPICH2 2.1rc2
  - 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

Unified Parallel C (UPC) Features

- UPC Language Specification 1.2 standard compliance
- (NEW) Based on Berkeley UPC v2.20.0 (contains changes/additions in preparation for UPC 1.3 specification)
- 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_allroadcast, upc_all_scatter, upc_all_gather, upc_all_gather_all, and upc_all_exchange)

OpenSHMEM Features

- OpenSHMEM 1.0 standard compliance
• (NEW) Based on OpenSHMEM reference implementation v1.0h
• (NEW) Support for on-demand establishment of connections
• (NEW) Improved job start up and memory footprint
• Optimized RDMA-based implementation of OpenSHMEM data movement routines
• Support for OpenSHMEM ‘shmem_ptr’ functionality
• 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

(NEW)CAF Features

• (NEW) Based on University of Houston CAF implementation
• (NEW) Efficient point-point read/write operations
• (NEW) Efficient CO_REDUCE and CO_BROADCAST collective operations

Hybrid Program Features

• Supports hybrid programming using MPI(+OpenMP), MPI(+OpenMP)+UPC, MPI(+OpenMP)+OpenSHMEM and MPI(+OpenMP)+CAF
• Compliance to MPI-3, UPC 1.2, OpenSHMEM 1.0 and CAF Fortran 2015 standards
• Optimized network resource utilization through the unified communication runtime
• Efficient deadlock-free progress of MPI and UPC/OpenSHMEM/CAF calls

Unified Runtime Features

• Based on MVAPICH2 2.1 (OFA-IB-CH3 interface). MPI, UPC, OpenSHMEM, CAF 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

http://mvapich.cse.ohio-state.edu/
∗ 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

– 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
∗ Portable Hardware Locality (hwloc v1.9) support for defining CPU affinity
∗ Efficient CPU binding policies (bunch and scatter patterns, socket and numa node 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
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. As an initial technology preview, we are providing RHEL6 and RHEL5 RPMs. We provide RPMs compatible with either OFED-1.5.4 and Mellanox OFED 1.5.3, or OFED 3.5, Mellanox OFED 2.0, and the stock RHEL distro InfiniBand packages.

3.1 Example downloading RHEL6 package

Below are the steps to download MVAPICH2-X RPMs for RHEL6:

```bash
$ wget http://mvapich.cse.ohio-state.edu/download/mvapich2x/mvapich2-x-2.1rc2-1.rhel6.tar.gz
$ tar xzf mvapich2-x-2.1rc2-1.rhel6.tar.gz
$ cd mvapich2-x-2.1rc2-1.rhel6
$ ls
install.sh
mvapich2-x_gnu-2.1-0.3.rc2.el6.x86_64.rpm
mvapich2-x_intel-2.1-0.3.rc2.el6.x86_64.rpm
openshmem-osu_gnu-2.1-0.2.rc1.el6.x86_64.rpm
openshmem-osu_intel-2.1-0.2.rc1.el6.x86_64.rpm
berkeley_upc-osu_gnu-2.1-0.2.rc1.el6.x86_64.rpm
berkeley_upc-osu_gnu-gupc-2.1-0.2.rc1.el6.x86_64.rpm
berkeley_upc-osu_intel-2.1-0.2.rc1.el6.x86_64.rpm
libcaf-osu_gnu-2.1-0.1.rc2.el6.x86_64.rpm
osu-micro-benchmarks_gnu-4.4.1-2.el6.x86_64.rpm
osu-micro-benchmarks_intel-4.4.1-2.el6.x86_64.rpm
README
```

3.2 Example installing RHEL6 package

Running the `install.sh` script will install the software in /opt/mvapich2-x. The /opt/mvapich2-x/gnu directory contains the software built using gcc distributed with RHEL6. The /opt/mvapich2-x/intel directory contains the software built using Intel 13 compilers. The install.sh script runs the following command:

```bash
rpm -Uvh *.rpm --force --nodeps
```

This will upgrade any prior versions of MVAPICH2-X that may be present as well as ignore any dependency issues that may be present with the Intel RPMs (some dependencies are available after sourcing the env scripts provided by the intel compiler).

These RPMs are relocatable and advanced users may skip the install.sh script to directly use alternate commands to install the desired RPMs.

GCC RPMs:

http://mvapich.cse.ohio-state.edu/
3.2.1 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, \texttt{upcc} will not work. In this situation, a local translator should be installed. The local Berkeley UPC-to-C translator can be downloaded from \url{http://upc.lbl.gov/download/}. After installing it, you should edit the \texttt{upcc} configure file (\texttt{/opt/mvapich2-x/gnu/etc/upcc.conf} or \texttt{$HOME/.upccrc})), and set the \texttt{translator} option to be the path of the local translator (e.g. \texttt{/usr/local/berkeley_upc/translator-<VERSION>/targ}).

3.2.2 Installing with GNU UPC translator support

GNU UPC (GUPC) translator support is included in MVAPICH2-X GNU UPC RPM (\texttt{berkeley_upc-osu-gnu-gupc-2.1-0.2.rc1.el6.x86_64.rpm}) as an optional install item. If it is installed, applications can be compiled directly using \texttt{upcc}, and \texttt{gupc} will be invoked internally. To enable GUPC support, GUPC should be installed separately as a prerequisite, which can be downloaded from \url{http://www.gccupc.org/download}. By default, MVAPICH2-X expects GUPC to be installed in \texttt{/usr/local/gupc}.

3.2.3 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 or CAF) applications and hybrid (MPI+ OpenSHMEM, MPI+UPC or MPI+CAF) applications. User should choose the corresponding compilers according to the applications. These compilers (oshcc, upcc, uhcaf 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 RHEL6 GCC RPMs for example, the C compiler should be GCC 4.4.7. 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/openh tests/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

### 4.2 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](http://mvapich.cse.ohio-state.edu/) 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 delimitied 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 used 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.

```bash
$ 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.

```bash
$ 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:

```bash
$ 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.

```bash
$ 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.

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

4.2.5 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:

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

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


http://mvapich.cse.ohio-state.edu/
5 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.

5.1 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.

```c
#include <stdio.h>
#include <shmem.h>
#include <mpi.h>

static int sum = 0;
int main(int c, char *argv[])
{
    int rank, size;
    
    /* SHMEM init */
    start_pes(0);
    
    /* get rank and size */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    
    /* SHMEM barrier */
    shmem_barrier_all();
    
    /* fetch-and-add at root */
    shmem_int_fadd(&sum, rank, 0);
    
    /* MPI barrier */
    MPI_Barrier(MPI_COMM_WORLD);
    
    /* root broadcasts sum */
    MPI_Bcast(&sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
    
    /* print sum */
    fprintf(stderr, "(%d): Sum: %d\n", rank, sum);
    
    shmem_barrier_all();
}
```

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

```sh
$ 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`

### 5.2 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.

```c
#include <stdio.h>
#include <upc.h>
#include <mpi.h>

shared [1] int A[THREADS];
int main() {
    int sum = 0;
    int rank, size;

    /* get MPI rank and size */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

http://mvapich.cse.ohio-state.edu/
/* UPC barrier */
upc_barrier;

/* initialize global array */
A[MYTHREAD] = rank;
int *local = ((int *)&A[MYTHREAD]);

/* MPI barrier */
MPI_Barrier(MPI_COMM_WORLD);

/* sum up the value with each UPC thread */
MPI_Allreduce(local, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

/* print sum */
if (MYTHREAD == 0)
    fprintf(stderr, "(%d): Sum: %d\n", rank, sum);

upc_barrier;

return 0;
}

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 OSU PGAS Benchmarks

6.1 OSU OpenSHMEM Benchmarks

We have extended the OSU Micro Benchmark (OMB) suite with tests to measure performance of OpenSHMEM operations. OSU Micro Benchmarks (OMB-4.4.1) 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

6.2 OSU UPC Benchmarks

OSU Microbenchmarks extensions include UPC benchmarks also. Current version (OMB-4.4.1) 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_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 Runtime Parameters

MVAPICH2-X supports all the runtime parameters of MVAPICH2 (OFA-IB-CH3). A comprehensive list of all runtime parameters of MVAPICH2 2.1 can be found in User Guide. Runtime parameters specific to MVAPICH2-X are listed below.

7.1 UPC Runtime Parameters

7.1.1 UPC_SHARED_HEAP_SIZE

- Class: Run time
- Default: 64M

Set UPC Shared Heap Size

7.2 OpenSHMEM Runtime Parameters

7.2.1 OOSHM_USE_SHARED_MEM

- Class: Run time
- Default: 1

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

7.2.2 OOSHM_SYMMETRIC_HEAP_SIZE

- Class: Run time
- Default: 512M

Set OpenSHMEM Symmetric Heap Size

7.2.3 OSHM_USE_CMA

- Class: Run time
- Default: 1

Enable/Disable CMA based intra-node communication design
8 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.

8.1 General Questions and Troubleshooting

8.1.1 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.

8.1.2 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

8.1.3 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 7.2.2) or disable shared memory by setting OOSHM_USE_SHARED_MEM=0 (Section 7.2.1). Please be aware that setting a very large shared memory limit or disabling shared memory will have a performance impact.
8.1.4 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:

```bash
$ rpm -Uvh --prefix <specific-location>
mvapich2-x.gnu-2.1-0.2.rc1.el6.x86_64.rpm
openshmem-osu.gnu-2.1-0.2.rc1.el6.x86_64.rpm
berkeley_upc-osu.gnu-2.1-0.2.rc1.el6.x86_64.rpm
osu-micro-benchmarks.gnu-4.4-1.el6.x86_64.rpm
```