SOFTWARE MANUAL

Mellanox HPC-X™ Software Toolkit User Manual

Rev 1.8

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT ("PRODUCT(S)") AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES "AS IS" WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCT(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Table of Contents

List of Tables

List of Figures

Document Revision History

Revision	Date	Description

1.8	April 18, 2017	• Added the following sections:
		  • Section 6, “Unified Communication - X Framework Library”, on page 53
		  • Section 6.2, “Configuring UCX”, on page 53
		  • Section 6.3, “Tuning UCX Settings”, on page 54
		  • Section 6.4.1, “ucx_perftest”, on page 54
		• Updated the following sections:
		  • Section 1.1, “HPC-X Package Contents”, on page 11
		  • Updated the FCA version to 3.7
		  • Added the “MXM_IB_USE_GRH” parameter to table Section 7, “MXM Environment Parameters”, on page 36

1.7.406	December 15, 2016	• Added the following section: • Section 4.4.7.1, “HCOLL v3.7 Limitations”, on page 26
1.7	October 05, 2016	• Added the following sections: • Section 2.3, “Building HPC-X with the Intel Compiler Suite”, on page 13
	September 2016	• Added the following sections: • Section 4.4.5, “Enabling Offloaded MPI Non-blocking Collectives”, on page 24

1.6	June 30, 2016	• Added the following sections:
		  • Section 3.10, “IB-Router”, on page 18
		  • Section 4.4.7, “Enabling HCOLL Topology Awareness”, on page 25
		  • Section 4.4.8, “Enabling SHARP Accelerated Collectives”, on page 26
		• Updated the following sections:
		  • Section 1.1, “HPC-X Package Contents”, on page 11
		  • Updated the FCA version to 3.5
		  • Updated the MXM version to 3.5
		  • Section 3.4, “Rebuilding Open MPI from HPC-X™ Sources”, on page 16
		  • Updated the MXM version to 3.5

Revision	Date	Description

1.5	January 31st 2016	• Added the following sections:
		  • Section 5.7, “Running MXM with RoCE”, on page 32
		  • Section 5.7.1, “Running MXM with RoCE v1”, on page 32
		  • Section 5.7.2.1, “Using RoCE v2 in Open MPI”, on page 33
		• Updated the following sections:
		  • Section 1.1, “HPC-X Package Contents”, on page 11
		  • Updated the Open MPI version to 1.10
		  • Updated the MXM version to 3.4
		  • Section 5.13, “MXM Environment Parameters”, on page 36
		  • Added the following parameter: `MXM_IB_GID_INDEX`
		  • Section 5.8, “Configuring MXM over Different Transports”, on page 33
		  • Updated the MXM version to 3.4
		• Removed section Running MPI with MXM

1.4	September 2015	• Updated the following sections:
		  • Section 4.1, “Overview”, on page 20
		  • Section 4.4.2, “Enabling FCA in Open MPI”, on page 24
		  • Section 5.8, “Configuring MXM over Different Transports”, on page 33

1.3	June 2015	• Added the following section:
		  • Section 4.4.6, “Enabling Multicast Accelerated Collectives”, on page 25
		  • Section 4.4.6.1, “Configuring IPoIB”, on page 25
		• Updated the following sections:
		  • Section 3.9, “Running MPI with FCA v3.x (hcoll)”, on page 18
		  • Section 4.4.2, “Enabling FCA in Open MPI”, on page 24
		  • Section 5.3, “Running Open MPI with pml “yalla””, on page 29
		  • Section 5.11, “Support for a Non-Base LID”, on page 34
		  • Section 7.2.1.1, “Enabling MXM for HPC-X OpenSHMEM Jobs”, on page 56
		  • Section 8.3, “Various Executable Examples”, on page 64
		• Removed the following sections:
		  • Runtime Configuration of FCA and its subsections
		  • FCA 3.1 Integration
		  • Configuring Hugepages

1.2	August 2014	Initial release

About This Manual

Scope

This document describes Mellanox HPC-X™ Software Toolkit acceleration packages. It includes information on installation, configuration and rebuilding of HPC-X packages.

Intended Audience

This manual is intended for system administrators responsible for the installation, configuration, management and maintenance of the software and hardware.

It is also for users who would like to use the latest Mellanox software accelerators to achieve the best possible application performance.

Syntax Conventions

Table 1 - Syntax Conventions

Prompt	Shell
machine-name%	C shell on UNIX, Linux, or AIX
machine-name#	C shell superuser on UNIX, Linux, or AIX
$	Bourne shell and Korn shell on UNIX, Linux, or AIX
#	Bourne shell and Korn shell superuser on UNIX, Linux, or AIX
C:\>	Windows command line

1 HPC-X™ Software Toolkit Overview

Mellanox HPC-X™ is a comprehensive software package that includes MPI, SHMEM and UPC communications libraries. HPC-X also includes various acceleration packages to improve both the performance and scalability of applications running on top of these libraries, including MXM (Mellanox Messaging) which accelerates the underlying send/receive (or put/get) messages, and FCA (Fabric Collectives Accelerations) which accelerates the underlying collective operations used by the MPI/PGAS languages. This full-featured, tested and packaged version of HPC software enables MPI, SHMEM and PGAS programming languages to scale to extremely large clusters by improving on memory and latency related efficiencies, and to assure that the communication libraries are fully optimized of the Mellanox interconnect solutions.

Mellanox HPC-X™ allow OEM's and System Integrators to meet the needs of their end-users by deploying the latest available software that takes advantage of the features and capabilities available in the most recent hardware and firmware changes.

1.1 HPC-X Package Contents

Mellanox FCA 3.x (HCOLL) is now the default FCA used in HPC-X as of version 1.8 and it replaces FCA v2.x.HPC-X package contains the following pre-compiled HPC packages:

Table 2 - HPC-X Package Contents

Components	Description
MPI	• Open MPI and OpenSHMEM v2.x (MPI-3 complaint, OpenSHMEM v1.0 compliant). Open MPI and OpenSHMEM are available at: http://www.open-mpi.org/software/ompi/v1.10/ • MPI profiler (IPM - open source tool from http://ipm-hpc.org/) • MPI tests (OSU, IMB, random ring, etc.)
HPC Acceleration Package	• MXM 3.6 (default) • FCA v3.7 (code name: "hcoll" - default)[a] • UCX • knem (High-Performance Intra-Node MPI Communication module from: http://runtime.bordeaux.inria.fr/knem/)
Extra packages	• Berkeley UPC v2.22.0 • libibprof (IB verbs profiler)

a. As of HPC-X v1.8, FCA 3.x (HCOLL) is the default FCA used in HPC-X and it replaces FCA v2.x.

1.2 HPC-X™ Requirements

The platform and requirements for HPC-X are detailed in the following table:

Table 3 - HPC-X™ Requirements

Platform	Drivers and HCAs
OFED / MLNX_OFED	• OFED 1.5.3 and later • MLNX_OFED 1.5.3-x.x.x, 2.0-x.x.x and later
HCAs	• ConnectX®-5 / ConnectX®-5 Ex **Note:** Using ConnectX®-5 adapter cards requires MLNX_OFED v4.0-1.0.0.0 and above. • ConnectX®-4 / ConnectX®-4 Lx • ConnectX®-3 / ConnectX®-3 Pro • ConnectX®-2 • Connect-IB®

2 Installing and Loading HPC-X™

2.1 Installing HPC-X

➢ ***To install HPC-X:***

Step 1. Extract hpcx.tar into current working directory.

```
$ tar zxvf hpcx.tar
```

Step 2. Update shell variable of the location of hpc-x installation.

```
$ cd hpcx
$ export HPCX_HOME=$PWD
```

2.2 Loading HPC-X Environment from bash

HPC-X includes Open MPI v2.x. Each Open MPI version has its own module file which can be used to load desired version.

The symbolic links `hpcx-init.sh` and `modulefiles/hpcx` point to the default version (Open MPI v1.10).

➢ ***To load Open MPI/OpenSHMEM v2.x based package:***

```
$ source $HPCX_HOME/hpcx-init.sh
$ hpcx_load
$ env | grep HPCX
$ mpirun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_usempi
$ oshrun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_oshmem
$ hpcx_unload
```

2.3 Building HPC-X with the Intel Compiler Suite

As of version 1.7, Mellanox no longer distributes HPC-X builds based on the Intel compiler suite. However, after following the HPC-X deployment example below, HPC-X can subsequently be rebuilt from source with your Intel compiler suite as follows:

```
$ tar xfp ${HPCX_HOME}/sources/openmpi-gitclone.tar.gz
$ cd ${HPCX_HOME}/sources/openmpi-gitclone
$ ./configure CC=icc CXX=icpc F77=ifort FC=ifort --prefix=${HPCX_HOME}/ompi-icc
            --with-knem=${HPCX_HOME}/knem \
            --with-fca=${HPCX_HOME}/fca --with-mxm=${HPCX_HOME}/mxm \
            --with-hcoll=${HPCX_HOME}/hcoll \
            --with-platform=contrib/platform/mellanox/optimized \
              2>&1 | tee config-icc-output.log
$ make -j32 all 2>&1 | tee build_icc.log && make -j24 install 2>&1 | tee install_icc.log
```

In the example above, 4 switches are used to specify the compiler suite:

- **CC:** Specifies the C compiler
- **CXX:** Specifies the C++ compiler
- **F77:** Specifies the Fortran 77 compiler

- **FC:** Specifies the Fortran 90 compiler

We strongly recommend using a single compiler suite whenever possible. Unexpected or undefined behavior can occur when you mix compiler suites in unsupported ways (e.g., mixing Fortran 77 and Fortran 90 compilers between different compiler suites is almost guaranteed not to work.)

In all cases, the Intel compiler suite must be found in your PATH and be able to successfully compile and link non-MPI applications before Open MPI will be able to be built properly.

2.4 Loading HPC-X Environment from Modules

➢ ***To load Open MPI/OpenSHMEM v1.10 based package:***

```
$ module use $HPCX_HOME/modulefiles
$ module load hpcx
$ mpirun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_c
$ oshrun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_oshmem
$ module unload hpcx
```

The instrumentation can be applied on a per-collective basis, and is controlled by the following environment variables:

```
$ export IPM_ADD_BARRIER_TO_REDUCE=1
$ export IPM_ADD_BARRIER_TO_ALLREDUCE=1
$ export IPM_ADD_BARRIER_TO_GATHER=1
$ export IPM_ADD_BARRIER_TO_ALL_GATHER=1
$ export IPM_ADD_BARRIER_TO_ALLTOALL=1
$ export IPM_ADD_BARRIER_TO_ALLTOALLV=1
$ export IPM_ADD_BARRIER_TO_BROADCAST=1
$ export IPM_ADD_BARRIER_TO_SCATTER=1
$ export IPM_ADD_BARRIER_TO_SCATTERV=1
$ export IPM_ADD_BARRIER_TO_GATHERV=1
$ export IPM_ADD_BARRIER_TO_ALLGATHERV=1
$ export IPM_ADD_BARRIER_TO_REDUCE_SCATTER=1
```

By default, all values are set to '0'.

3 Running, Configuring and Rebuilding HPC-X™

The sources for BUPC, SHMEM and OMPI can be found at `$HPCX_HOME/sources/`.

Please refer to `$HPCX_HOME/sources/` for more information on building details.

3.1 Starting FCA Manager from HPC-X

Prior to using FCA, the following command should be executed as root once on all cluster nodes in order to mimic post-install procedure.

```
# $HPCX_HOME/fca/scripts/udev-update.sh
```

The FCA manager should be run on only one machine, and not on one of the compute nodes.

```
$ $HPCX_HOME/fca/scripts/fca_managerd start
```

FCA 2.5 is the default FCA version embedded in the HPC-X package. To install the FCA Manager please refer to FCA 2.5 User Manual section Installing the FCA Manager on a Dedicated Node

3.2 Profiling IB verbs API

- ***To profile IB verbs API:***

```
$ export IBPROF_DUMP_FILE=ibprof_%J_%H_%T.txt
$ export LD_PRELOAD=$HPCX_IBPROF_DIR/lib/libibprof.so:$HPCX_MXM_DIR/lib/lib-
mxm.so:$HPCX_HCOLL_DIR/lib/libhcoll.so
$ mpirun -x LD_PRELOAD <...>
```

For further details on profiling IB verbs API, please refer to libibprof README file.

README file location is:

`$HPCX_HOME/libibprof/README`

3.3 Profiling MPI API

- ***To profile MPI API:***

```
$ export IPM_KEYFILE=$HPCX_IPM_DIR/etc/ipm_key_mpi
$ export IPM_LOG=FULL
$ export LD_PRELOAD=$HPCX_IPM_DIR/lib/libipm.so
$ mpirun -x LD_PRELOAD <...>
$ $HPCX_IPM_DIR/bin/ipm_parse -html outfile.xml
```

For further details on profiling MPI API, please refer to:

http://ipm-hpc.org/

The Mellanox-supplied version of IMP contains an additional feature (Barrier before Collective), not found in the standard package, that allows end users to easily determine the extent of application imbalance in applications which use collectives. This feature instruments each collective so that it calls `MPI_Barrier()` before calling the collective operation itself. Time spent in this `MPI_Barrier()` is not counted as communication time, so by running an application with and without the Barrier before Collective feature, the extent to which application imbalance is a factor in performance, can be assessed.

The instrumentation can be applied on a per-collective basis, and is controlled by the following environment variables:.

```
$ export IPM_ADD_BARRIER_TO_REDUCE=1
$ export IPM_ADD_BARRIER_TO_ALLREDUCE=1
$ export IPM_ADD_BARRIER_TO_GATHER=1
$ export IPM_ADD_BARRIER_TO_ALL_GATHER=1
$ export IPM_ADD_BARRIER_TO_ALLTOALL=1
$ export IPM_ADD_BARRIER_TO_ALLTOALLV=1
$ export IPM_ADD_BARRIER_TO_BROADCAST=1
$ export IPM_ADD_BARRIER_TO_SCATTER=1
$ export IPM_ADD_BARRIER_TO_SCATTERV=1
$ export IPM_ADD_BARRIER_TO_GATHERV=1
$ export IPM_ADD_BARRIER_TO_ALLGATHERV=1
$ export IPM_ADD_BARRIER_TO_REDUCE_SCATTER=1
```

By default, all values are set to '0'.

3.4 Rebuilding Open MPI from HPC-X™ Sources

HPC-X package contains Open MPI sources which can be found at `$HPCX_HOME/sources/` folder.

- ***To build Open MPI from sources:***

```
$ HPCX_HOME=/path/to/extracted/hpcx
$ ./configure --prefix=${HPCX_HOME}/hpcx-ompi --with-knem=${HPCX_HOME}/knem \
            --with-fca=${HPCX_HOME}/fca --with-mxm=${HPCX_HOME}/mxm \
            --with-hcoll=${HPCX_HOME}/hcoll \
            --with-platform=contrib/platform/mellanox/optimized \
            --with-slurm --with-pmi
$ make -j9 all && make -j9 install
```

Open MPI and OpenSHMEM are pre-compiled with MXM v3.6 and use it by default.

3.5 Generating MXM Statistics for Open MPI/OpenSHMEM

In order to generate statistics, the statistics destination and trigger should be set.

- Destination is set by MXM_STATS_DEST environment variable whose values can be one of the following:

Table 4 - MXM_STATS_DEST Environment Variables

Value	Description
`empty string`	statistics are not reported
`stdout`	Print to standard output
`stderr`	Print to standard error
`file:<filename>`	Write to a file. Following substitutions are made: %h: host, %p: pid, %c: cpu, %t: time, %e: exe
`file:<filename>:bin`	Same as previous, but a binary format is used when saving. The file will be smaller, but not human-readable. The mxm_stats_parser tool can be used to parse binary statistics files

Examples:

```
$ export MXM_STATS_DEST="file:mxm_%h_%e_%p.stats"
$ export MXM_STATS_DEST="file:mxm_%h_%c.stats:bin"
$ export MXM_STATS_DEST="stdout"
```

- Trigger is set by MXM_STATS_TRIGGER environment variables. It can be one of the following:

Table 5 - MXM_STATS_TRIGGER Environment Variables

Environment Variable	Description
`exit`	Dump statistics just before exiting the program
`timer:<interval>`	Dump statistics periodically, interval is given in seconds

Example:

```
$ export MXM_STATS_TRIGGER=exit
$ export MXM_STATS_TRIGGER=timer:3.5
```

The statistics feature is only enabled for the 'debug' version of MXM which is included in HPC-X. To use the statistics, run the command below from the command line:

```
$ mpirun -x LD_PRELOAD=$HPCX_DIR/mxm/debug/lib/libmxm.so ...
```

3.6 Loading KNEM Module

MXM's intra-node communication uses the KNEM module which improves the performance significantly.

➢ ***To use the KNEM module:***

- Load the KNEM module.

  Please run the following commands on all cluster nodes to enable KNEM intra-node device.

```
# insmod $HPCX_HOME/knem/lib/modules/$(uname -r)/knem.ko
# chmod 666 /dev/knem
```

On RHEL systems, to enable the KNEM module on machine boot, add these commands into the `/etc/rc.modules` script.

Making `/dev/knem` public accessible posses no security threat, as only the memory buffer that was explicitly made readable and/or writable can be accessed read and/or write through the 64bit cookie. Morover, recent KNEM releases enforce by default that the attacker and the target process have the same UID which prevent any security issues.

3.7 Running MPI with FCA v2.5

Make sure FCA manager is running in the fabric.

```
$ mpirun -mca coll_fca_enable 1 <...>
```

For further details on starting FCA manager, please refer to 3.1 “Starting FCA Manager from HPC-X,” on page 15

3.8 Running OpenSHMEM with FCA v2.5

Make sure FCA manager is running in the fabric.

```
$ oshrun -mca scoll_fca_enable 1 <...>
```

For further details on starting FCA manager, please refer to 3.1 “Starting FCA Manager from HPC-X,” on page 15

3.9 Running MPI with FCA v3.x (hcoll)

FCA v3.x is enabled by default in HPC-X.

- Running with default FCA configuration parameters:

```
$ mpirun -mca coll_hcoll_enable 1 -x HCOLL_MAIN_IB=mlx4_0:1 <...>
```

- Running OSHMEM with FCA v3.x:

```
% oshrun -mca scoll_mpi_enable 1 -mca scoll basic,mpi -mca coll_hcoll_enable 1  <...>
```

3.10 IB-Router

As of v1.6, HPC-X supports ib-router to allow hosts that are located on different IB subnets to communicate with each other. This support is currently available when using the `'openib btl'` in Open MPI.

To use `ib-router`, make sure MLNX_OFED v3.3-1.0.0.0 and above is installed and then recompile your Open MPI with `'--enable-openib-rdmacm-ibaddr'` (for further information of now to compile Open MPI, refer to Section 3.4, “Rebuilding Open MPI from HPC-X™ Sources”, on page 16)

➢ ***To enable routing over IB, please follow these steps:***

1. Configure Open MPI with `--enable-openib-rdmacm-ibaddr`.
2. Use `rdmacm` with `openib btl` from the command line.
3. Set the `btl_openib_allow_different_subnets` parameter to 1.
   It is 0 by default.
4. Set the `btl_openib_gid_index` parameter to 1.

For example - to run the IMB benchmark on host1 and host2 which are on separate subents, i.e. have different `subnet_prefix`, use the following command line:

```
shell$ mpirun -np 2 --display-map --map-by node -H host1,host2 -mca pml ob1 -mca btl
self,sm,openib --mca btl_openib_cpc_include rdmacm -mca btl_openib_if_include mlx5_0:1 -mca
btl_openib_gid_index 1 -mca btl_openib_allow_different_subnets 1  ./IMB/src/IMB-MPI1 pingpong
```

More information about how to enable and use ib-router is here - https://www.open-mpi.org/faq/?category=openfabrics#ib-router

When using “`openib btl`”, RoCE and IB router are mutually exclusive. The Open MPI inside HPC-X 1.6 is not compiled with ib-router support, therefore it supports RoCE out-of-the-box.

4 Mellanox Fabric Collective Accelerator (FCA)

4.1 Overview

To meet the needs of scientific research and engineering simulations, supercomputers are growing at an unrelenting rate. As supercomputers increase in size from mere thousands to hundreds-of-thousands of processor cores, new performance and scalability challenges have emerged. In the past, performance tuning of parallel applications could be accomplished fairly easily by separately optimizing their algorithms, communication, and computational aspects. However, as systems continue to scale to larger machines, these issues become co-mingled and must be addressed comprehensively.

Collective communications execute global communication operations to couple all processes/nodes in the system and therefore must be executed as quickly and as efficiently as possible. Indeed, the scalability of most scientific and engineering applications is bound by the scalability and performance of the collective routines employed. Most current implementations of collective operations will suffer from the effects of systems noise at extreme-scale (system noise increases the latency of collective operations by amplifying the effect of small, randomly occurring OS interrupts during collective progression.) Furthermore, collective operations will consume a significant fraction of CPU cycles, cycles that could be better spent doing meaningful computation.

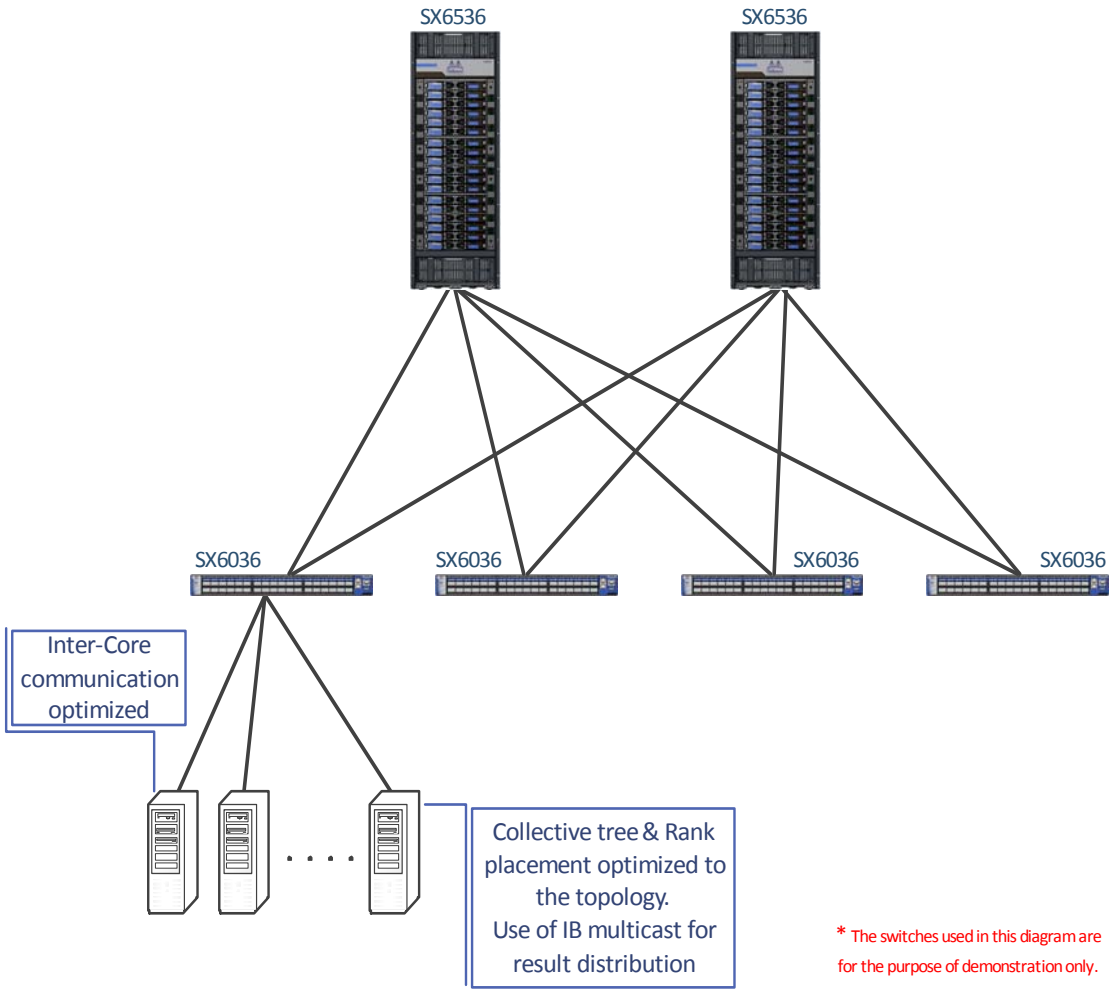
Mellanox Technologies has addressed these two issues, lost CPU cycles and performance lost to the effects of system noise, by offloading the communications to the host channel adapters (HCAs) and switches. The technology, named CORE-Direct® (Collectives Offload Resource Engine), provides the most advanced solution available for handling collective operations thereby ensuring maximal scalability, minimal CPU overhead, and providing the capability to overlap communication operations with computation allowing applications to maximize asynchronous communication.

Additionally, FCA v3.x also contains support for building runtime configurable hierarchical collectives. As with FCA 2.X, FCA v3.x leverages hardware multicast capabilities to accelerate collective operations. In FCA v3.x we also expose the performance and scalability of Mellanox's advanced point-to-point library, MXM, in the form of the "mlnx_p2p" BCOL. This allows users to take full advantage of new hardware features with minimal effort.

FCA v3.x and above is a standalone library that can be integrated into any MPI or PGAS runtime. Support for FCA is currently integrated into Open MPI versions 1.7.4 and higher. FCA v3.x release currently supports blocking and non-blocking variants of "Allgather", "Allreduce", "Barrier", and "Bcast".

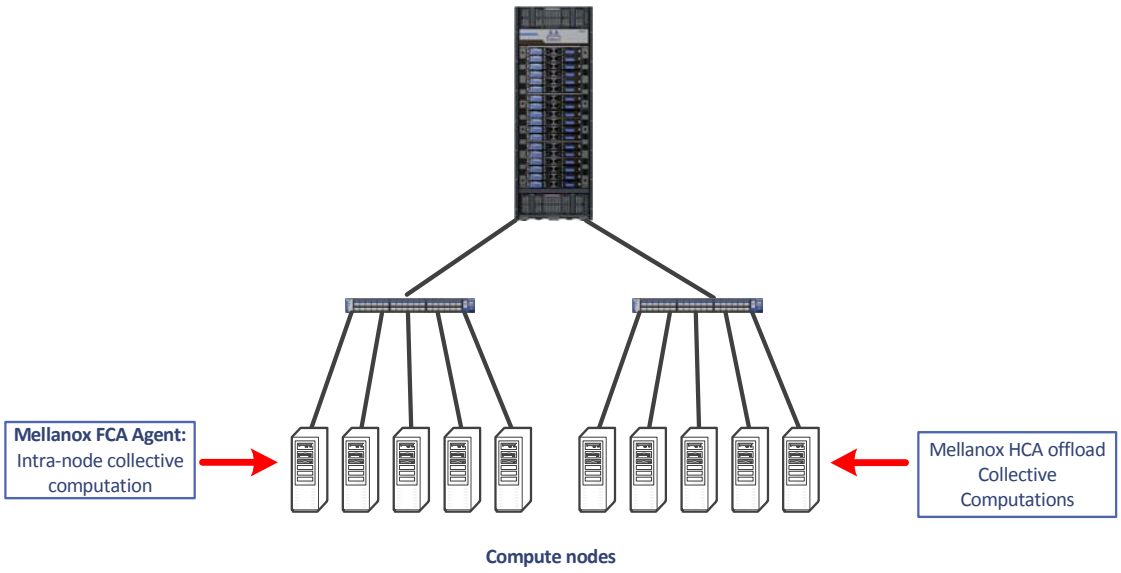
The following diagram summarizes the FCA architecture:

Figure 1: FCA Architecture



The following diagram shows the FCA components and the role that each plays in the acceleration process:

Figure 2: FCA Components



4.2 FCA Installation Package Content

hcoll is part of the HPC-X software toolkit and does not requires special installation.

The FCA installation package includes the following items:

- FCA- Mellanox Fabric Collector Accelerator Installation files
  - hcoll-<version>.x86_64.<OS>.rpm
  - hcoll-<version>.x86_64.<OS>.tar.gz

    where:

    <version>: The version of this release

    <OS>: One of the supported Linux distributions.
- Mellanox Fabric Collective Accelerator (FCA) Software: End-User License Agreement
- FCA MPI runtime libraries
- Mellanox Fabric Collective Accelerator (FCA) Release Notes

4.3 Differences Between FCA v2.5 and FCA v3.x

FCA v3.x is new software which continues to expose the power of CORE-Direct® to offload collective operations to the HCA. It adds additional scalable algorithms for collectives and supports both blocking and non-blocking APIs (MPI-3 SPEC compliant). Additionally, FCA v3.x (hcoll) does not require FCA manager daemon.

4.4 Configuring FCA

4.4.1 Compiling Open MPI with FCA 3.x

➢ ***To compile Open MPI with FCA 3.x***

Step 1. Install FCA 3.x from:

- an RPM.

```
# rpm -ihv hcoll-x.y.z-1.x86_64.rpm
```

- a tarball.

```
% tar jxf hcoll-x.y.z.tbz
```

FCA 3.X will be installed automatically in the /opt/mellanox/hcoll folder.

Step 2. Enter the Open MPI source directory and run the following command:

```
% cd $OMPI_HOME
% ./configure --with-hcoll=/opt/mellanox/hcoll --with-mxm=/opt/mellanox/mxm < ... other
configure  parameters>
% make -j 9 && make install -j 9
```

libhcoll requires MXM v2.1 or higher.

➢ ***To check the version of FCA installed on your host:***

```
% rpm -qi hcoll
```

➢ ***To upgrade to a newer version of FCA 3.X:***

Step 1. Remove the existing FCA 3.X.

```
% rpm -e hcoll
```

Step 2. Remove the precompiled Open MPI.

```
% rpm -e mlnx-openmpi_gcc
```

Step 3. Install the new FCA 3.X and compile the Open MPI with it.

4.4.2 Enabling FCA in Open MPI

To enable FCA 3.X HCOLL collectives in Open MPI, explicitly ask for them by setting the following MCA parameter:

```
%mpirun -np 32  -mca coll_hcoll_enable 1 -x coll_hcoll_np=0 -x HCOLL_-
MAIN_IB=<device_name>:<port_num> -x MXM_RDMA_PORTS=<device_name>:<port_num> -x
HCOLL_ENABLE_MCAST_ALL=1 ./a.out
```

4.4.3 Tuning FCA 3.X Setting

The default FCA 3.X settings should be optimal for most systems. To check the available FCA 3.X parameters and their default values, run the following command:

```
% /opt/mellanox/hcoll/bin/hcoll_info --all
```

FCA 3.X parameters are simply environment variables and can be modified in one of the following ways:

- Modify the default FCA 3.X parameters as part of the `mpirun` command:

```
% mpirun ... -x HCOLL_ML_BUFFER_SIZE=65536
```

- Modify the default FCA 3.X parameter values from SHELL:

```
% export -x HCOLL_ML_BUFFER_SIZE=65536
% mpirun ...
```

4.4.4 Selecting Ports and Devices

➢ ***To select the HCA device and port you would like FCA 3.X to run over:***

```
-x HCOLL_MAIN_IB=<device_name>:<port_num>
```

4.4.5 Enabling Offloaded MPI Non-blocking Collectives

In order to use hardware offloaded collectives in non-blocking MPI calls (e.g. MPI_Ibcast()), set the following parameter

```
-x HCOLL_ENABLE_NBC=1
```

Note that enabling non-blocking MPI collectives will disable multicast acceleration in blocking MPI collectives.

The supported non-blocking MPI collectives are:

- MPI_Ibarrier
- MPI_Ibcast
- MPI_Iallgather
- MPI_allreduce (4b, 8b, SUM, MIN, PROD, AND, OR, LAND, LOR)

4.4.6 Enabling Multicast Accelerated Collectives

FCA 3.x, like its 2.x predecessor, uses hardware multicast to accelerate certain collective operations. In order to take full advantage of this unique capability, you must first have IPoIB configured on every adapter card/port pair that collective message traffic flows through.

4.4.6.1 Configuring IPoIB

To configure IPoIB, you need to define an IP address on the IB interface.

Step 1. Use /usr/bin/ibdev2netdev to show all IB interfaces.

```
hpchead ~ >ibdev2netdev
mlx4_0 port 1 ==> ib0 (Down)
mlx4_0 port 2 ==> ib1 (Down)
mlx5_0 port 1 ==> ib2 (Down)
mlx5_0 port 2 ==> ib3 (Down)
```

Step 2. Use /sbin/ifconfig to get the address informations for a specific interface (e.g. ib0).

```
hpchead ~ >ifconfig ib0
ifconfig uses the ioctl access method to get the full address information, which limits
hardware addresses to 8 bytes. Since InfiniBand address has 20 bytes, only the first 8
bytes are displayed correctly.
Ifconfig is obsolete! For replacement check ip.
ib0       Link encap:InfiniBand  HWaddr
          A0:04:02:20:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          BROADCAST MULTICAST  MTU:2044  Metric:1
          RX packets:58 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1332 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1024
          RX bytes:3248 (3.1 KiB)  TX bytes:80016 (78.1 KiB)
```

Or you can use /sbin/ip for the same purpose

```
hpchead ~ >ip addr show ib0
4: ib0: <BROADCAST,MULTICAST> mtu 2044 qdisc mq state DOWN qlen 1024
    link/infiniband a0:04:02:20:fe:80:00:00:00:00:00:00:00:02:c9:03:00:21:f9:31 brd
    00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global ib0
```

In the example above, the IP is defined (192.168.1.1). If it is not defined, then you can define an IP address now.

4.4.7 Enabling HCOLL Topology Awareness

FCA v3.x (HCOLL) is able to leverage fabric topology information. To allow HCOLL to take advantage of this capability, the steps below need to be executed during fabric bring-up in order to stage the proper information for HCOLL to retrieve.

1. Generate a fabric.cache file.

```
ibnetdiscover --cache fabric.cache
```

In case of several cards connected as independent fabrics, choose the correct card/port to start from:

```
ibnetdiscover -C mlx5_0 -P 1 --cache fabric.cache
```

2. Stage the fabric.cache file on NFS.

   The `fabric.cache` file can be placed on a shared file system in one of these locations:

   - In a user defined location, `/path/to/fabric.cache`. The location should be passed to HCOLL via the command line parameter value

     ```
     -x HCOLL_TOPOLOGY_DATAFILE=/path/to/faric.cache
     ```

   - The standard location where HCOLL expects to find the `file:$HPCX_HCOLL_DIR/etc/fabric.cache`

     a. Node local placement of `fabric.cache`.

     Super user needs to run as root:

     ```
     export cluster=[partition_name]
     ibnetdiscover --cache /tmp/${cluster}_fabrictopo.cache > /tmp/${cluster}_fabric-topo.data
     sudo pdsh -P $cluster ibstat > /tmp/${cluster}_fabrictopo.map
     ```

The `fabric.cache` file must be rebuilt if there were changes in the cluster topology (e.g. connections.)

All `fabric.cache` files across the fabric must be identical. For this purpose each rank calculates the md5sum of its `fabric.cache` file and exchanges this data with all other ranks.

➢ ***To enable this feature:***

```
-x HCOLL_ENABLE_TOPOLOGY=1
```

4.4.7.1 HCOLL v3.7 Limitations

For the restrictions which may be applicable to end-user applications looking to exploit/utilize the fabric collective accelerator, please refer to the HCOLL/S822LC (8335-GTB) SUPPORT README.

4.4.8 Enabling SHARP Accelerated Collectives

As of v1.7, HPC-X supports SHARP Accelerated Collectives. These collectives are enabled by default if FCA 3.5 (HCOLL) and above detects that it is running in a supporting environment.

➢ ***To enable SHARP acceleration:***

```
-x HCOLL_ENABLE_SHARP=1
```

➢ ***To disable SHARP acceleration:***

```
-x HCOLL_ENABLE_SHARP=0
```

➢ ***To change sharp message threshold:***

```
-x HCOLL_BCOL_P2P_ALLREDUCE_SHARP_MAX=<threshold>  ( default:256)
```

The maximum allreduce size runs through SHARP. Messages with a size greater than the above will fallback to non-SHARP based algorithms (multicast based or non-multicast based)

- ***To use SHArP non-blocking interface:***

```
-x HCOLL_ENABLE_SHARP_NONBLOCKING=1
```

For instructions on how to deploy SHARP in Infiniband fabric, see the SHARP Deployment Guide.

Once SHARP is deployed, you need to only specify the HCA device (device_name) and port number (port_num) that is connected to the SHARP tree in the following way:

```
-x HCOLL_MAIN_IB=<device_name>:<port_num>
```

5 MellanoX Messaging Library

5.1 Overview

MellanoX Messaging (MXM) library provides enhancements to parallel communication libraries by fully utilizing the underlying networking infrastructure provided by Mellanox HCA/switch hardware. This includes a variety of enhancements that take advantage of Mellanox networking hardware including:

- Multiple transport support including RC, DC and UD
- Proper management of HCA resources and memory structures
- Efficient memory registration
- One-sided communication semantics
- Connection management
- Receive side tag matching
- Intra-node shared memory communication

These enhancements significantly increase the scalability and performance of message communications in the network, alleviating bottlenecks within the parallel communication libraries.

The latest MXM software can be downloaded from the Mellanox website.

5.2 Compiling Open MPI with MXM

MXM has been integrated into the HPC-X Toolkit package. The steps described below are only required if you have downloaded the mxm.rpm from the Mellanox site

Step 1. Install MXM from:

- an RPM

```
% rpm -ihv mxm-x.y.z-1.x86_64.rpm
```

- a tarball

```
% tar jxf mxm-x.y.z.tar.bz
```

MXM will be installed automatically in the `/opt/mellanox/mxm` folder.

Step 2. Enter Open MPI source directory and run:

```
% cd $OMPI_HOME
% ./configure --with-mxm=/opt/mellanox/mxm <... other configure parameters...>
% make all && make install
```

Older versions of MLNX_OFED come with pre-installed older MXM and Open MPI versions. Uninstall any old MXM version prior to installing the latest MXM version in order to use it with older MLNX_OFED versions.

Table 6 - MLNX_OFED and MXM Versions

MLNX_OFED Version	MXM Version
v1.5.3-3.1.0 and v2.0-3.0.0	MXM v1.x and Open MPI compiled with MXM v1.x
v2.0-3.0.0 and higher	MXM v2.x/3.x and Open MPI compiled with MXM v2.x/ 3.x

To check the version of MXM installed on your host, run:

```
% rpm -qi mxm
```

➢ ***To upgrade MLNX_OFED v1.5.3-3.1.0 or later with a newer MXM:***

Step 1. Remove MXM.

```
# rpm -e mxm
```

Step 2. Remove the pre-compiled Open MPI.

```
# rpm -e mlnx-openmpi_gcc
```

Step 3. Install the new MXM and compile the Open MPI with it.

To run Open MPI without MXM, run:

```
% mpirun -mca mtl ^mxm --mca pml ^yalla <...>
```

When upgrading to MXM v3.6, Open MPI compiled with the previous versions of the MXM should be recompiled with MXM v3.6.

5.3 Running Open MPI with pml “yalla”

The pml “yalla” layer in the Mellanox's Open MPI v1.8 is used to reduce overhead by cutting through layers and using MXM directly. Consequently, for messages < 4K in size, it yields a latency improvement of up to 5%, message rate of up to 50% and bandwidth of up to 45%.

Open MPI’s default behavior is to run with pml 'yalla'. To directly use MXM with it, perform the steps below.

➢ ***To run MXM with the pml “yalla”:***

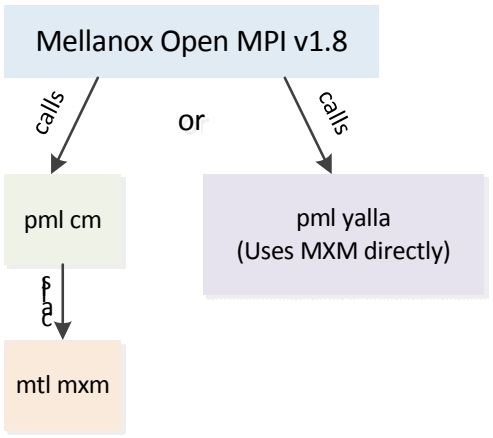
Step 1. Download the HPC-X package from the Mellanox site (Mellanox's Open MPI v1.10 has been integrated into HPC-X package).

http://www.mellanox.com/page/products_dyn?product_family=189&mtag=hpc-x

Step 2. Use the new yalla pml.

```
% mpirun -mca pml yalla
```

pml "yalla" uses MXM directly by default. If the pml "yalla" is not used, Open MPI will use MXM through "pml cm" and "mtl mxm" (see figure below).



MXM is selected automatically starting from any Number of Processes (NP) when using Open MPI v1.8.x and above.

For older Open MPI versions, use the command below.

➢ ***To activate MXM for any NP, run:***

```
% mpirun -mca mtl_mxm_np 0 <...other mpirun parameters ...>
```

As of Open MPI v1.8, MXM is selected when the number of processes is higher than 0. i.e. by default.

5.4 Tuning MXM Settings

The default MXM settings are already optimized. To check the available MXM parameters and their default values, run the `$HPCX_MXM_DIR/bin/mxm_dump_config -f` utility which is part of the MXM RPM.

MXM parameters can be modified in one of the following methods:

- Modifying the default MXM parameters value as part of the mpirun:

```
% mpirun -x MXM_UD_RX_MAX_BUFFS=128000 <...>
```

- Modifying the default MXM parameters value from SHELL:

```
% export MXM_UD_RX_MAX_BUFFS=128000
% mpirun <...>
```

5.5 Configuring Multi-Rail Support

Multi-Rail support enables the user to use more than one of the active ports on the card, making better use of system resources, allowing increased throughput.

Multi-Rail support as of MXM v3.5 allows different processes on the same host to use different active ports. Every process can only use one port (as opposed to MXM v1.5).

➢ ***To configure dual rail support:***

- Specify the list of ports you would like to use to enable multi rail support.

```
-x MXM_RDMA_PORTS=cardName:portNum
```

or

```
-x MXM_IB_PORTS=cardName:portNum
```

For example:

```
-x MXM_IB_PORTS=mlx5_0:1
```

It is also possible to use several HCAs and ports during the run (separated by a comma):

```
-x MXM_IB_PORTS=mlx5_0:1,mlx5_1:1
```

MXM will bind a process to one of the HCA ports from the given ports list according to the MXM_IB_MAP_MODE parameter (for load balancing).

Possible values for MXM_IB_MAP_MODE are:

- first - [Default] Maps the first suitable HCA port to all processes
- affinity - Distributes the HCA ports evenly among processes based on CPU affinity
- nearest - Tries to find the nearest HCA port based on CPU affinity

You may also use an asterisk (`*`) and a question mark (`?`) to choose the HCA and the port you would like to use.

- * - use all active cards/ports that are available
- ? - use the first active card/port that is available

For example:

```
-x MXM_IB_PORTS=*:?
```

will take all the active HCAs and the first active port on each of them.

5.6 Configuring MXM over the Ethernet Fabric

➢ ***To configure MXM over the Ethernet fabric:***

Step 1. Make sure the Ethernet port is active.

```
% ibv_devinfo
```

ibv_devinfo displays the list of cards and ports in the system. Make sure (in the ibv_devinfo output) that the desired port has Ethernet at the `link_layer` field and that its state is `PORT_ACTIVE`.

Step 2. Specify the ports you would like to use, if there is a non Ethernet active port in the card.

```
-x MXM_RDMA_PORTS=mlx4_0:1
```

or

```
-x MXM_IB_PORTS=mlx4_0:1
```

Please note that the MXM_RDMA_PORTS and MXM_IB_PORTS parameters are alias that mean the same used in both Section 5.5 (page 30) and Section 5.6 (page 32).

`MXM_IB_PORTS` and `MXM_RDMA_PORTS` are aliases which mean the same, but one is used in Section 5.5 (page 30) and the second is Section 5.6 (page 32).

5.7 Running MXM with RoCE

MXM supports RDMA over Converged Ethernet (RoCE). Remote Direct Memory Access (RDMA) is the remote memory management capability that allows server-to-server data movement directly between application memory without any CPU involvement. RDMA over Converged Ethernet (RoCE) is a mechanism to provide this efficient data transfer with very low latencies on lossless Ethernet networks. With advances in data center convergence over reliable Ethernet, ConnectX® Ethernet adapter cards family with RoCE uses the proven and efficient RDMA transport to provide the platform for deploying RDMA technology in mainstream data center application at 10GigE and 40GigE link-speed. ConnectX® Ethernet adapter cards family with its hardware offload support takes advantage of this efficient RDMA transport (InfiniBand) services over Ethernet to deliver ultra-low latency for performance-critical and transaction intensive applications such as financial, database, storage, and content delivery networks.

5.7.1 Running MXM with RoCE v1

To use RoCE with MXM, the desired Ethernet port must be specified (with the MXM_IB_PORTS parameter).

```
mpirun --mca pml yalla -x MXM_RDMA_PORTS=mlx4_0:2 ...
```

For further information on the pml yalla, please refer to Section 5.3, “Running Open MPI with pml “yalla””, on page 29.

If using older versions of MXM, the following additional environment parameter might be required to be set:

```
mpirun --mca pml yalla -x MXM_TLS=self,shm,ud -x MXM_RDMA_PORTS=mlx4_0:2 ...
```

5.7.2 Running MXM with RoCE v2

To enable RoCE v2, add the following parameter to the command line.

```
mpirun --mca pml yalla -x MXM_RDMA_PORTS=mlx4_0:2  -x MXM_IB_GID_INDEX=1 ...
```

5.7.2.1 Using RoCE v2 in Open MPI

RoCE v2 is supported in Open MPI as of v1.8.8.

To use it, enable RoCE v2 in the command line:

```
$ mpirun --mca pml ob1 --mca btl openib,self,sm --mca btl_openib_cpc_include rdmacm --mca
btl_openib_rroce_enable 1 ...
```

5.8 Configuring MXM over Different Transports

MXM v3.6 supports the following transports.

- Intra node communication via Shared Memory with KNEM support
- Unreliable Datagram (UD)
- Reliable Connected (RC)
- SELF transport - a single process communicates with itself
- Dynamically Connected Transport (DC)

  Note: DC is supported on Connect-IB® HCAs with MLNX_OFED v2.1-1.0.0 and higher.

  To use DC set the following:

  - in the command line:

    ```
    % mpirun -x MXM_TLS=self,shm,dc
    ```

  - from the SHELL:

    ```
    % export MXM_TLS=self,shm,dc
    ```

By default the transports (TLS) used are: `MXM_TLS=self,shm,ud`

5.9 Configuring Service Level Support

Service Level enables Quality of Service (QoS). If set, every InfiniBand endpoint in MXM will generate a random Service Level (SL) within the given range, and use it for outbound communication.

Setting the value is done via the following environment parameter:

```
export MXM_IB_NUM_SLS=1
```

Available Service Level values are 1-16 where the default is 1.

You can also set a specific service level to use. To do so, use the `MXM_IB_FIRST_SL` parameter together with `MXM_IB_NUM_SLS=1` (which is the default).

For example:

```
% mpirun -x MXM_IB_NUM_SLS=1 -x MXM_IB_FIRST_SL=3 …
```

where a single SL is being used and the first SL is 3.

5.10 Adaptive Routing for UD Transport

Adaptive Routing (AR) enables the switch to select the output port based on the port's load.

Adaptive Routing for the UD transport layer enable out of order packet arrival.

- When the `MXM_UD_RX_OOO=n`, parameter is set to "`n`", the out of order packet indicates a packet loss and triggers MXM UD flow control/congestion avoidance.
- When the parameter `MXM_UD_RX_OOO=y`, is set to "`y`" MXM will queue out of order packets until it is possible to process them in order instead of assuming a packet loss.

To configure adaptive routing one must use OpenSM with adaptive route manager plugin and a switch with Mellanox OS. AR support in UD is set to ON by default.

➢ ***To disable Adaptive Routing for UD transport:***

```
% mpirun -x MXM_UD_RX_OOO=y ...
```

5.11 Support for a Non-Base LID

MXM enables the user to use multiple LIDs per port when the LID Mask Control (LMC) configured in the fabric is higher than zero. By default, MXM will use all the possible LIDs that are enabled except for the LMC (from zero to 2^LMC) unless their number is higher than the `MXM_IB_MAX_PATH_BITS` parameter in which case the latter value will be used (this number is derived from the number of ports on a switch). MXM also enables the user to specify a list of LIDs to use via the `MXM_IB_LID_PATH_BITS` parameter.

The usage of multiple LIDs enables MXM to distribute the traffic in the fabric over multiple LIDs and therefore achieve a better usage of the bandwidth. It prevents the overloading of single link and results in an improved performance and more balanced traffic.

5.12 MXM Performance Tuning

MXM uses the following features to improve performance:

- **Bulk Connections**

  The `mxm_ep_wireup` and `mxm_ep_powerdown` functions were added to the MXM API to allow pre-connection establishment for MXM. This will enable MXM to create and connect all the required connections for the future communication during the initialization stage rather than creating the connections between the peers in an on-demand manner.

Please note that the usage of these parameters is only possible when using MXM with "`-mca pml cm --mca mtl mxm`", not when running with the `yalla pml`.

Bulk connection is the default for establishing connection for MXM in the Open MPI,'mtl mxm' layer.

When running an application that has a sparse communication pattern, it is recommended to disable Bulk Connections.

➢ ***To enable Bulk Connections:***

```
% mpirun -mca mtl_mxm_bulk_connect 1 -mca mtl_mxm_bulk_disconnect 1 ...
```

➢ ***To disable Bulk Connections:***

```
% mpirun -mca mtl_mxm_bulk_connect 0 -mca  mtl_mxm_bulk_disconnect 0 ...
```

- **Solicited event interrupt for the rendezvous protocol**

The solicited event interrupt for the rendezvous protocol improves performance for applications which have large messages communication overlapping with computation. This feature is disabled by default.

➢ ***To enable Solicited event interrupt for the rendezvous protocol:***

```
% mpirun -x MXM_RNDV_WAKEUP_THRESH=512k ...
```

*<thresh>: Minimal message size which will trigger an interrupt on the remote side, to switch the remote process from computation phase and force it to handle MXM communication.

5.13 MXM Environment Parameters

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_LOG_LEVEL	• FATAL • ERROR • INFO • DEBUG • TRACE • REQ • DATA • ASYNC • FUNC • POLL • WARN (default)	MXM logging level. Messages with a level higher or equal to the selected will be printed.
MXM_LOG_FILE	String	If not empty, MXM will print log messages to the specified file instead of stdout. The following substitutions are performed on this string: • %p - Replaced with process ID • %h - Replaced with host name Value: String.
MXM_LOG_BUFFER	1024	Buffer size for a single log message. Value: memory units: <number>[b|kb|mb|gb]
MXM_LOG_DATA_SIZE	0	How much of the packet payload to print, at most, in data mode. Value: unsigned long.
MXM_HANDLE_ERRORS	• None: No error handling • Freeze: Freeze and wait for a debugger • Debug: attach debugger • bt: print backtrace	Error handling mode.
MXM_ERROR_SIGNALS	• ILL • SEGV • BUS • FPE • PIPE	Signals which are considered an error indication and trigger error handling. Value: comma-separated list of: system signal (number or SIGxxx)

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_GDB_COMMAND	gdb	If non-empty, attaches a gdb to the process in case of error, using the provided command. Value: string
MXM_DEBUG_SIGNO	HUP	Signal number which causes MXM to enter debug mode. Set to 0 to disable. Value: system signal (number or SIGxxx)
MXM_ASYNC_INTERVAL	50000.00us	Interval of asynchronous progress. Lower values may make the network more responsive, at the cost of higher CPU load. Value: time value: <number>[s|us|ms|ns]
MXM_ASYNC_SIGNO	ALRM	Signal number used for async signaling. Value: system signal (number or SIGxxx)
MXM_STATS_DEST	• udp:<host>[:<port>] - send over UDP to the given host:port. • stdout: print to standard output. • stderr: print to standard error. • file:<filename>[:bin] - save to a file (%h: host, %p: pid, %c: cpu, %t: time, %e: exe)	Destination to send statistics to. If the value is empty, statistics are not reported.
MXM_STATS_TRIGGER	• timer:<interval>: dump in specified intervals. • exit: dump just before program exits (default)	Trigger to dump statistics Value: string

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_MEMTRACK_DEST	• file:<filename>: save to a file (%h: host, %p: pid, %c: cpu, %t: time, %e: exe) • stdout: print to standard output • stderr: print to standard error	Memory tracking report output destination. If the value is empty, results are not reported.
MXM_INSTRUMENT	• %h: host • %p: pid • %c: cpu • %t: time • %e: exe.	File name to dump instrumentation records to. Value: string
MXM_INSTRUMENT_SIZE	1048576	Maximal size of instrumentation data. New records will replace old records. Value: memory units: <number>[b|kb|mb|gb]
MXM_PERF_STALL_LOOPS	0	Number of performance stall loops to be performed. Can be used to normalize profile measurements to packet rate Value: unsigned long
MXM_ASYNC_MODE	• Signal • none • Thread (default)	Asynchronous progress method Value: [none|signal|thread]

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_MEM_ALLOC	• cpages: Contiguous pages, provided by Mellanox-OFED. • hugetlb - Use System V shared memory API for getting pre-allocated huge pages. • mmap: Use private anonymous mmap() to get free pages. • libc: Use libc's memory allocation primitives. • sysv: Use system V's memory allocation.	Memory allocators priority. Value: comma-separated list of: [libc|hugetlb|cpages|mmap|sysv]
MXM_MEM_ON_DEMAND_MAP	• n: disable • y: enable	Enable on-demand memory mapping. USE WITH CARE! It requires calling mxm_mem_unmap() when any buffer used for communication is unmapped, otherwise data corruption could occur. Value: <y|n>
MXM_INIT_HOOK_SCRIPT	-	Path to the script to be executed at the very beginning of MXM initialization Value: string
MXM_SINGLE_THREAD	• y - single thread • n - not single thread	Mode of the thread usage. Value: <y|n>

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_SHM_KCOPY_MODE	• off: Don't use any kernel copy mode. • knem: Try to use knem. If it fails, default to 'off'. • autodetect: If knem is available, first try to use knem. If it fails, default to 'off' (default)	Modes for using to kernel copy for large messages.
MXM_IB_PORTS	*:*	Specifies which Infiniband ports to use. Value: comma-separated list of: IB port: <device>:<port_num>
MXM_EP_NAME	%h:%p	Endpoint options. Endpoint name used in log messages. Value: string
MXM_TLS	• self • shm • ud	Comma-separated list of transports to use. The order is not significant. Value: comma-separated list of: [self|shm|rc|dc|ud|oob]
MXM_ZCOPY_THRESH	2040	Threshold for using zero copy. Value: memory units: <number>[b|kb|mb|gb]
MXM_IB_CQ_MODERATION	64	Number of send WREs for which a CQE is generated. Value: unsigned
MXM_IB_CQ_WATERMARK	127	Consider ep congested if poll cq returns more than n wqes. Value: unsigned
MXM_IB_DRAIN_CQ	• n • y	Poll CQ till it is completely drained of completed work requests. Enabling this feature may cause starvation of other endpoints Value: <y|n>
MXM_IB_RESIZE_CQ	• n • y	Allow using resize_cq(). Value: <y|n>

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_IB_TX_BATCH	16	Number of send WREs to batch in one post-send list. Larger values reduce the CPU usage, but increase the latency because we might need to process lots of send completions at once. Value: unsigned
MXM_IB_RX_BATCH	64	Number of post-receives to be performed in a single batch. Value: unsigned
MXM_IB_MAP_MODE	• first: Map the first suitable HCA port to all processes (default). • affinity: Distribute evenly among processes based on CPU affinity. • nearest: Try finding nearest HCA port based on CPU affinity. • round-robin	HCA ports to processes mapping method. Ports not supporting process requirements (e.g. DC support) will be skipped. Selecting a specific device will override this setting.
MXM_IB_NUM_SLS	1: (default)	Number of InfiniBand Service Levels to use. Every InfiniBand endpoint will generate a random SL within the given range FIRST_SL..(FIRST_SL+NUM_SLS-1), and use it for outbound communication. applicable values are 1 through 16. Value: unsigned

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_IB_WC_MODE	• wqe: Use write combining to post full WQEs (default). • db: Use write combining to post doorbell. • flush: Force flushing CPU write combining buffers.wqe • flush (default)	Write combining mode flags for Infini-Band devices. Using write combining for 'wqe' improves latency performance due to one less wqe fetch. Avoiding 'flush' relaxes CPU store ordering, and reduces overhead. Write combining for 'db' is meaningful only when used without 'flush'. Value: comma-separated list of: [wqe|db|flush]
MXM_IB_LID_PATH_BITS	-: (default) 0 <= value< 2^(LMC) - 1	List of InfiniBand Path bits separated by comma (a,b,c) which will be the low portion of the LID, according to the LMC in the fabric. If no value is given, MXM will use all the available offsets under the value of MXM_IB_MAX_PATH_BITS To disable the usage of LMC, please set this value to zero. Value: comma-separated list of: unsigned
MXM_IB_MAX_PATH_BITS	18: (default)	Number of offsets to use as lid_path_bits in case 2^LMC is larger than this value. This value derives from the number of ports on the switch. Value: unsigned
MXM_IB_FIRST_SL	0-15	The first Infiniband Service Level num-ber to use. Value: unsigned
MXM_IB_CQ_STALL	100	CQ stall loops for SandyBridge far socket. Value: unsigned
MXM_UD_ACK_TIMEOUT	300000.00us	Timeout for getting an acknowledgment for sent packet. Value: time value: <num-ber>[s|us|ms|ns]

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_UD_FAST_ACK_TIME-OUT	1024.00us	Timeout for getting an acknowledgment for sent packet. Value: time value: <number>[s|us|ms|ns]
MXM_FAST_TIMER_RESO-LUTION	64.00us	Resolution of ud fast timer.The value is treated as a recommendation only. Real resolution may differ as mxm rounds up to power of two. Value: time value: <number>[s|us|ms|ns]
MXM_UD_INT_MODE	rx	Traffic types to enable interrupt for. Value: comma-separated list of: [rx|tx]
MXM_UD_INT_THRESH	20000.00us	The maximum amount of time that may pass following an mxm call, after which interrupts will be enabled. Value: time value: <number>[s|us|ms|ns]
MXM_UD_WINDOW_SIZE	1024	The maximum number of unacknowledged packets that may be in transit. Value: unsigned
MXM_UD_MTU	65536	Maximal UD packet size. The actual MTU is the minimum of this value and the fabric MTU. Value: memory units: <number>[b|kb|mb|gb]
MXM_UD_CA_ALGO	• none: no congestion avoidance • bic: binary increase (default)	Use congestion avoidance algorithm to dynamically adjust send window size. Value: [none|bic]
MXM_UD_CA_LOW_WIN	0	Use additive increase multiplicative decrease congestion avoidance when current window is below this threshold. Value: unsigned
MXM_UD_RX_QUEUE_LEN	4096	Length of receive queue for UD QPs. Value: unsigned
MXM_UD_RX_MAX_BUFS	-1	Maximal number of receive buffers for one endpoint. -1 is infinite. Value: integer

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_UD_RX_MAX_INLINE	0	Maximal size of data to receive as inline. Value: memory units: <number>[b|kb|mb|gb]
MXM_UD_RX_DROP_RATE	0	If nonzero, network packet loss will be simulated by randomly ignoring one of every X received UD packets. Value: unsigned
MXM_UD_RX_OOO	• n • y	If enabled, keep packets received out of order instead of discarding them. Must be enabled if network allows out of order packet delivery, for example, if Adaptive Routing is enabled Value: <y|n>
MXM_UD_TX_QUEUE_LEN	128	Length of send queue for UD QPs. Value: unsigned
MXM_UD_TX_MAX_BUFS	32768	Maximal number of send buffers for one endpoint. -1 is infinite. Value: integer
MXM_UD_TX_MAX_INLINE	128	Bytes to reserve in TX WQE for inline data. Messages which are small enough will be sent inline. Value: memory units: <number>[b|kb|mb|gb]
MXM_CIB_PATH_MTU	default	Path MTU for CIB QPs. Possible values are: default, 512, 1024, 2048, 4096. Setting “default” will select the best MTU for the device. Value: [default|512|1024|2048|4096]
MXM_CIB_HARD_ZCOPY_-THRESH	16384	Threshold for using zero copy. Value: memory units: <number>[b|kb|mb|gb]
MXM_CIB_MIN_RNR_TIMER	25	InfiniBand minimum receiver not ready timer, in seconds (must be >= 0 and <= 31)
MXM_CIB_TIMEOUT	20	InfiniBand transmit timeout, plugged into formula: 4.096 microseconds * (2 ^ timeout) (must be >= 0 and <= 31) Value: unsigned

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_CIB_MAX_RDMA_DST_OPS	4	InfiniBand maximum pending RDMA destination operations (must be >= 0) Value: unsigned
MXM_CIB_RNR_RETRY	7	InfiniBand “receiver not ready” retry count, applies **ONLY** for SRQ/XRC queues. (must be >= 0 and <= 7: 7 = “infinite”) Value: unsigned
MXM_UD_RNDV_THRESH	262144	UD threshold for using rendezvous protocol. Smaller value may harm performance but excessively large value can cause a deadlock in the application. Value: memory units: <number>[b|kb|mb|gb]
MXM_UD_TX_NUM_SGE	3	Number of SG entries in the UD send QP. Value: unsigned
MXM_UD_HARD_ZCOPY_THRESH	65536	Threshold for using zero copy. Value: memory units: <number>[b|kb|mb|gb]
MXM_CIB_RETRY_COUNT	7	InfiniBand transmit retry count (must be >= 0 and <= 7) Value: unsigned
MXM_CIB_MSS	4224	Size of the send buffer. Value: memory units: <number>[b|kb|mb|gb]
MXM_CIB_TX_QUEUE_LEN	256	Length of send queue for RC QPs. Value: unsigned
MXM_CIB_TX_MAX_BUFS	-1	Maximal number of send buffers for one endpoint. -1 is infinite. **Warning:** Setting this param with value != -1 is a dangerous thing in RC and could cause deadlock or performance degradation Value: integer
MXM_CIB_TX_CQ_SIZE	16384	Send CQ length. Value: unsigned

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_CIB_TX_MAX_INLINE	128	Bytes to reserver in TX WQE for inline data. Messages which are small enough will be sent inline. Value: memory units: <number>[b|kb|mb|gb]
MXM_CIB_TX_NUM_SGE	3	Number of SG entries in the RC QP. Value: unsigned
MXM_CIB_USE_EAGER_RDMA	y	Use RDMA WRITE for small messages. Value: <y|n>
MXM_CIB_EAGER_RDMA_THRESHOLD	16	Use RDMA for short messages after this number of messages are received from a given peer, must be >= 1 Value: unsigned long
MXM_CIB_MAX_RDMA_CHANNELS	8	Maximum number of peers allowed to use RDMA for short messages, must be >= 0 Value: unsigned
MXM_CIB_EAGER_RDMA_BUFFS_NUM	32	Number of RDMA buffers to allocate per rdma channel, must be >= 1 Value: unsigned
MXM_CIB_EAGER_RDMA_BUFF_LEN	4224	Maximum size (in bytes) of eager RDMA messages, must be >= 1 Value: memory units: <number>[b|kb|mb|gb]
MXM_SHM_RX_MAX_BUFFERS	-1	Maximal number of receive buffers for endpoint. -1 is infinite. Value: integer
MXM_SHM_RX_MAX_MEDIUM_BUFFERS	-1	Maximal number of medium sized receive buffers for one endpoint. -1 is infinite. Value: integer
MXM_SHM_FIFO_SIZE	64	Size of the shmem tl's fifo. Value: unsigned
MXM_SHM_WRITE_RETRY_COUNT	64	Number of retries in case where cannot write to the remote process. Value: unsigned

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_SHM_READ_RETRY_COUNT	64	Number of retries in case where cannot read from the shmem FIFO (for multi-thread support). Value: unsigned
MXM_SHM_HUGETLB_MODE	• y: Allocate memory using huge pages only. • n: Allocate memory using regular pages only. • try: Try to allocate memory using huge pages and if it fails, allocate regular pages (default).	Enable using huge pages for internal shared memory buffers Values: <yes|no|try>
MXM_SHM_RX_BUFF_LEN	8192	Maximum size (in bytes) of medium sized messages, must be >= 1 Value: memory units: <number>[b|kb|mb|gb]
MXM_SHM_HARD_ZCOPY_THRESH	2048	Threshold for using zero copy. Value: memory units: <number>[b|kb|mb|gb]
MXM_SHM_RNDV_THRESH	65536	SHM threshold for using rendezvous protocol. Smaller value may harm performance but too large value can cause a deadlock in the application. Value: memory units: <number>[b|kb|mb|gb]
MXM_SHM_KNEM_MAX_SIMULTANEOUS	0	Maximum number of simultaneous ongoing knem operations to support in shmem tl. Value: unsigned
MXM_SHM_KNEM_DMA_CHUNK_SIZE	67108864	Size of a single chunk to be transferred in one dma operation. Larger values may not work since they are not supported by the dma engine Value: memory units: <number>[b|kb|mb|gb]

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_SHM_RELEASE_FIFO_FACTOR	0.500	Frequency of resource releasing on the receiver's side in shmem tl. This value refers to the percentage of the fifo size. (must be >= 0 and < 1) Value: floating point number
MXM_TM_UPDATE_THRESHOLD_MASK	8	Update bit-mask length for connections traffic counters.(must be >= 0 and <= 32: 0 - always update, 32 - never update.)# Value: bit count
MXM_TM_PROMOTE_THRESHOLD	5	Relative threshold (percentage) of traffic for promoting connections, Must be >= 0. Value: unsigned
MXM_TM_PROMOTE_BACKOFF	1	Exponential backoff degree (bits) for all counters upon a promotion, Must be >= 0 and <=32. Value: unsigned
MXM_RC_QP_LIMIT	64	Maximal amount of RC QPs allowed (Negative for unlimited). Value: integer
MXM_DC_QP_LIMIT	64	Maximal amount of DC QPs allowed (Negative for unlimited). Value: integer
MXM_DC_RECV_INLINE	• 128 • 512 • 1024 • 2048 • 4096	Bytes to reserve in CQE for inline data. In order to allow for inline data, -x MLX5_CQE_SIZE=128 must also be specified. Value: memory units: <number>[b|kb|mb|gb]
MXM_CIB_RNDV_THRESH	16384	CIB threshold for using rendezvous protocol. Smaller value may harm performance but excessively large value can cause a deadlock in the application. Value: memory units: <number>[b|kb|mb|gb]
MXM_SHM_USE_KNEM_DMA	• n • y	Whether or not to offload to the DMA engine when using KNEM Value: <y|n>
MXM_IB_GID_INDEX	• 0	GID index to use as local Ethernet port address.

Table 7 - MXM Environment Parameters

Variable	Valid Values and Default Values	Description
MXM_IB_USE_GRH	• yes • no • try	Whether or not to use global routing: • yes - Always use. • no - Never use. • try - Use if needed. This parameter needs to be set to 'yes' when one of the following is used: 1. Socket Direct 2. Multi-Host 3. SR-IOV

5.14 MXM Utilities

When MXM is used as part of HPC-X sotfware toolkit, the MXM utilities can be found at the `$HPCX_HOME/mxm/bin` directory.

When MXM is used as part of MLNX_OFED driver, the MXM utilities can be found at the `/opt/mellanox/mxm/bin` directory.

5.14.1 mxm_dump_config

Enables viewing of all the environment parameters that MXM uses.

To see all the parameters, run: `$HPCX_HOME/mxm/bin/mxm_dump_config -f`.

For further information, run: `$HPCX_HOME/mxm/bin/mxm_dump_config -help`

Environment parameters can be set by using the “export” command.
For example, to set the MXM_TLS environment parameter, run:

```
% export MXM_TLS=<...>
```

5.14.2 mxm_perftest

A client-server based application which is designed to test MXM's performance and sanity checks on MXM.

To run it, two terminals are required to be opened, one on the server side and one on the client side.

The working flow is as follow:

1. The server listens to the request coming from the client.
2. Once a connection is established, MXM sends and receives messages between the two sides according to what the client requested.
3. The results of the communications are displayed.

For further information, run: `$HPCX_HOME/mxm/bin/mxm_perftest -help`.

Example:

- From the server side run: `$HPCX_HOME/mxm/bin/mxm_perftest`
- From the client side run:
  `$HPCX_HOME/mxm/bin/mxm_perftest <server_host_name> -t send_lat`

Among other parameters, you can specify the test you would like to run, the message size and the number of iterations.

6 Unified Communication - X Framework Library

Unified Communication - X Framework (UCX) is currently at Beta level.

6.1 Overview

Unified Communication - X Framework (UCX) is a new acceleration library, integrated into the Open MPI (as a pml layer) and available as part of HPC-X. It is an open source communication library designed to achieve the highest performance for HPC applications. UCX has a broad range of optimizations for achieving low-software overheads in communication path which allow near native-level performance.

UCX supports receive side tag matching, one-sided communication semantics, efficient memory registration and a variety of enhancements which increase the scalability and performance of HPC applications significantly.

UCX supports the following transports:

- InfiniBand transports:
  - Unreliable Datagram (UD)
  - Reliable connected (RC)
  - Dynamically Connected (DC)

    DC is supported on Connect-IB® HCAs with MLNX_OFED v2.1-1.0.0 and higher.
  - Accelerated verbs
- Shared Memory communication with support for KNEM, CMA and XPMEM
- RoCE
- TCP

For further information on UCX, please refer to: https://github.com/openucx/ucx

6.2 Configuring UCX

➢ ***To use UCX with Open MPI:***

```
$mpirun --mca pml ucx …
```

➢ ***To use UCX with Open SHMEM:***

```
$oshrun --mca spml ucx …
```

6.3 Tuning UCX Settings

The default UCX settings are already optimized. To check the available UCX parameters and their default values, run the `'$HPCX_UCX_DIR/bin/ucx_info -f'` utility.

The UCX parameters can be modified in one of the following methods:

- Modifying the default UCX parameters value as part of the mpirun:

```
$mpirun -x UCX_RC_VERBS_RX_MAX_BUFS=128000 <...>
```

- Modifying the default UCX parameters value from SHELL:

```
$ export UCX_RC_VERBS_RX_MAX_BUFS=128000
$ mpirun <...>
```

- Selecting the transports to use from the command line:

```
$mpirun -mca pml ucx -x UCX_TLS=sm,rc_x …
```

  The above command will select pml ucx and set its transports for usage, shared memory and accelerated verbs.

- Selecting the devices to use from the command line:

```
$mpirun -mca pml ucx -x UCX_NET_DEVICES=mlx5_1:1
```

  The above command will select pml ucx and set the HCA for usage, mlx5_1, port 1.

6.4 UCX Utilities

6.4.1 ucx_perftest

A client-server based application which is designed to test UCX's performance and sanity checks.

To run it, two terminals are required to be opened, one on the server side and one on the client side.

The working flow is as follow:

1. The server listens to the request coming from the client.
2. Once a connection is established, UCX sends and receives messages between the two sides according to what the client requested.
3. The results of the communications are displayed.

For further information, run: `$HPCX_HOME/ucx/bin/uxc_perftest -help`.

Example:

- From the server side run: `$HPCX_HOME/ucx/bin/ucx_perftest`
- From the client side run:
  `$HPCX_HOME/ucx/bin/ucx_perftest <server_host_name> -t send_lat`

Among other parameters, you can specify the test you would like to run, the message size and the number of iterations.

7 PGAS Shared Memory Access Overview

The Shared Memory Access (SHMEM) routines provide low-latency, high-bandwidth communication for use in highly parallel scalable programs. The routines in the SHMEM Application Programming Interface (API) provide a programming model for exchanging data between cooperating parallel processes. The SHMEM API can be used either alone or in combination with MPI routines in the same parallel program.

The SHMEM parallel programming library is an easy-to-use programming model which uses highly efficient one-sided communication APIs to provide an intuitive global-view interface to shared or distributed memory systems. SHMEM's capabilities provide an excellent low level interface for PGAS applications.

A SHMEM program is of a single program, multiple data (SPMD) style. All the SHMEM processes, referred as processing elements (PEs), start simultaneously and run the same program. Commonly, the PEs perform computation on their own sub-domains of the larger problem, and periodically communicate with other PEs to exchange information on which the next communication phase depends.

The SHMEM routines minimize the overhead associated with data transfer requests, maximize bandwidth, and minimize data latency (the period of time that starts when a PE initiates a transfer of data and ends when a PE can use the data).

SHMEM routines support remote data transfer through:

- `"put"` operations - data transfer to a different PE
- `"get"` operations - data transfer from a different PE, and remote pointers, allowing direct references to data objects owned by another PE

Additional supported operations are collective broadcast and reduction, barrier synchronization, and atomic memory operations. An atomic memory operation is an atomic read-and-update operation, such as a fetch-and-increment, on a remote or local data object.

SHMEM libraries implement active messaging. The sending of data involves only one CPU where the source processor puts the data into the memory of the destination processor. Likewise, a processor can read data from another processor's memory without interrupting the remote CPU. The remote processor is unaware that its memory has been read or written unless the programmer implements a mechanism to accomplish this.

7.1 HPC-X Open MPI/OpenSHMEM

HPC-X Open MPI/OpenSHMEM programming library is a one-side communications library that supports a unique set of parallel programming features including point-to-point and collective routines, synchronizations, atomic operations, and a shared memory paradigm used between the processes of a parallel programming application.

HPC-X OpenSHMEM is based on the API defined by the OpenSHMEM.org consortium. The library works with the OpenFabrics RDMA for Linux stack (OFED), and also has the ability to utilize MellanoX Messaging libraries (MXM) as well as Mellanox Fabric Collective Accelerations (FCA), providing an unprecedented level of scalability for SHMEM programs running over InfiniBand.

7.2 Running HPC-X OpenSHMEM

7.2.1 Running HPC-X OpenSHMEM with MXM

MellanoX Messaging (MXM) library provides enhancements to parallel communication libraries by fully utilizing the underlying networking infrastructure provided by Mellanox HCA/switch hardware. This includes a variety of enhancements that take advantage of Mellanox networking hardware including:

- Multiple transport support including RC, DC and UD
- Proper management of HCA resources and memory structures
- Efficient memory registration
- One-sided communication semantics
- Connection management
- Receive side tag matching
- Intra-node shared memory communication

These enhancements significantly increase the scalability and performance of message com-muni-cations in the network, alleviating bottlenecks within the parallel communication libraries

7.2.1.1 Enabling MXM for HPC-X OpenSHMEM Jobs

➢ ***To enable MXM for SHMEM jobs for any PE:***

- Add the following MCA parameter to the `oshrun` command line.

```
-mca spml_ikrit_np <number>
```

➢ ***To force MXM usage:***

- Add the following MCA parameter `oshrun` command line.

```
-mca spml_ikrit_np 0
```

For additional MXM tuning information, please refer to the MellanoX Messaging Library README file found in the Mellanox website.

7.2.1.2 Working with Multiple HCAs

If there several HCAs in the system, MXM will choose the first HCA with the active port to work with. The HCA/port to be used can be specified by setting the `MXM_RDMA_PORTS` environment variable. The variable format is as follow: `MXM_RDMA_PORTS=hca_name:port,...`

For example, the following will cause MXM to use port one on two installed HCAs:

```
MXM_RDMA_PORTS=mlx4_0:1,mlx4_1:1
```

The environment variables must be run via the oshrun command line:

```
% oshrun -x MXM_RDMA_PORTS=mlx4_0:1 ...
```

7.2.2 Running HPC-X™ OpenSHMEM with FCA v2.x

The Mellanox Fabric Collective Accelerator (FCA) is a unique solution for offloading collective operations from the Message Passing Interface (MPI) or HPC-X OpenSHMEM process onto Mellanox InfiniBand managed switch CPUs. As a system-wide solution, FCA utilizes intelligence on Mellanox InfiniBand switches, Unified Fabric Manager and MPI nodes without requiring additional hardware. The FCA manager creates a topology based collective tree, and orchestrates an efficient collective operation using the switch-based CPUs on the MPI/HPC-X OpenSHMEM nodes.

FCA accelerates MPI/HPC-X OpenSHMEM collective operation performance by up to 100 times providing a reduction in the overall job runtime. Implementation is simple and transparent during the job runtime.

FCA v2.x is disabled by default and *must* be configured prior to using it from the HPC-X OpenSHMEM.

➢ ***To enable FCA by default in the HPC-X OpenSHMEM:***

1. Edit the `$HPCX_OSHMEM_DIR/etc/openmpi-mca-params.conf` file.
2. Set the `scoll_fca_enable` parameter to `1`.

```
scoll_fca_enable=1
```

3. Set the `scoll_fca_np` parameter to 0.

```
scoll_fca_np=0
```

➢ ***To enable FCA in the*** `oshrun` ***command line, add the following:***

```
-mca scoll_fca_enable=1
-mca scoll_fca_enable_np 0
```

➢ ***To disable FCA:***

```
-mca scoll_fca_enable 0 -mca coll_fca_enable 0
```

For more details on FCA installation and configuration, please refer to the FCA User Manual found in the Mellanox website.

7.2.3 Developing Application using HPC-X OpenSHMEM together with MPI

The SHMEM programming model can provide a means to improve the performance of latency-sensitive sections of an application. Commonly, this requires replacing MPI send/recv calls with `shmem_put/ shmem_get` and `shmem_barrier` calls. The SHMEM programming model can deliver significantly lower latencies for short messages than traditional MPI calls. An alternative to `shmem_get /shmem_put` calls can also be considered the MPI-2 MPI_Put/ MPI_Get functions.

An example of MPI-SHMEM mixed code.

```
/* example.c */

#include <stdlib.h>
#include <stdio.h>
#include "shmem.h"
#include "mpi.h"
int main(int argc, char *argv[])
{
 MPI_Init(&argc, &argv);
    start_pes(0);

 {
        int version = 0;
        int subversion = 0;
        int num_proc = 0;
        int my_proc = 0;
        int comm_size = 0;
        int comm_rank = 0;

        MPI_Get_version(&version, &subversion);
        fprintf(stdout, "MPI version: %d.%d\n", version, subversion);

        num_proc = _num_pes();
        my_proc = _my_pe();

        fprintf(stdout, "PE#%d of %d\n", my_proc, num_proc);

        MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
        MPI_Comm_rank(MPI_COMM_WORLD, &comm_rank);

        fprintf(stdout, "Comm rank#%d of %d\n", comm_rank, comm_size);
    }

    return 0;
}
```

7.2.4 HPC-X™ OpenSHMEM Tunable Parameters

HPC-X™ OpenSHMEM uses Modular Component Architecture (MCA) parameters to provide a way to tune your runtime environment. Each parameter corresponds to a specific function. The following are parameters that you can change their values to change the application's the function:

- memheap - controls memory allocation policy and thresholds
- scoll - controls HPC-X OpenSHMEM collective API threshold and algorithms
- spml - controls HPC-X OpenSHMEM point-to-point transport logic and thresholds
- atomic - controls HPC-X OpenSHMEM atomic operations logic and thresholds
- shmem - controls general HPC-X OpenSHMEM API behavior

➢ ***To display HPC-X OpenSHMEM parameters:***

1. Print all available parameters. Run:

```
% oshmem_info -a
```

2. Print HPC-X OpenSHMEM specific parameters. Run:

```
% oshmem_info --param shmem all
% oshmem_info --param memheap all
% oshmem_info --param scoll all
% oshmem_info --param spml all
% oshmem_info --param atomic all
```

7.2.4.1 OpenSHMEM MCA Parameters for Symmetric Heap Allocation

SHMEM memheap size can be modified by adding the `SHMEM_SYMMETRIC_HEAP_SIZE` parameter to the `oshrun` file. The default heap size is 256M.

➢ ***To run SHMEM with memheap size of 64M. Run:***

```
% oshrun -x SHMEM_SYMMETRIC_HEAP_SIZE=64M -np 512 -mca mpi_paffinity_alone 1 -bynode -
display-map -hostfile myhostfile example.exe
```

Memheap can be allocated with the following methods:

- sysv - system V shared memory API. Allocation with hugepages is curently not supported
- verbs - IB verbs allocator is used
- mmap - mmap() is used to allocate memory

By default HPC-X OpenSHMEM will try a to find the best possible allocator. The priority is verbs, sysv and mmap. It is possible to choose a specific memheap allocation method by running `-mca sshmem <name>`

7.2.4.2 Parameters Used to Force Connection Creation

Commonly SHMEM creates connection between PE lazily. That is at the sign of the first traffic.

➢ ***To force connection creating during startup:***

- Set the following MCA parameter.

```
-mca shmem_preconnect_all 1
```

Memory registration (ex: infiniband rkeys) information is exchanged between ranks during startup.

➢ ***To enable on demand memory key exchange:***

- Set the following MCA parameter.

```
-mca shmalloc_use_modex 0
```

7.3 Tuning OpenSHMEM Atomics Performance

HPC-X OpenSHMEM uses separate communication channel to perform atomic operations. By default this channel is disabled and uses RC transport.

When running on Connect-IB® adapter cards, it is recommended to use DC transport instead of RC.

Atomic tunable parameters:

- `-mca spml_ikrit_hw_rdma_channle 0|1` - default is 1 (enabled)
- `MXM_OSHMEM_HW_RDMA_TLS=rc|dc` - Decides what transport is used for atomic operations. Default is rc

7.4 Tuning MTU Size to the Recommended Value

The procedures described below apply to user using MLNX_OFED 1.5.3.-3.0.0 only.

When using MLNX_OFED 1.5.3-3.0.0, it is recommended to change the MTU to 4k. Whereas in MLNX_OFED 3.1-x.x.x and above, the MTU is already set by default to 4k.

➢ ***To check the current MTU support of an InfiniBand port, use the smpquery tool:***

```
# smpquery -D PortInfo 0 1 | grep -i mtu
```

If the MtuCap value is lower than 4K, enable it to 4K.

Assuming the firmware is configured to support 4K MTU, the actual MTU capability is further limited by the mlx4 driver parameter.

➢ ***To further tune it:***

1. Set the `set_4k_mtu` mlx4 driver parameter to 1 on all the cluster machines. For instance:

```
# echo "options mlx4_core set_4k_mtu=1" >> /etc/modprobe.d/mofed.conf
```

2. Restart openibd.

```
# service openibd restart
```

➢ ***To check whether the parameter was accepted, run:***

```
# cat /sys/module/mlx4_core/parameters/set_4k_mtu
```

To check whether the port was brought up with 4K MTU this time, use the smpquery tool again.

7.4.1 HPC Applications on Intel Sandy Bridge Machines

Intel Sandy Bridge machines have NUMA hardware related limitation which affects performance of HPC jobs utilizing all node sockets. When installing MLNX_OFED 3.1-x.x.x, an automatic workaround is activated upon Sandy Bridge machine detection, and the following message is printed in the job`s standard output device: "`mlx4: Sandy Bridge CPU was detected`"

➢ ***To disable MLNX_OFED 3.1-x.x.x Sandy Bridge NUMA related workaround:***

- Set the SHELL environment variable before launching HPC application. Run:

```
% export MLX4_STALL_CQ_POLL=0
% oshrun <...>
```

or

```
oshrun -x MLX4_STALL_CQ_POLL=0 <other params>
```

8 Unified Parallel C Overview

Unified Parallel C (UPC) is an extension of the C programming language designed for high performance computing on large-scale parallel machines.The language provides a uniform programming model for both shared and distributed memory hardware. The programmer is presented with a single shared, partitioned address space, where variables may be directly read and written by any processor, but each variable is physically associated with a single processor. UPC uses a Single Program Multiple Data (SPMD) model of computation in which the amount of parallelism is fixed at program startup time, typically with a single thread of execution per processor.

In order to express parallelism, UPC extends ISO C 99 with the following constructs:

- An explicitly parallel execution model
- A shared address space
- Synchronization primitives and a memory consistency model
- Memory management primitives

The UPC language evolved from experiences with three other earlier languages that proposed parallel extensions to ISO C 99: AC, Split-C, and Parallel C Preprocessor (PCP). UPC is not a superset of these three languages, but rather an attempt to distill the best characteristics of each. UPC combines the programmability advantages of the shared memory programming paradigm and the control over data layout and performance of the message passing programming paradigm.

HPC-X UPC is based on Berkely UPC package (see http://upc.lbl.gov/) and contains the following enhancements:

- GasNet library used within UPC integrated with Mellanox FCA which off-loads from UPC collective operations.
- GasNet library contains MXM conduit which offloads from UPC all P2P operations as well as some synchronization routines.

8.1 Compiling and Running the UPC Application

8.1.1 Compiling the UPC Application

➢ ***To build the UPC application:***

- Use the `upcc` compiler.

```
$ upcc -o app app.c
```

➢ ***To build the application with a debug info in UPC and GASNet:***

```
$ upcc -g -o app app.c
```

For further information on additional build options, please refer to the `upcc` man page.

8.1.2 Running the UPC Application

The UPC application can be run using the UPC execution command.

```
$ upcrun -shared-heap=<size_per_process>  \
-n <total_number_of_processes>   \
-N <machines_number_to_be_used>  \
-bind-threads <executable>
```

8.1.2.1 Basic upcrun Options

Each UPC process uses shared heap. The exact size of the required shared heap is application-specific. The amount of shared memory per UPC thread is controlled by the `-shared-heap` parameter or by the `UPC_SHARED_HEAP_SIZE` environment variable.

A hard limit (per UNIX process) of the shared memory heap can be set using the `-shared-heap-max` parameter or the `UPC_SHARED_HEAP_SIZE` environment variable. This constitutes an upper limit on the `-shared-heap` parameter. Setting this value too high can lead to long application startup times or memory exhaustion on some systems.

For optimal performance, you should bind (a.k.a pin) the UPC threads to the processors using the `-bind-threads` option or using the `UPC_BIND_THREADS` environment variable. These parameters are silently ignored on unsupported platforms.

The following is an example of running a binary with PPN=8 on 8 hosts.

```
$ upcrun -shared-heap=256M -n 64 -N 8 -bind-threads <my_application>
```

For further information on additional usage options, please refer to the `upcrun` man pages or go to http://upc.lbl.gov/docs/user/upcrun.html

8.1.2.2 Environment Variables

Any command line argument has its environment variable equivalent which is required as the BUPC supports direct execution using the schedulers.

Any environment variable that begins with `UPC_` or `GASNET_` is automatically propagated to all the nodes.

However, there is no Open MPI's equivalent of passing any environment variable to all the nodes, therefore, in order to pass an arbitrary environment variable to the BUPC or to your application, this variable has to be present in the environment where the UPC application is executed.

For example, add your environment variables to `~/.bashrc` or `~/.bash_profile`

8.2 FCA Runtime Parameters

The following parameters can be passed to "`upcrun`" in order to change FCA support behavior:

Table 8 - Runtime Parameters

Parameter	Description
`-fca_enable <0|1>`	Disables/Enables FCA support at runtime (default: disable).

Table 8 - Runtime Parameters

Parameter	Description
`-fca_np <value>`	Enables FCA support for collective operations if the number of processes in the job is greater than the `fca_np` value (default: 64).
`-fca_verbose <level>`	Sets verbosity level for the FCA modules
`-fca_ops <+/->[op_list]`	`op_list` - comma separated list of collective operations. • `-fca_ops <+/->[op_list]` - Enables/disables only the specified operations • `-fca_ops <+/->` - Enables/disables all operations By default all operations are enabled. Allowed operation names are: barrier (br), bcast (bt), reduce (rc), allgather (ag). Each operation can be also enabled/disabled via environment variable: • GASNET_FCA_ENABLE_BARRIER • GASNET_FCA_ENABLE_BCAST, • GASNET_FCA_ENABLE_REDUCE, **Note:** All the operations are enabled by default.

8.2.1 Enabling FCA Operations through Environment Variables in HPC-X UPC

This method can be used to control UPC FCA offload from environment using job scheduler srun utility. The valid values are: 1 - enable, 0 - disable.

➢ ***To enable a specific operation with shell environment variables in HPC-X UPC:***

```
% export GASNET_FCA_ENABLE_BARRIER=1
% export GASNET_FCA_ENABLE_BCAST=1
% export GASNET_FCA_ENABLE_REDUCE=1
```

8.2.2 Controlling FCA Offload in HPC-X UPC using Environment Variables

➢ ***To enable FCA module under HPC-X UPC:***

```
% export GASNET_FCA_ENABLE_CMD_LINE=1
```

➢ ***To set FCA verbose level:***

```
% export GASNET_FCA_VERBOSE_CMD_LINE=10
```

➢ ***To set the minimal number of processes threshold to activate FCA:***

```
% export GASNET_FCA_NP_CMD_LINE=1
```

HPC-X UPC contains modules configuration file (http://modules.sf.net) which can be found at `/opt/mellanox/bupc/2.2/etc/bupc_modulefile`.

8.3 Various Executable Examples

The following are various executable examples.

- ***To run a HPC-X UPC application without FCA support:***

```
% upcrun -np 128 -fca_enable 0 <executable filename>
```

- ***To run HPC-X UPC applications with FCA enabled for any number of processes:***

```
% export GASNET_FCA_ENABLE_CMD_LINE=1 GASNET_FCA_NP_CMD_LINE=0
% upcrun -np 64 <executable filename>
```

- ***To run HPC-X UPC application on 128 processes, verbose mode:***

```
% upcrun -np 128 -fca_enable 1 -fca_np 10 -fca_verbose 5 <executable filename>
```

- ***To run HPC-X UPC application, offload to FCA Barrier and Broadcast only:***

```
% upcrun -np 128 -fca_ops +barrier,bt <executable filename>
```

BUPC provided with HPC-X is not re-locatable. In order to use BUPC from a desired location, run post install procedure from HPCX BUPC source tarball: `bupc/contrib/relocate_bupc`.