Mellanox HPC-X™ Software Toolkit User Manual

Rev 2.3

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT ("PRODUCT(S)") AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES "ASIS" 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 WARRANTTHAT 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) 9703400
Fax: (408) 9703403

Table of Contents

List of Tables

List of Figures

Document Revision History

Revision	Date	Description

2.3	December 3, 2018	• Updated the following sections:
		  • Section 2.5, “HPC-X Environments”, on page 14
		  • Section 2.6, “HPC-X and Singularity”, on page 15
		  • Section 5.3, “Tuning UCX Settings”, on page 29
		• Removed the following chapter:
		  • Mellanox Messaging Library
		• Removed the following sections:
		  • Running HPC-X OpenSHMEM with MXM
		  • Tuning OpenSHMEM Atomics Performance with MXM

2.2	July 5, 2018	• Added the following sections:
		  • Section 2.6, “HPC-X and Singularity”, on page 15
		  • Section 5.4.7, “Memory in Chip (MEMIC)”, on page 35
		  • Section 4.4.9, “Limitations”, on page 26
		• Updated the following sections:
		  • Section 3.3, “Loading KNEM Module”, on page 17
		  • Section 5.3, “Tuning UCX Settings”, on page 29
		  • Section 5.4.2, “Single Root IO Virtualization (SR-IOV)”, on page 32
		  • Section 5.4.5.4, “Configuring CUDA Support - UCX”, on page 34
		  • Section 6.2, “Compiling Open MPI with MXM”, on page 39
		  • Section 6.3, “Running Open MPI with pml “yalla””, on page 40
		• Removed the following section:
		  • “Enabling HCOLL Topology Awareness”

Revision	Date	Description

2.1	February 28, 2018	• Added the following sections:
		  • Section 2.5, “HPC-X Environments”, on page 14
		  • Section 5.2.2, “Configuring UCX with XPMEM”, on page 29
		  • Section 5.4.5, “CUDA GPU”, on page 33
		  • Section 5.4.6, “Multi-Rail”, on page 35
		  • Section 6.2.1, “Running HPC-X OpenSHMEM with UCX”, on page 39
		  • Section 4.4.8, “Configuring NVIDIA® CUDA® GPUs Support - HCOLL”, on page 26
		  • Section 5.6, “Generating UCX Statistics for Open MPI/OpenSHMEM”, on page 36
		• Updated the following sections:
		  • Section 1.1, “HPC-X Package Contents”, on page 11
		  • Section 2.1, “Installing HPC-X”, on page 13
		  • Section 2.2, “Loading HPC-X Environment from Bash”, on page 13
		  • Section 2.4, “Loading HPC-X Environment from Modules”, on page 14
		  • Section 6.3, “Running Open MPI with pml “yalla””, on page 40
		  • Section 5.2, “Configuring UCX”, on page 28
		  • Section 5.3, “Tuning UCX Settings”, on page 29
		  • Section 5.4.1, “Hardware Tag Matching”, on page 31
		  • Section 6.2.2, “Developing Application using HPC-X OpenSHMEM together with MPI”, on page 39
		• Removed the following sections:
		  • 3.1 Profiling IB verbs API
		  • 7.2.3 Running HPC-X™ OpenSHMEM with FCA v2.x

2.0	October 30, 2017	• Added the following sections:
		  • Section 3.6, “Direct Launch of Open MPI and OpenSHMEM using SLURM 'srun'”, on page 19
		  • Section 5.4.1, “Hardware Tag Matching”, on page 31
		  • Section 5.4.2, “Single Root IO Virtualization (SR-IOV)”, on page 32
		  • Section 5.4.3, “Adaptive Routing”, on page 32
		  • Section 5.4.4, “Error Handling”, on page 33
		• Updated the following sections:
		  • Section 1.1, “HPC-X Package Contents”, on page 11

Revision	Date	Description

1.9	December 3, 2018	• Added the following sections:
		  • Section 5.1.1, “Supported CPU Architectures”, on page 28
		• Updated the following sections:
		  • Section 2.3, “Building HPC-X with the Intel Compiler Suite”, on page 13
		  • Section 5.1, “Overview”, on page 28
		  • Section 5.3, “Tuning UCX Settings”, on page 29
		• Removed the following sections:
		  • Unified Parallel C Overview and its subsections
		  • Starting FCA Manager from HPC-X
		  • Running MPI with FCA v2.5
		  • Running OpenSHMEM with FCA v2.5

1.8	April 18, 2017	• Added the following sections:
		  • Section 5, “Unified Communication - X Framework Library”, on page 28
		  • Section 5.2, “Configuring UCX”, on page 28
		  • Section 5.3, “Tuning UCX Settings”, on page 29
		  • Section 5.5.1, “ucx_perftest”, on page 35
		• 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 7, “MXM Environment Parameters,” on page 47

1.7.406	December 15, 2016	• Added the following section: • Section 4.4.9, “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 3.5, “IB-Router”, on page 18
		  • Section 4.4.7, “Enabling Mellanox SHARP Software Accelerated Collectives”, on page 25
		• 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 3.2, “Rebuilding Open MPI”, on page 16
		  • Updated the MXM version to 3.5

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 and SHMEM 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 UCX (Unified Communication X) and MXM (Mellanox Messaging), which accelerate the underlying send/receive (or put/get) messages. It also includes 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 memory and latency related efficiencies, assuring that the communication libraries are fully optimized with the Mellanox interconnect solutions.

Mellanox HPC-X™ allows OEMs 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

HPC-X package contains the following pre-compiled HPC packages:

Table 2 - HPC-X Package Contents

Components	Description
MPI	• Open MPI and OpenSHMEM v4.0.x (MPI-3.2 and OpenSHMEM v1.4 compliant)[a]. Open MPI and OpenSHMEM are available at: https://www.open-mpi.org/software/ompi/ • MPI profiler (IPM - open source tool from http://ipm-hpc.org/) • MPI tests (OSU, IMB, random ring, etc.)
HPC Acceleration Package	• FCA v4.2 (code name: "hcoll" - default)[b] • UCX v1.5 (default) • Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) 1.7.2

a. Open SHMEM v1.4 compliance is at beta level.
b. 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 4.3-x.x.x.x and later

Table 3 - HPC-X™ Requirements

Platform	Drivers and HCAs
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.tbz into your current working directory.

```
tar -xvf hpcx.tbz
```

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 v4.0.x. Each Open MPI version has its own module file which can be used to load the desired version.

The symbolic links `hpcx-init.sh` and `modulefiles/hpcx` point to the default version (Open MPIv4.0.x).

➢ ***To load Open MPI/OpenSHMEM v4.0.x based package:***

```
% source $HPCX_HOME/hpcx-init.sh
% hpcx_load
% env | grep HPCX
% mpicc $HPCX_MPI_TESTS_DIR/examples/hello_c.c -o $HPCX_MPI_TESTS_DIR/examples/hello_c
% mpirun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_c
% oshcc $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c.c -o $HPCX_MPI_TESTS_DIR/examples/
hello_oshmem_c
% oshrun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c
% 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=${HPCX_HOME}/ompi-icc \
--with-hcoll=${HPCX_HOME}/hcoll \
--with-ucx=${HPCX_HOME}/ucx \
--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 v4.0.x based package:***

```
% module use $HPCX_HOME/modulefiles
% module load hpcx
% mpicc $HPCX_MPI_TESTS_DIR/examples/hello_c.c -o $HPCX_MPI_TESTS_DIR/examples/hello_c
% mpirun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_c
% oshcc $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c.c -o $HPCX_MPI_TESTS_DIR/examples/
hello_oshmem_c
% oshrun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c
% module unload hpcx
```

2.5 HPC-X Environments

Staring from version 2.1, HPC-X toolkit is provided with three environments. You are to select the environment that meets your needs best.

- **Vanilla HPC-X** - `hpcx`

  This is the default option which is optimized for best performance for single-thread mode.

- **HPC-X with multi-threading support** - `hpcx-mt`

  This option enables multi-threading support in all of the HPC-X components.
  Please use this module in order to run multi-threaded applications.

- **HPC-X with NVIDIA® CUDA® GPUs support** - `hpcx-cuda`

  This option includes support for CUDA in UCX and HCOLL. It requires CUDA (v8.0 or v9.1), GDRCOPY and NCCL (v2.1) to be part of your system.

  Note: Open MPI is compiled without CUDA support.

When HPC-X is launched in an environment where there is no resource manager installed (slurm, pbs, etc.), or when it is launched from a compute node, the OMPI default rsh/ssh based launcher will be used. This would prevent the launch from propagating the library path to the compute nodes. Thus, make sure to pass LD_LIBRARY_PATH variable as follows:

```
% mpirun -x LD_LIBRARY_PATH -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_c
```

Note that only one of the three environments can be loaded to be run.

For information on how to load and use the additional environments, please refer to the HPC-X README file (embedded in the HPC-X package).

2.6 HPC-X and Singularity

HPC-X supports Singularity containerization technology, which helps deploying and running distributed applications without launching an entire virtual machine (VM) for each application.

For instructions on the technology and how to create a standalone Singularity container with MLNX_OFED and HPC-X inside, please visit:

- Placing HPC-X in a Singularity Container Community post
- $HPCX_HOME/utils/singularity/hpcx-singularity.md file inside HPC-X package

3 Running, Configuring and Rebuilding HPC-X™

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

Please refer to `$HPCX_HOME/sources/` and HPC-X README file for more information on building details.

3.1 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 IPM 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'.

3.2 Rebuilding Open MPI

3.2.1 Rebuilding Open MPI Using a Helper Script

The $HPCX_ROOT/utils/hpcx_rebuild.sh script can rebuild OMPI and UCX from HPC-X using the same sources and configuration. It also takes into account HPC-X's environments: vanilla, MT and CUDA.

For details, run:

```
$HPCX_ROOT/utils/hpcx_rebuild.sh --help
```

3.2.2 Rebuilding Open MPI from HPC-X™ Sources

HPC-X package contains Open MPI sources that can be found in `$HPCX_HOME/sources/` folder. Further information can be found in HPC-X README file.

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

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

Open MPI and OpenSHMEM are pre-compiled with UCX v1.5 and HCOLL v4.2, and use them by default.

If HPC-X is intended to be used with SLURM PMIx plugin, Open MPI should be build against external PMIx, Libevent and HWLOC and the same Libevent and PMIx libraries should be used for both SLURM and Open MPI.

Additional configuration options:

```
--with-pmix=<path-to-pmix>
--with-libevent=<path-to-libevent>
--with-hwloc=<path-to-hwloc>
```

3.3 Loading KNEM Module

UCX intra-node communication uses the KNEM module, which improves the performance significantly. Make sure this module is loaded on your system:

```
$ modprobe 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. Moreover, recent KNEM releases enforce by default that the attacker and the target process have the same UID which prevent any security issues.

3.4 Running MPI with FCA v4.x (hcoll)

FCA v4.2 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:

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

3.5 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 3.2, “Rebuilding Open MPI”, 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,openib
--mca btl_openib_cpc_include rdmacm -mca btl_openib_if_include mlx5_0:1 -mca btl_openib_gid_in-
dex 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 is not compiled with ib-router support, therefore it supports RoCE out-of-the-box.

3.6 Direct Launch of Open MPI and OpenSHMEM using SLURM 'srun'

If Open MPI was built with SLURM support, and SLURM has PMI2 or PMIx support, the Open MPI and OpenSHMEM applications can be launched directly using the "srun" command:

- Open MPI:

```
`env <MPI/OSHMEM-application-env> srun --mpi={pmi2|pmix} <srun-args> <mpi-app-args>`
```

All Open MPI/OpenSHMEM parameters that are supported by the `mpirun/oshrun` command line can be provided through environment variables using the following rule:

`"-mca <param_name> <param-val>" => "export OMPI_MCA_<param_name>=<param-val>"`

For example an alternative to `"-mca coll_hcoll_enable 1"` with `'mpirun'` is `"export OMPI_MCA_coll_hcoll_enable=1"` with `'srun'`

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.

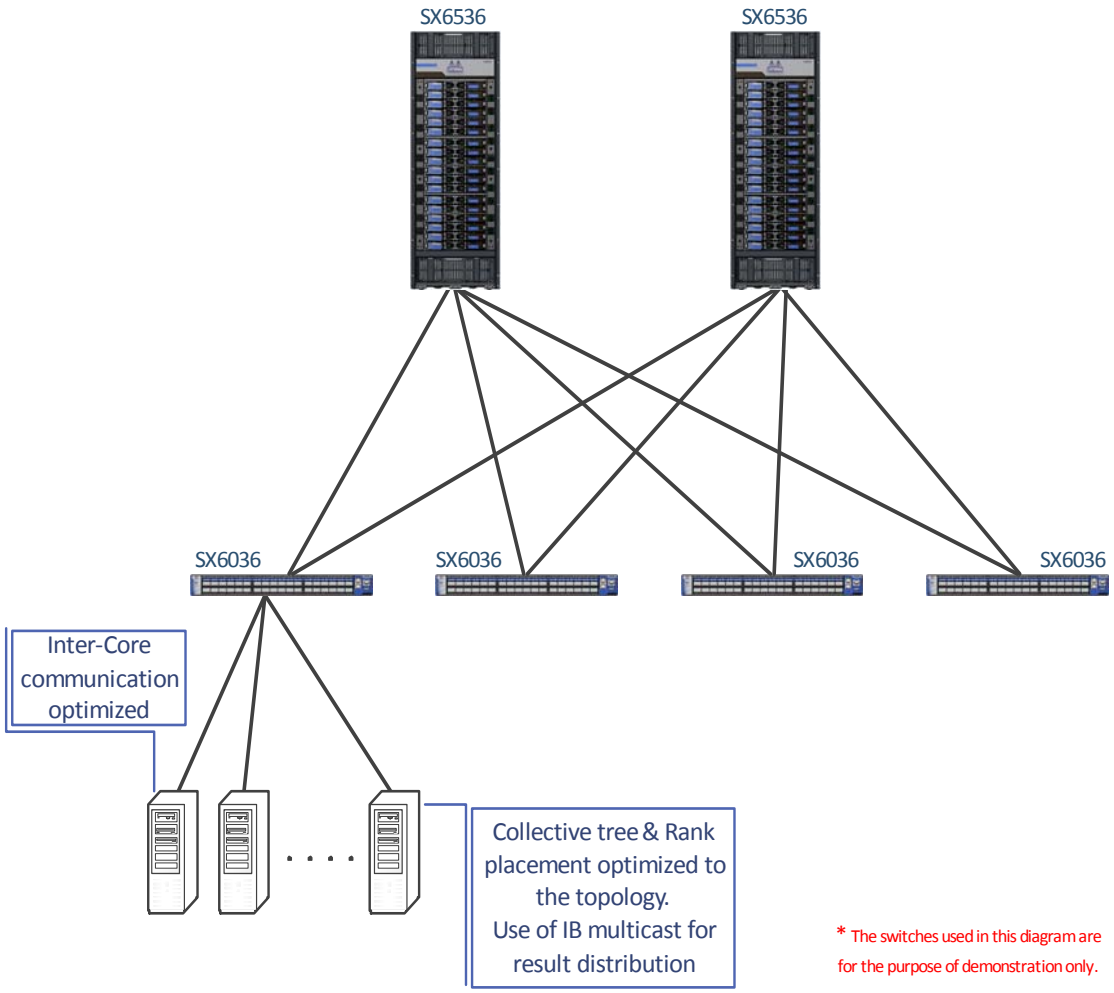
Additionally, FCA v4.2 also contains support for building runtime configurable hierarchical collectives. As with FCA 2.X, FCA v4.2 leverages hardware multicast capabilities to accelerate collective operations. In FCA v4.2, we take full advantage of the performance and scalability of the UCX point-to-point library in the form of the "ucx_p2p" BCOL. This enables users to leverage Mellanox hardware offloads transparently and with minimal effort.

FCA v4.2 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 v4.2 release currently supports blocking and non-blocking variants of "Allgather", "Allgatherv", "Allreduce", "AlltoAll", "AlltoAllv", "Barrier", and "Bcast".

As of HPC-X v2.2, FCA (v4.1), which is part of the HPC-X package, will not be compiled with MXM. FCA will be compiled with UCX and will use it by default.

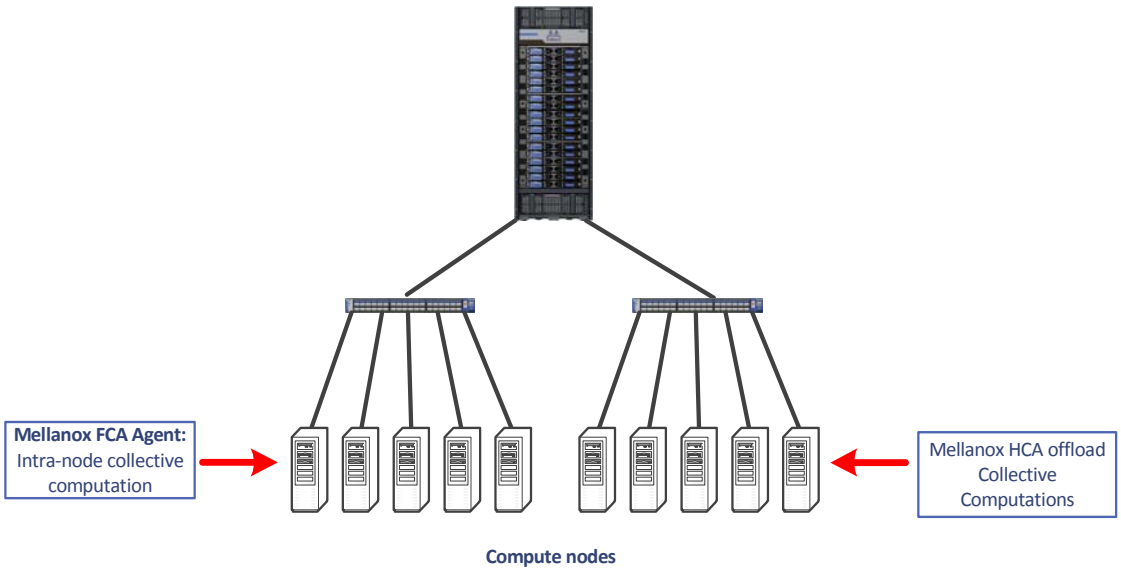
The following diagram summarizes the FCA architecture:

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 require 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 v3.x and FCA v4.2

FCA v4.2 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 v4.2 (hcoll) does not require FCA manager daemon.

4.4 Configuring FCA

4.4.1 Compiling Open MPI with FCA v4.2

➢ ***To compile Open MPI with FCA v4.2***

Step 1. Install FCA v4.2 from:

- an RPM.

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

- a tarball.

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

FCA v4.2will 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 UCX v1.3 or higher.

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

```
% rpm -qi hcoll
```

➢ ***To upgrade to a newer version of FCA:***

Step 1. Remove the existing FCA version.

```
% rpm -e hcoll
```

Step 2. Remove the precompiled Open MPI.

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

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

4.4.2 Enabling FCA in Open MPI

To enable FCA v4.2 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> ./a.out
```

4.4.3 Tuning FCA v4.2 Setting

The default FCA v4.2 settings should be optimal for most systems. To check the available FCA parameters and their default values, run the following command:

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

FCA v4.2 parameters are simply environment variables and can be modified in one of the following ways:

- Modify the default FCA v4.2 parameters as part of the `mpirun` command:

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

- Modify the default FCA v4.2 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 v4.2 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_Iallreduce (4b, 8b, SUM, MIN, PROD, AND, OR, LAND, LOR)

4.4.6 Enabling Multicast Accelerated Collectives

FCA v4.2, 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 Mellanox SHARP Software Accelerated Collectives

As of v1.7, HPC-X supports Mellanox SHARP Software Accelerated Collectives. These collectives are enabled by default if FCA (HCOLL) v3.5 and above detects that it is running in a supporting environment.

- ***To enable Mellanox SHARP acceleration:***

```
-x HCOLL_ENABLE_SHARP=1
```

- ***To disable Mellanox SHARP acceleration:***

```
-x HCOLL_ENABLE_SHARP=0
```

- ***To change Mellanox 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 Mellanox SHARP non-blocking interface:***

```
-x HCOLL_ENABLE_SHARP_NONBLOCKING=1
```

For instructions on how to deploy Mellanox SHARP software in InfiniBand fabric, see Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) Deployment Guide.

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

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

4.4.8 Configuring NVIDIA® CUDA® GPUs Support - HCOLL

Collective operations with CUDA memory is enabled in HCOLL using NVIDIA's NCCL collective communication library. HCOLL CUDA support is enabled in HPC-X through HCOLL CUDA build. LD_PRELOAD CUDA-enabled libhcoll.so library to enable collective operation with CUDA buffers.

- ***To select an HCOLL CUDA topology:***

```
-x HCOLL_CUDA_SBGP=p2p  -x HCOLL_CUDA_BCOL=nccl
```

- ***To tune maximum message size threshold to HCOLL staging scheme with CUDA buffers:***

```
-x HCOLL_CUDA_STAGING_MAX_THRESHOLD=262144
```

For further information on CUDA support in HPC-X, please refer to section Section 5.4.5, “CUDA GPU”, on page 33.

4.4.9 Limitations

- HCOLL, as of v4.1 release, does not fully support mixed MPI datatypes.

  In this context, mixed datatypes refers to collective operations where the datatype layout of input and output buffers may be different on different ranks. For example:

  For an arbitrary MPI collective operation:

```
MPI_Collective_op( input, count1, datatype-in_i, output, count2,datatype-out_i,
communicator)
```

  Where i = 0,...,(number_of_mpi_processes - 1)

Mixed mode means when i is not equal to j, (datatype-in_i, datatype-out_i) is not necessarily equal to (datatype-in_j, datatype-out_j).

Mixed MPI datatypes, in general, can prevent protocol consensus inside HCOLL, resulting in hangs. However, because HCOLL contains a datatype engine with packing and unpacking flows built into the collective algorithms, mixed MPI datatypes will work under the following scenarios:

- If the packed length of the data (a value all ranks must agree upon regardless of datatype) can fit inside a single HCOLL buffer (the default is (64Kbytes – header_space)), then mixed datatypes will work.
- If the packed length of count*datatype is bigger than an internal HCOLL buffer, then HCOLL will need to fragment the message. If the datatypes and counts are defined on each rank so that all ranks agree on the number of fragments needed to complete the operation, then mixed datatypes will work. Our datatype engine cannot split across primitive types and padding, this may result in non-agreement on the number of fragments required to process the operation. When this happens, HCOLL will hang with some ranks expecting incoming fragments and other believing the operation is complete.

- The environment variable HCOLL_ALLREDUCE_ZCOPY_TUNE=<static/dynamic> (default - dynamic) selects the level of automatic runtime tuning of HCOLL's large data allreduce algorithm. "Static" means no tuning is applied at runtime. "Dynamic" - allows HCOLL to dynamically adjust the algorithms radix and zero-copy threshold selection based on runtime sampling of performance.

  Note: The "dynamic" mode should not be used in cases where numerical reproducibility is required, as this mode may result in a variation of the floating point reduction result from one run to another due to non-fixed reduction order.

5 Unified Communication - X Framework Library

5.1 Overview

Unified Communication - X Framework (UCX) is a new acceleration library, integrated into the Open MPI (as a pml layer) and to OpenSHMEM (as an spml 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®/ConnectX®-4 and above 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 and http://www.openucx.org/

5.1.1 Supported CPU Architectures

Unified Communication - X Framework (UCX) supported CPU architectures are: x86, ARM, PowerPC.

5.2 Configuring UCX

As of HPC-X v2.1, UCX is set as the default pml for Open MPI, default spml for OpenSHMEM.

5.2.1 Using UCX with OpenMPI

UCX is the default pml in Open MPI and the default spml in OpenSHMEM.

- ***To use UCX with Open MPI explicitly:***

```
$mpirun --mca pml ucx -mca osc ucx …
```

- ***To use UCX with OpenSHMEM explicitly:***

```
$oshrun --mca spml ucx …
```

5.2.2 Configuring UCX with XPMEM

By default, UCX library embedded within HPC-X is compiled without XPMEM support.
In order to compile UCX with XPMEM support, follow the steps below:

1. Make sure your host has XPMEM headers and the userspace library is installed.
2. Untar the UCX sources available inside the $HPCX_HOME/sources directory, and recompile UCX:

```
% ./autogen.sh
% ./contrib/configure-release --with-xpmem=/path/to/xpmem --prefix=/path/to/new/ucx/install
% make -j8 install
```

Note: In case the new UCX version is installed in a different location, use LD_PRELOAD for Open MPI to use the new location:

```
% mpirun -mca pml ucx -x LD_PRELOAD=/path/to/new/ucx/install ...
```

When UCX is compiled from sources, it should be optimized for the best performance.
To accomplish this, please compile UCX with:
`./contrib/configure-release --enable-optimizations`

5.3 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.

- ***To check the UCX version, run:***

```
$HPCX_UCX_DIR/bin/ucx_info -v
```

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.

- Improving performance at scale by increasing the value of number of DC initiator QPs (DCI) used by the interface when using the DC transport:

```
$mpirun -mca pml ucx -x UCX_TLS=sm,dc_x  -x UCX_DC_MLX5_NUM_DCI=16
```

  or

```
$mpirun -mca pml ucx -x UCX_TLS=sm,dc  -x UCX_DC_VERBS_NUM_DCI=16
```

- Running UCX on a RoCE port, by:
  - Configuring the fabric as lossless (see RoCE Deployment Community post), and setting UCX_IB_TRAFFIC_CLASS=106.

    OR
  - Setting the specific port using the UCX_NET_DEVICES environment variable. For example:

```
$mpirun -mca pml ucx -x UCX_NET_DEVICES=mlx5_0:2
```

- By default, RoCE v2 and IPv4 are used, if available. Otherwise, RoCE v1 with MAC address is used. In order to set a specific RoCE version to use, set UCX_IB_GID_INDEX to the index of the required RoCE version and address type, as reported by "show_gids" command. For example:

```
$mpirun -x UCX_NET_DEVICES=mlx5_0:2 -x UCX_TRAFFIC_CLASS=106 -x UCX_IB_GID_INDEX=3 <application command line>
```

  Note: RoCE and RoCEv2 are not supported with dc and dc_x transports in UCX.

- Setting the threshold for using the Rendezvous protocol in UCX:

```
$mpirun -mca pml ucx -x  UCX_RNDV_THRESH=16384
```

  By default, UCX will calculate the optimal threshold on its own, but the value can be overwritten using the above environment parameter.

- Setting the threshold for using the zero-copy in UCX:

```
$mpirun -mca pml ucx -x  UCX_ZCOPY_THRESH=16384
```

  By default, UCX will calculate the optimal threshold on its own, but the value can be overwritten using the above environment parameter.

5.4 UCX Features

5.4.1 Hardware Tag Matching

Starting ConnectX-5, Tag Matching previously done by the software, can now be offloaded in UCX to the HCA. For MPI applications, sending messages with numeric tags accelerates the processing of incoming messages, leading to better CPU utilization and lower latency for expected messages. In Tag Matching, the software holds a list of matching entries called matching list. Each matching entry contains a tag and a pointer to an application buffer. The matching list is used to steer arriving messages to a specific buffer according to the message tag. The action of traversing the matching list and finding the matching entry is called Tag Matching, and it is performed on the HCA instead of the CPU. This is useful for cases where incoming messages are consumed not in the order they arrive, but rather based on numeric identifier coordinated with the sender.

Hardware Tag Matching avails the CPU for other application needs. Currently, hardware Tag Matching is supported for the regular and accelerated RC and DC transports (RC, RC_X, DC, DC_X), and can be enabled in UCX with the following environment parameters:

- For the RC transports:

```
UCX_RC_TM_ENABLE=y
```

- For the DC transports:

```
UCX_DC_TM_ENABLE=y
```

By default, only messages larger than a certain threshold are offloaded to the transport. This threshold is managed by the “UCX_TM_THRESH” environment variable (its default value is 1024 bytes).

UCX may also use bounce buffers for hardware Tag Matching, offloading internal pre-registered buffers instead of user buffers up to a certain threshold. This threshold is controlled by the UCX_TM_MAX_BCOPY environment variable. The value of this variable has to be equal or less than the segment size, and it must be larger than the value of UCX_TM_THRESH to take effect (1024 bytes is the default value, meaning that optimization is disabled by default).

With hardware Tag Matching enabled, the Rendezvous threshold is limited by the segment size. Thus, the real Rendezvous threshold is the minimum value between the segment size and the value of UCX_RNDV_THRESH environment variable.

Hardware Tag Matching for InfiniBand requires MLNX_OFED v4.1-x.x.x.x and above.

Hardware Tag Matching for RoCE is not supported.

For further information, refer to https://community.mellanox.com/docs/DOC-2781.

5.4.2 Single Root IO Virtualization (SR-IOV)

SR-IOV is a technology that allows a physical PCIe device to present itself multiple times through the PCIe bus. This technology enables multiple virtual instances of the device with separate resources. These virtual functions can then be provisioned separately. Each VF can be seen as an additional device connected to the Physical Function. It shares the same resources with the Physical Function, and its number of ports equals those of the Physical Function.

SR-IOV is commonly used in conjunction with an SR-IOV enabled hypervisor to provide virtual machines direct hardware access to network resources hence increasing its performance.

To enable SR-IOV in UCX while it is configured in the fabric, use the following environment parameter:

```
UCX_IB_ADDR_TYPE=ib_global
```

Notes:

- This environment parameter should also be used when running UCX on a fabric with Socket Direct HCA installed. When working with Socket Direct HCAs, make sure Multi-Rail feature is enabled as well (refer to Section 5.4.6, “Multi-Rail”, on page 35).
- SRI-OV is not supported with dc and dc_x transports in UCX.

5.4.3 Adaptive Routing

This feature is supported on ConnectX-5 HCAs and above only.

Adaptive Routing (AR) enables sending messages between two HCAs on different routes, based on the network load. While in static routing, a packet that arrives to the switch is forwarded based on its destination only, in Adaptive Routing, the packet is loaded to all possible ports that the packet can be forwarded to, resulting in the load being balanced between ports, and the fabric adapting to load changes over time. This feature requires support for out-of-order arrival of messages, which UCX has for the RC, rc_x and DC, dc_x transports.

To enable AR in MLNX_OFED v4.1-x.x.x.x and above, set the following environmental parameters, according to the used transport:

```
UCX_RC_VERBS_OOO_RW=y
UCX_DC_VERBS_OOO_RW=y
UCX_RC_MLX5_OOO_RW=y
UCX_DC_MLX5_OOO_RW=y
```

To be able to use Adaptive Routing on the fabric, make sure it is enabled in OpenSM and in the switches.

5.4.4 Error Handling

Error Handling enables UCX to handle errors that occur due to algorithms with fault recovery logic. To handle such errors, a new mode was added, guaranteeing an accurate status on every sent message. In addition, the process classifies errors by their origin (i.e. local or remote) and severity, thus allowing the user to decide how to proceed and what would that possibly recovery method be. To use Error Handling in UCX, the user must register with the UCP API (the ucp_ep_create API function needs to be addressed, for example).

5.4.5 CUDA GPU

5.4.5.1 Overview

CUDA environment support in HPC-X enables the use of NVIDIA's GPU memory in UCX and HCOLL communication libraries for point-to-point and collective routines, respectively.

5.4.5.2 Supported Architectures

- CPU architecture: x86
- NVIDIA GPU architectures:
  - Tesla
  - Kepler
  - Pascal
  - Volta

5.4.5.3 System Requirements

- CUDA v8.0 or higher - for information on how to install CUDA, refer to NVIDIA documents for CUDA Toolkit.
- In HPC-X Rev 2.3, UCX is compiled with CUDA v9.2.
- Mellanox OFED GPUDirect RDMA plugin module - for information on how to install:
  - Mellanox OFED - refer to MLNX_OFED webpage
  - GPUDirect RDMA - refer to Mellanox OFED GPUDirect RDMA webpage

    Once the NVIDIA software components are installed, it is important to verify that the GPUDirect RDMA kernel module is properly loaded on each of the compute systems where you plan to run the job that requires the GPUDirect RDMA.

    ➢ ***To check whether the GPUDirect RDMA module is loaded, run:***

    ```
    service nv_peer_mem status
    ```

    ➢ ***To run this verification on other Linux flavors:***

    ```
    lsmod | grep nv_peer_mem
    ```

- GDR COPY plugin module - GDR COPY is a fast copy library from NVIDIA, used to transfer between HOST and GPU. For information on how to install GDR COPY, refer to its GitHub webpage

  Once GDR COPY is installed, it is important to verify that the gdrcopy kernel module is properly loaded on each of the compute systems where you plan to run the job that requires the GDR COPY.

  ➢ ***To check whether the GDR COPY module is loaded, run:***

  ```
  lsmod | grep gdrdrv
  ```

- NVIDIA Collective Communication Library (NCCL) - NCCL is a collective library from NVIDIA for multi-GPU collective communication primitives that are topology-aware. For information on how to install NCCL, refer NVIDIA's Deep Learning SDK Documentation

5.4.5.4 Configuring CUDA Support - UCX

CUDA support in UCX depends on the GPUDirect RDMA and GDR COPY services.

Building UCX with CUDA support requires the following configuration flags:

- `--with-cuda=<cuda/runtime/install/path>`
- `--with-gdrcopy=<gdr_copy/install/path>`

Currently, CUDA is not supported in shared memory channels.

To achieve best performance, please add the following to the command line:
`-x UCX_TLS=rc_x,cuda_copy,gdr_copy`

Make sure to use the hpcx-cuda module when running the UCX library that was already compiled with CUDA support (see hpcx-cuda module bullet in Section 2.5, "HPC-X Environments", on page 14).

5.4.6 Multi-Rail

Multi-Rail enables users to use more than one of the active ports on the host, making better use of system resources, and allowing increased throughput.

Each process would be able to use up to the first 3 active ports on the host in parallel (this 3 port limitation is for performance considerations), if the following parameters are set:

- ***For setting the number of active ports to use for the Eager protocol, i.e. for small messages, please set the following parameter:***

```
% mpirun -mca pml ucx -x UCX_MAX_EAGER_RAILS=3 ...
```

- ***For setting the number of active ports to use for the Rendezvous protocol, i.e. for large messages, please set the following parameter:***

```
% mpirun -mca pml ucx -x UCX_MAX_RNDV_RAILS=3 ...
```

- Possible values for these parameters are: 1, 2, and 3. The default value for both is 1.

The Multi-Rail feature will be disabled while the Hardware Tag Matching feature is enabled.

5.4.7 Memory in Chip (MEMIC)

Memory in chip feature allows for using on-device memory for sending messages from the UCX layer. This feature is enabled by default on ConnectX-5 HCAs. It is supported only for the rc_x and dc_x transports in UCX.

The environment parameters that control this feature's behavior are:

- UCX_RC_MLX5_DM_SIZE
- UCX_RC_MLX5_DM_COUNT
- UCX_DC_MLX5_DM_SIZE
- UCX_DC_MLX5_DM_COUNT

For more information on these parameters, please refer to the ucx_info utility: % $HPCX_UCX-_DIR/bin/ucx_info -f.

5.5 UCX Utilities

5.5.1 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.

5.6 Generating UCX Statistics for Open MPI/OpenSHMEM

In order to generate statistics, the statistics destination and trigger should be set, and they can optionally be filtered and/or formatted.

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

Table 4 - UCX_STATS_DEST Environment Variables

Value	Description
`empty string`	Statistics are not reported
`stdout`	Print to standard output
`stderr`	Print to standard error
`file:<filename>`	Save to a file. Following substitutions are made: %h: host, %p: pid, %c: cpu, %t: time, %e: exe
`udp:<host>[:<port>]`	Send over UDP to the given host:port

Example:

```
$ export UCX_STATS_DEST="file:ucx_%h_%e_%p.stats"
$ export UCX_STATS_DEST="stdout"
```

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

Table 5 - UCX_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
`signal:<signo>`	Dump when process is signaled

Example:

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

- It is possible to filter the counters in the report using the UCX_STATS_FILTER environment parameter. It accepts a comma-separated list of glob patterns specifying counters to display. Statistics summary will contain only the matching counters. The order is not meaningful. Each expression in the list may contain any of the following options:

Environment Variable	Description
*	Matches any number of any characters including none (prints a full report)
?	Matches any single character
[abc]	Matches one character given in the bracket
[a-z]	Matches one character from the range given in the bracket

  More information about this parameter can be found at: https://github.com/openucx/ucx/wiki/Statistics

- It is possible to control the formatting of the statistics using the UCX_STATS_FORMAT parameter:

Environment Variable	Description
full	Each counter will be displayed in a separate line
agg	Each counter will be displayed in a separate line. However, there will also be an aggregation between similar counters
summary	All counters will be printed in the same line

The statistics feature is only enabled when UCX is compiled with the `enable-stats` flag. This flag is set to 'No' by default. Therefore, in order to use the statistics feature, please recompile UCX using the contrib/configure-prof file, or use the 'debug' version of UCX, which can be found in $HPCX_UCX_DIR/debug:

```
$ mpirun -mca pml ucx -x LD_PRELOAD=$HPCX_UCX_DIR/debug/lib/libucp.so ...
```

Please note that recompiling UCX using the aforementioned methods may impact the performance.

6 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.

6.1 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 UCX (Unified Communication - X), MellanoX Messaging libraries (MXM), as well as Mellanox Fabric Collective Accelerations (FCA), providing an unprecedented level of scalability for SHMEM programs running over InfiniBand.

6.2 Running HPC-X OpenSHMEM

6.2.1 Running HPC-X OpenSHMEM with UCX

Unified Communication - X Framework (UCX) is a new acceleration library, integrated into the Open MPI (as a pml layer) and to OpenSHMEM (as an spml 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®/ConnectX®-4 and above 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 and http://www.openucx.org/

6.2.1.1 Enabling UCX for HPC-X OpenSHMEM Jobs

UCX is the default spml starting from HPC-X v2.1. For older versions of HPC-X, add the following MCA parameter to the oshrun command line:

```
-mca spml ucx
```

All the UCX environment parameters can be used in the same way with oshrun, as well as with mpirun. For the complete list of the UCX environment parameters, please run:

```
$HPCX_UCX_DIR/bin/ucx_info -f
```

6.2.2 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;
}
```

6.2.3 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
```

It is required to drop_caches on all test machines before running OpenSHMEM application and/or benchamarks in order to free memory:

```
echo 3 > /proc/sys/vm/drop_caches
```

6.2.3.1 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 --map-by
node -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
- ucx - used to allocate and register memory via the UCX library

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

6.2.3.2 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
```

6.3 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.

6.3.1 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>
```