



Connect. Accelerate. Outperform.™

Mellanox HPC-X™ Software Toolkit User Manual

Rev 1.2

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 NON-INFRINGEMENT 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

Mellanox Technologies, Ltd.
 Beit Mellanox
 PO Box 586 Yokneam 20692
 Israel
www.mellanox.com
 Tel: +972 (0)74 723 7200
 Fax: +972 (0)4 959 3245

© Copyright 2014. Mellanox Technologies. All Rights Reserved.

Mellanox®, Mellanox logo, BridgeX®, ConnectX®, Connect-IB®, CORE-Direct®, InfiniBridge®, InfiniHost®, InfiniScale®, MetroX®, MLNX-OS®, PhyX®, ScalableHPC®, SwitchX®, UFM®, Virtual Protocol Interconnect® and Voltaire® are registered trademarks of Mellanox Technologies, Ltd.

ExtendX™, FabricIT™, Mellanox Open Ethernet™, Mellanox Virtual Modular Switch™, MetroDX™, TestX™, Unbreakable-Link™ are trademarks of Mellanox Technologies, Ltd.

All other trademarks are property of their respective owners.

Table of Contents

Table of Contents	3
List of Tables	6
List of Figures	7
Document Revision History	8
About This Manual	9
Scope	9
Intended Audience	9
Syntax Conventions	9
Chapter 1 HPC-X™ Software Toolkit Overview	10
1.1 HPC-X Package Contents	10
1.2 HPC-X™ Requirements	10
Chapter 2 Installing and Loading HPC-X™	11
2.1 Installing HPC-X	11
2.2 Loading HPC-X Environment from bash	11
2.3 Loading HPC-X Environment from Modules	11
Chapter 3 Running, Configuring and Rebuilding HPC-X™	12
3.1 Starting FCA Manager from HPC-X	12
3.2 Profiling IB verbs API	12
3.3 Profiling MPI API	12
3.4 Rebuilding Open MPI from HPC-X™ Sources	13
3.5 Running MPI with MXM	13
3.6 Generating MXM Statistics for Open MPI/OpenSHMEM	14
3.7 Generating MXM Environment Parameters	15
3.8 Loading KNEM Module	25
3.9 Running MPI with FCA v2.5	25
3.10 Running OpenSHMEM with FCA v2.5	25
3.11 Running MPI with FCA v3.1 (hcoll)	25
Chapter 4 Mellanox Fabric Collective Accelerator (FCA)	27
4.1 Overview	27
4.2 FCA Installation Package Content	29
4.3 Differences Between FCA v2.5 and FCA v3.X	30
4.4 Configuring FCA	30
4.4.1 Compiling Open MPI with FCA 3.X	30
4.4.2 Enabling FCA in Open MPI	30
4.4.3 Tuning FCA 3.X Setting	31
4.4.4 Selecting Ports and Devices	31
4.5 Runtime Configuration of FCA	31
4.5.1 Memory Hierarchy	31
4.5.1.1 Available SBGPs	31
4.5.1.2 Available BCOLs	31

4.5.1.3	Supported Collectives	32
4.5.1.4	Different Memory Hierarchy Usages	32
4.5.2	Enabling Mellanox Specific Features and Optimizations	32
4.5.3	Selecting Shared Memory and MXM Point-To-Point Hierarchies for Collectives in FCA 33	
4.6	FCA 3.1 Integration	33
Chapter 5	MellanoX Messaging Library	34
5.1	Overview	34
5.2	Compiling Open MPI with MXM	34
5.3	Enabling MXM in Open MPI	35
5.4	Tuning MXM Settings	36
5.5	Configuring Multi-Rail Support	36
5.6	Configuring MXM over the Ethernet Fabric	37
5.7	Configuring MXM over Different Transports	37
5.8	Configuring Service Level Support	37
5.9	Running Open MPI with pml “yalla”	38
5.10	Adaptive Routing for UD Transport	39
5.11	Support for a Non-Base LID	39
5.12	MXM Performance Tuning	39
5.13	MXM Utilities	40
5.13.1	mxm_dump_config	40
5.13.2	mxm_perftest	40
Chapter 6	PGAS Shared Memory Access Overview	43
6.1	HPC-X OpenSHMEM	43
6.2	Running HPC-X OpenSHMEM	44
6.2.1	Running HPC-X OpenSHMEM with MXM	44
6.2.1.1	Enabling MXM for HPC-X OpenSHMEM Jobs	44
6.2.1.2	Working with Multiple HCAs	44
6.2.2	Running HPC-X™ OpenSHMEM with FCA	45
6.2.3	Developing Application using HPC-X OpenSHMEM together with MPI	45
6.2.4	HPC-X™ OpenSHMEM Tunable Parameters	46
6.2.4.1	OpenSHMEM MCA Parameters for Symmetric Heap Allocation	47
6.2.4.2	Parameters Used to Force Connection Creation	47
6.3	Performance Optimization	48
6.3.1	Configuring Hugepages	48
6.4	Tuning OpenSHMEM Atomics Performance	49
6.5	Tuning MTU Size to the Recommended Value	49
6.5.1	HPC Applications on Intel Sandy Bridge Machines	50
Chapter 7	Unified Parallel C Overview	51
7.1	HPC-X HPC-X HPC-X HPC-X Compiling and Running the UPC Application	51
7.1.1	Compiling the UPC Application	51
7.1.2	Running the UPC Application	51
7.1.2.1	Basic upcrun Options	52
7.1.2.2	Environment Variables	52
7.2	FCA Runtime Parameters	52

7.2.1	Enabling FCA Operations through Environment Variables in HPC-X UPC	53
7.2.2	Controlling FCA Offload in HPC-X UPC using Environment Variables	53
7.3	Various Executable Examples	53

List of Tables

Table 1:	Syntax Conventions	9
Table 2:	Generating MXM Environment Parameters	15
Table 3:	Available SBGPs	31
Table 4:	Available BCOLs	31
Table 5:	Supported Collectives	32
Table 6:	MLNX_OFED and MXM Versions	35
Table 7:	Runtime Parameters	52

List of Figures

Figure 1: FCA Architecture	28
Figure 2: FCA Components	29

Document Revision History

Revision	Date	Description
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 HPC-X package contains the following pre-compiled HPC packages:

Components	Description
MPI	<ul style="list-style-type: none"> Open MPI and OpenSHMEM v1.8 (MPI-3 complaint, OpenSHMEM v1.0 compliant). Open MPI and OpenSHMEM are available at: http://www.open-mpi.org/software/ompi/v1.8/ MPI profiler (IPM - open source tool from http://ipm-hpc.org/) MPI tests (OSU, IMB, random ring, etc.)
HPC Acceleration Package	<ul style="list-style-type: none"> MXM 3.1 MXM 3.2 (default) FCA v2.5 (default) FCA v3.1 (code name: "hcoll") knem (High-Performance Intra-Node MPI Communication module from: http://runtime.bordeaux.inria.fr/knem/)
Extra packages	<ul style="list-style-type: none"> Berkeley UPC v2.18.0 libibprof (IB verbs profiler)

1.2 HPC-X™ Requirements

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

Platform	Drivers and HCAs
OFED / MLNX_OFED	<ul style="list-style-type: none"> OFED 1.5.3 and later MLNX_OFED 1.5.3-x.x.x, 2.0-x.x.x and later
HCAs	<ul style="list-style-type: none"> Mellanox ConnectX®-2 / ConnectX®-3 / ConnectX®-3 Pro Mellanox 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 v1.8. 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 v1.8).

➤ *To load Open MPI/OpenSHMEM v1.8 based package:*

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

2.3 Loading HPC-X Environment from Modules

➤ *To load Open MPI/OpenSHMEM v1.8 based package:*

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

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=$HPCX_HOME_IBPROF_DIR/lib/libibprof.so:$HPCX_HOME_MXM_DIR/lib/lib-
mxm.so:$HPCX_HOME_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_HOME_IPM_DIR/etc/ipm_key_mpi
$ export IPM_LOG=FULL
$ export LD_PRELOAD=$HPCX_HOME_IPM_DIR/lib/libipm.so
$ mpirun -x LD_PRELOAD <...>
$ $HPCX_HOME_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=${HPCX_HOME}/hpcx-mpi --with-knem=${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
```

3.5 Running MPI with MXM

Open MPI and OpenSHMEM are pre-compiled with MXM v3.2 and use it by default.

➤ *To run with MXM v3.1:*

```
$ LD_PRELOAD=$HPCX_DIR/mxm-v3.1/lib/libmxm.so mpirun -x LD_PRELOAD <...>
$ LD_PRELOAD=$HPCX_DIR/mxm-v3.1/lib/libmxm.so oshrun -x LD_PRELOAD <...>
```

For further details on running MPI with MXM, please refer to:

`$HPCX_HOME/mxm/README.txt`

3.6 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:

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 <code>mxm_stats_parser</code> 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:

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 ...
```

3.7 Generating MXM Environment Parameters

Table 2 - Generating MXM Environment Parameters

Variable	Valid Values	Description
MXM_LOG_LEVEL	<ul style="list-style-type: none"> • 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	<p>If not empty, MXM will print log messages to the specified file instead of stdout.</p> <p>The following substitutions are performed on this string:</p> <ul style="list-style-type: none"> • %p - Replaced with process ID • %h - Replaced with host name <p>Value: String.</p>
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	<ul style="list-style-type: none"> • None: No error handling • Freeze: Freeze and wait for a debugger • Debug: attach debugger • bt: print backtrace 	Error handling mode.
MXM_ERROR_SIGNALS	<ul style="list-style-type: none"> • 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)
MXM_GDB_COMMAND	gdb	If non-empty, attaches a gdb to the process in case of error, using the provided command. Value: string

Variable	Valid Values	Description
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	<ul style="list-style-type: none"> udp:<host>[:<port>] - send over UDP to the given host:port. stdout: print to standard output. stderr: print to standard error. file:<file-name>[: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	<ul style="list-style-type: none"> timer:<interval>: dump in specified intervals. exit: dump just before program exits (default) 	Trigger to dump statistics Value: string
MXM_MEMTRACK_DEST	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> %h: host %p: pid %c: cpu %t: time %e: exe. 	File name to dump instrumentation records to. Value: string

Variable	Valid Values	Description
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	<ul style="list-style-type: none"> Signal none Thread (default) 	Asynchronous progress method. Value: [none signal thread]
MXM_MEM_ALLOC	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> y - single thread n - not single thread 	Mode of the thread usage. Value: <y n>

Variable	Valid Values	Description
MXM_SHM_KCOPY_MODE	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> n y 	Allow using resize_cq(). Value: <y n>
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

Variable	Valid Values	Description
MXM_IB_MAP_MODE	<ul style="list-style-type: none"> • 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 <code>FIRST_SL..(FIRST_SL+NUM_SLS-1)</code> , and use it for outbound communication. applicable values are 1 through 16. Value: unsigned
MXM_IB_WC_MODE	<ul style="list-style-type: none"> • 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 InfiniBand 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	0: (default) $0 \leq \text{value} < 2^{(\text{LMC})} - 1$	InfiniBand path in bits which will be the low portion of the LID, according to the LMC in the fabric. Value: unsigned
MXM_IB_FIRST_SL	0-15	The first Infiniband Service Level number 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: <number>[s us ms ns]

Variable	Valid Values	Description
MXM_UD_FAST_ACK_TIMEOUT	1024.00us	Timeout for getting an acknowledgment for sent packet. Value: time value: <number>[s us ms ns]
MXM_FAST_TIMER_RESOLUTION	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	<ul style="list-style-type: none"> • 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_LENGTH	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
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

Variable	Valid Values	Description
MXM_UD_RX_OOO	<ul style="list-style-type: none"> • 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_LENGTH	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: $4.096 \text{ microseconds} * (2^{\text{timeout}})$ (must be ≥ 0 and ≤ 31) Value: unsigned
MXM_CIB_MAX_RDMA_DEST_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]

Variable	Valid Values	Description
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_BUFFS	-1	Maximal number of send buffers for one endpoint. -1 is infinite. Warning: Setting this param with value $\neq -1$ 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
MXM_CIB_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_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

Variable	Valid Values	Description
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 shmем 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
MXM_SHM_READ_RETRY_COUNT	64	Number of retries in case where cannot read from the shmем FIFO (for multi-thread support). Value: unsigned
MXM_SHM_HUGETLB_MODE	<ul style="list-style-type: none"> • 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_THRESHOLD	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]

Variable	Valid Values	Description
MXM_SHM_KNEM_MAX_SIMULTANEOUS	0	Maximum number of simultaneous ongoing knem operations to support in shmем 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]
MXM_SHM_RELEASE_FIFO_FACTOR	0.500	Frequency of resource releasing on the receiver's side in shmем 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	<ul style="list-style-type: none"> • 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_THRESHOLD	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]

Variable	Valid Values	Description
MXM_SHM_USE_KNEM_DMA	<ul style="list-style-type: none"> n y 	Whether or not to offload to the DMA engine when using KNEM Value: <y n>

3.8 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 poses 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.

3.9 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 12

3.10 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 12

3.11 Running MPI with FCA v3.1 (hcoll)

- Running with default FCA configuration parameters:

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

- Running with NUMA aware CORE-Direct offloaded collectives:

```
$ mpirun -mca coll_hcoll_enable 1 -x HCOLL_BCOL=basesmuma,iboffload,mlnx_p2p -x  
HCOLL_SBGp=basesmuma,ibnet,p2p <...>
```

- Running with NUMA aware, Multicast accelerated collectives:

```
$ mpirun -mca coll_hcoll_enable 1 -x HCOLL_BCOL=basesmuma,mlnx_p2p -x HCOLL_SBGp=bases-  
muma,p2p -x HCOLL_MCAST_ENABLE_ALL=1 HCOLL_MCAST_LOG_LEVEL=0<...>
```

- Running with NUMA aware, network agnostic, logical layer (in the context of using FCA v3.x in Open MPI, the logical layer employs the 'PML' layer for transport services.):

```
$ mpirun -mca coll_hcoll_enable 1 -x HCOLL_BCOL=basesmuma,ptpcoll -x HCOLL_SBGp=bases-  
muma,p2p <...>
```

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.

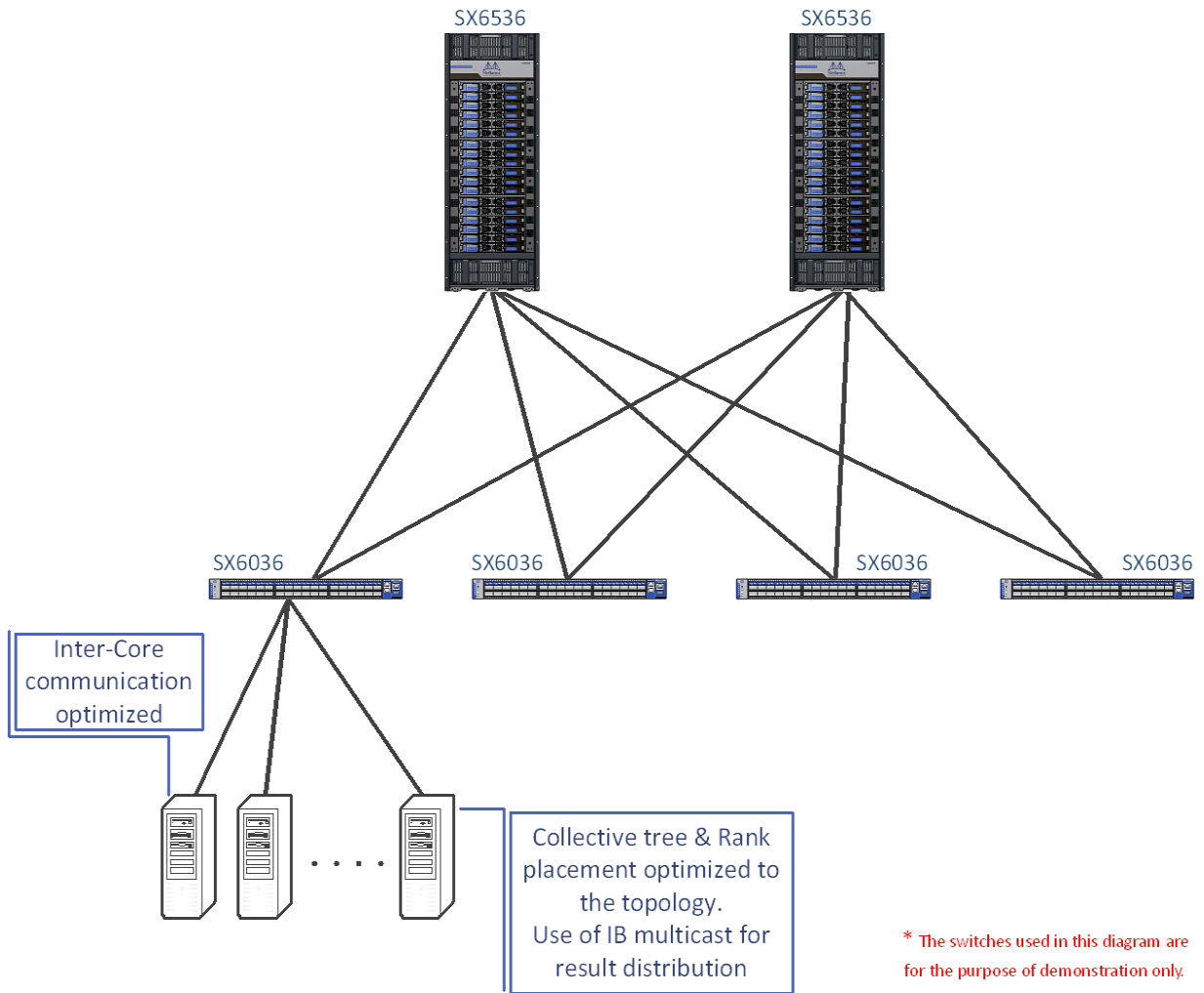
Users may benefit immediately from CORE-Direct® out-of-the-box by simply specifying the necessary BCOL/SBGP combinations. In order to take maximum advantage of CORE-Direct®, users may modify their applications to use MPI 3.1 non-blocking routines while using CORE-Direct® to offload the collective "under-the-covers", thereby allowing maximum opportunity to overlap communication with computation.

Additionally, FCA 3.1 also contains support to build runtime configurable hierarchical collectives. We currently support socket and UMA level discovery with network topology slated for future versions. As with FCA 2.X we also provide the ability to accelerate collectives with hardware multicast. In FCA 3.1 we also expose the performance and scalability of Mellanox's advanced point-to-point library, MXM 2.x, in the form of the "mlnx_p2p" BCOL. This allows users to take full advantage of new features with minimal effort.

FCA 3.1 and above is a standalone library that can be integrated into any MPI or PGAS runtime. Support for FCA 3.1 is currently integrated into Open MPI versions 1.7.4 and higher. The 3.1 release currently supports blocking and non-blocking variants of "Allgather", "Allreduce", "Barrier", and "Bcast".

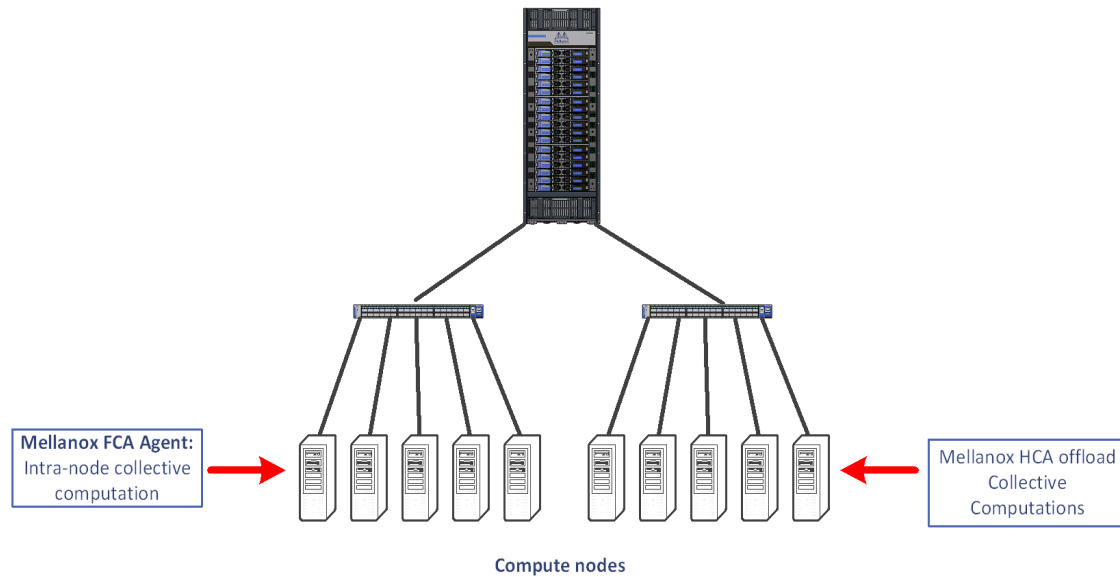
The following diagram summarizes the FCA architecture:

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 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 listed in Prerequisites (on page 11).
- 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 3.1*

Step 1. Install FCA 3.X from:

- an RPM.
- a tarball.

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

```
% 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 required 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 --display-map --bind-to-core -mca coll hcoll,tuned,libnbc,basic -mca
btl_openib_if_include mlx4_0:1 -mca coll_hcoll_np 0 -x HCOLL_MAIN_IB=mlx4_0:1 -x
HCOLL_BCOL=basesmuma,mlnx_p2p -x HCOLL_SBGp=basesmuma,p2p ./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>
```

4.5 Runtime Configuration of FCA

4.5.1 Memory Hierarchy

FCA 3.X is flexible and modular, providing the user a wide degree of latitude to customize collective algorithms to take full advantage of their Mellanox hardware at application runtime.

The FCA 3.X software model abstracts the notion of a memory hierarchy into sub-grouping or SBGP components. An SBGP group is a subset of endpoints that satisfy a reachability criterion, for example, all processes on the same socket. To each SBGP is associated a set of optimized collective primitives, basic collectives or BCOL components.

4.5.1.1 Available SBGPs

Table 3 - Available SBGPs

SBGPs	Description
basesmuma	A subset of ranks that share the same host.
basesmsocket	A subset of ranks that share the same socket.
ibnet	A subset of ranks that can communicate with CORE-Direct®.
p2p	A subset of ranks that can reach each other over point-to-point.

4.5.1.2 Available BCOLs

Table 4 - Available BCOLs

BCOLs	Description
basesmuma	Shared memory collective primitives.

Table 4 - Available BCOLs

BCOLs	Description
mlnx_p2p	MXM based point-to-point collective primitives.
iboffload	CORE-Direct® based collective primitives.
ptpcoll	Point-to-point logical layer.

4.5.1.3 Supported Collectives

Table 5 - Supported Collectives

Collectives	Description
Allgather/Iallgather	Blocking and non-blocking allgather for all possible bcol/sbgp combinations
Allreduce/Iallreduce	Blocking and non-blocking allreduce. Note: Currently not supported with iboffload BCOL
Barrier/Ibarrier	Blocking and non-blocking barrier for all possible BCOL/SBGp combinations.
Bcast/Ibcast	Blocking and non-blocking bcast for all possible BCOL/SBGp combinations.

4.5.1.4 Different Memory Hierarchy Usages

- Two-level hierarchy with CORE-Direct® used at the "top" level:

```
% mpirun -x HCOLL_BCOL=basesmuma,iboffload,mlnx_p2p -x HCOLL_SBGp=basesmuma,ibnet,p2p
```

- Three-level hierarchy with CORE-Direct® used at the "top" level:

```
% mpirun -x HCOLL_BCOL=basesmuma,basesmuma,iboffload,mlnx_p2p -x HCOLL_SBGp=basesmsocket,basesmuma,ibnet,p2p
```

- Two-level hierarchy with MXM p2p used at the "top" level:

```
% mpirun -x HCOLL_BCOL=basesmuma,mlnx_p2p -x HCOLL_SBGp=basesmuma,p2p
```

- Three-level hierarchy with MXM used at the "top" level:

```
% mpirun -x HCOLL_BCOL=basesmuma,basesmuma,mlnx_p2p -x HCOLL_SBGp=basesmsocket,basesmuma,p2p
```

4.5.2 Enabling Mellanox Specific Features and Optimizations

- Multicast acceleration:

FCA 3.1 uses hardware multicast to accelerate collective primitives in both the "mlnx_p2p" and "iboffload" BCOLs when possible.

➤ **To enable multicast based collectives, set:**

```
-x HCOLL_MCAST_ENABLE_ALL=1
```

- Context caching:

When using one of the two Mellanox specific BCOLs (mlnx_p2p, or iboffload), you may enable context caching. This optimization can benefit applications that create and destroy many MPI communicators.

➤ *To enable context caching in conjunction with a valid BCOL/SBGP pair, set:*

```
-x HCOLL_CONTEXT_CACHE_ENABLE=1
```

4.5.3 Selecting Shared Memory and MXM Point-To-Point Hierarchies for Collectives in FCA

Running IMB benchmark on 1,024 MPI processes with two levels of hierarchy:

- shared memory
- MXM point-to-point

Enable both context caching and multicast acceleration.

```
% mpirun -np 1024 --bind-to-core -bynode -mca btl_openib_if_include mlx4_0:1 -mca coll
hcoll,tuned,libnbc -mca btl sm,openib,self HCOLL_MCAST_ENABLE_ALL=1 -x
HCOLL_ENABLE_CONTEXT_CACHE=1
-x HCOLL_IB_IF_INCLUDE=mlx4_0:1 -x HCOLL_BCOL=basesmuma,mlnx_p2p
-x HCOLL_SBGP=basesmuma,p2p ~/IMB/src/IMB-MPI1 -exclude PingPong PingPing Sendrecv
```

4.6 FCA 3.1 Integration

In principle, FCA 3.1 can be integrated into any communication library. In order to do so, one must first implement the so-called "RTE interface", which is a collection of callbacks and handles that must be supplied to FCA 3.x from the calling library. For an example of full integration into an MPI library, please refer to the Open MPI source code under `ompi_src/ompi/mca/coll/hcoll`.

The "hcoll" component contained in the OMPI "coll" framework is the runtime integration layer of FCA into OMPI. A complete implementation of the RTE can be found at `ompi_src/ompi/mca/coll/hcoll`. A standalone example can be found at `/opt/mellanox/hcoll/sdk`.

For instructions on compiling and running, please refer to the SDK's README. The RTE implementation can be found in the "hcoll_sdk.c" file.

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 <...>
```



When upgrading to MXM v3.2, Open MPI compiled with the previous versions of the MXM should be recompiled with MXM v3.2.

5.3 Enabling MXM in Open MPI

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 ...>
```

From Open MPI v1.8, MXM is selected when the number of processes is higher or equal to 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 `/opt/mellanox/mxm/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 in MXM v3.2 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
```

5.7 Configuring MXM over Different Transports

MXM v3.2 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`

5.8 Configuring Service Level Support



Service Level Support is currently at alpha level.

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.

5.9 Running Open MPI with pml “yalla”



Open MPI pml 'yalla' is currently at beta level.

A new pml layer (pml “yalla”) was added to Mellanox's Open MPI v1.8. It 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 without pml 'yalla'. Therefore, to directly use MXM with it, perform the steps below.

➤ **To run MXM with the new pml “yalla”:**

Step 1. Download the HPC-X package from the Mellanox site (Mellanox's Open MPI v1.8 has been integrated into HPC-X package).

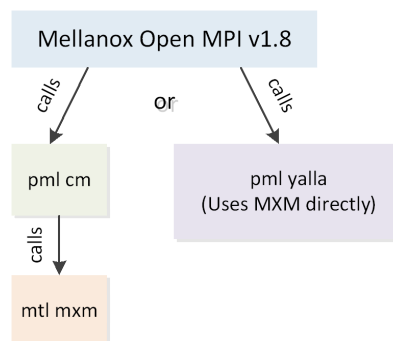
http://www.mellanox.com/page/products_dyn?product_family=189&mtag=hpc-x

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).



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. This feature 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 set the LID of the port according to the LMC that is set in the fabric by the SM. If the `LMC>0` then use the `MXM_IB_LID_PATH_BITS` parameter to specify an offset which will be added to the port's base LID in the range that is allowed by the LMC. The default offset is set to 0.

For example:

```
% mpirun -x MXM_IB_LID_PATH_BITS=1 ...
```

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.

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

When using an application which has sparse communication, 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 Utilities



When MXM is used as part of HPC-X software 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.

5.13.1 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=<...>
```

5.13.2 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 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 OpenSHMEM

HPC-X 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.

6.2 Running HPC-X OpenSHMEM

6.2.1 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 communications in the network, alleviating bottlenecks within the parallel communication libraries

6.2.1.1 Enabling MXM for HPC-X OpenSHMEM Jobs

MXM is activated automatically in ScalabeSHMEM for jobs with Number of Elements (PE) higher or equal to 128.

➤ **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](#).

6.2.1.2 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 ...
```

6.2.2 Running HPC-X™ OpenSHMEM with FCA

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 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](#).

6.2.3 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;
}

```

6.2.4 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

```

6.2.4.1 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 currently not supported
- `verbs` - IB verbs allocator is used
- `mmap` - `mmap()` is used to allocate memory

By default HPC-X OpenSHMEM will try 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>`

6.2.4.2 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
```

6.3 Performance Optimization

6.3.1 Configuring Hugepages



Hugepages is a feature applicable to users using MLNX_OFED v1.5.3-3.0.0.

Hugepages can be allocated using the `/proc/sys/vm/nr_hugepages` entry, or by using the `sysctl` command.

- **To view the current setting using the `/proc` entry:**

```
# cat /proc/sys/vm/nr_hugepages
0
```

- **To view the current setting using the `sysctl` command:**

```
# sysctl vm.nr_hugepages
vm.nr_hugepages = 0
```

- **To set the number of huge pages using `/proc` entry:**

```
# echo 1024 > /proc/sys/vm/nr_hugepages
```

- **To set the number of hugepages using `sysctl :man`**

```
# sysctl -w vm.nr_hugepages=1024
vm.nr_hugepages = 1024
```

To allocate all the hugepages needed, you might need to reboot your system since the hugepages requires large areas of contiguous physical memory.

In time, physical memory may be mapped and allocated to pages, thus the physical memory can become fragmented. If the hugepages are allocated early in the boot process, fragmentation is unlikely to have occurred.

It is recommended that the `/etc/sysctl.conf` file be used to allocate hugepages at boot time.

For example, to allocate 1024 hugepages at boot time, add the line below to the `sysctl.conf` file:

```
vm.nr_hugepages = 1024
```


6.4 Tuning OpenSHMEM Atomics Performance

HPC-X OpenSHMEM uses separate communication channel to perform atomic operations. By default this channel is enabled 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

6.5 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 1.8 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.

6.5.1 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 1.8, 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 MOFED 1.8 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>
```

7 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.

7.1 HPC-X HPC-X HPC-X HPC-X Compiling and Running the UPC Application

7.1.1 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.

7.1.2 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>
```

7.1.2.1 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>

7.1.2.2 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`

7.2 FCA Runtime Parameters

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

Table 7 - Runtime Parameters

Parameter	Description
<code>-fca_enable <0 1></code>	Disables/Enables FCA support at runtime (default: disable).
<code>-fca_np <value></code>	Enables FCA support for collective operations if the number of processes in the job is greater than the <code>fca_np</code> value (default: 64).
<code>-fca_verbose <level></code>	Sets verbosity level for the FCA modules

Table 7 - Runtime Parameters

Parameter	Description
<code>-fca_ops <+/->[op_list]</code>	<p><code>op_list</code> - comma separated list of collective operations.</p> <ul style="list-style-type: none"> <code>-fca_ops <+/->[op_list]</code> - Enables/disables only the specified operations <code>-fca_ops <+/-></code> - Enables/disables all operations <p>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:</p> <ul style="list-style-type: none"> <code>GASNET_FCA_ENABLE_BARRIER</code> <code>GASNET_FCA_ENABLE_BCAST</code>, <code>GASNET_FCA_ENABLE_REDUCE</code>, <p>Note: All the operations are enabled by default.</p>

7.2.1 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 `srn` 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
```

7.2.2 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`.

7.3 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>
```