NVIDIA HPC-X Software Toolkit Rev 2.18.1 LTS

Table of Contents

1 **Overview** **6**
1.1 Software Download 6
1.2 Document Revision History 6
1.3 Related Documentation 6
2 **Release Notes** **7**
2.1 Changes and New Features 7
2.2 HPC-X General Support 7
2.2.1 HPC-X Requirements 7
2.2.2 HPC-X Content 8
2.2.3 Important Note 8
2.2.4 Supported Platforms and Operating Systems 8
2.3 Bug Fixes in this Version 9
2.4 Known Issues 9
3 **Installing and Loading HPC-X** **13**
3.1 Installing HPC-X 13
3.2 Building and Running Applications with HPC-X 13
3.3 Building HPC-X with the Intel Compiler Suite 13
3.4 Loading HPC-X Environment from Modules 14
3.5 HPC-X Environments 14
3.6 HPC-X and Singularity 15
4 **Running, Configuring and Rebuilding HPC-X** **16**
4.1 Profiling MPI API 16
4.2 Rebuilding Open MPI 16
4.2.1 Rebuilding Open MPI Using a Helper Script 16
4.2.2 Rebuilding Open MPI from HPC-X Sources 17
4.3 Loading KNEM Module 17
4.4 Running MPI with HCOLL 17
4.5 Direct Launch of Open MPI and OpenSHMEM using SLURM 'srun' 18
5 **HCOLL** **19**
5.1 Overview 19
5.2 Using HCOLL 20
5.2.1 Enabling HCOLL in Open MPI 21

5.2.2 Tuning HCOLL Setting ... 21
5.2.3 Selecting Ports and Devices ... 21
5.2.4 Enabling Offloaded MPI Non-blocking Collectives ... 21
5.2.5 Enabling Multicast Accelerated Collectives ... 22
5.2.6 Enabling NVIDIA SHARP Software Accelerated Collectives ... 22
5.2.7 GPU Buffer Support in HCOLL ... 23
5.2.8 Limitations ... 24
6 Unified Communication - X Framework Library ... 26
6.1 Overview ... 26
6.1.1 Supported CPU Architectures ... 26
6.2 Configuring UCX ... 26
6.2.1 Using UCX with OpenMPI ... 26
6.2.2 Configuring UCX with XPMEM ... 27
6.3 Tuning UCX Settings ... 27
6.4 UCX Features ... 30
6.4.1 Hardware Tag Matching ... 30
6.4.2 Single Root IO Virtualization (SR-IOV) ... 31
6.4.3 Adaptive Routing ... 32
6.4.4 CUDA GPU ... 33
6.4.5 Multi-Rail ... 33
6.4.6 Memory in Chip (MEMIC) ... 34
6.4.7 PKey Support ... 34
6.4.8 Close Protocol ... 35
6.4.9 RoCE LAG ... 35
6.4.10 Flow Control for RDMA Read Operations ... 35
6.4.11 PCIe Relaxed Ordering Support ... 35
6.4.12 UCX Configuration File ... 35
6.4.13 Instrumentation and Monitoring FUSE-based Tool ... 36
6.5 UCX Utilities ... 36
6.5.1 ucx_perftest ... 36
6.6 Generating UCX Statistics for Open MPI/OpenSHMEM ... 37
7 Unified Collective Communication (UCC) ... 39
7.1 TL/UCP Special Service Worker ... 39
7.2 Out-Of-Box Native GPU Allreduce ... 39

7.3 Data Type Support in CUDA Executor Component (EC) 40
7.4 EC/CUDA One-shot Kernel with Cooperative Launch 40
8 PGAS Shared Memory Access Overview 41
8.1 HPC-X Open MPI/OpenSHMEM 41
8.2 Running HPC-X OpenSHMEM 42
8.2.1 Running HPC-X OpenSHMEM with UCX 42
8.2.2 Developing Application using HPC-X OpenSHMEM together with MPI 43
8.2.3 HPC-X® OpenSHMEM Tunable Parameters 43
8.3 Tuning MTU Size to the Recommended Value 45
8.3.1 HPC Applications on Intel Sandy Bridge Machines 45
9 ClusterKit 47
9.1 Running ClusterKit 47
9.2 Running ClusterKit via Script 48
10 NCCL-RDMA-SHARP Plugins 49
10.1 Overview 49
10.2 NCCL UCX Plugin 49
10.2.1 Running NCCL UCX Plugin 49
10.2.2 Performance Tuning 49
10.3 NCCL SHARP Plugin 51
10.3.1 NCCL Test Benchmark Example 51
11 Common Abbreviations 52
11.1 Syntax Conventions 52
12 User Manual Revision History 53
13 Release Notes History 54
13.1 Release Notes Change Log History 54
13.1.1 HPC-X Toolkit Change Log History 54
13.1.2 FCA Change Log History 63
13.1.3 HPC-X™ Open MPI/OpenSHMEM Change Log History 64
13.2 Bug Fixes History 65

You can download a PDF here.

1 Overview

NVIDIA® 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), which accelerates the underlying send/receive (or put/get) messages. It also includes HCOLL, which accelerates the underlying collective operations used by the MPI/PGAS languages.

The documentation here relates to HPC-X:

- Release Notes
- User Manual

1.1 Software Download

Please visit NVIDIA HPC-X

1.2 Document Revision History

A list of the changes made to this document are provided in Document Revision History.

1.3 Related Documentation

Software	Reference
NVIDIA SHARP	https://docs.nvidia.com/networking/category/mlnxsharp

2 Release Notes

This is a long-term support (LTS) release. LTS is the practice of maintaining a software product for an extended period of time (up to three years) to help increase product stability. LTS releases include bug fixes and security patches.

Release Notes Update History

Revision	Date	Description
2.18.1	July 03, 2023	Initial release of this document.

2.1 Changes and New Features

HPC-X current version provides the following changes and new features:

Category	Description
SHARP Collective Operations	Added support for SHARP Reduce-Scatter and All-Gather collective operations with the concurrent use of NCCL v2.20 or above.
HPC-X Content	Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release: • NVIDIA SHARP v3.6.1

2.2 HPC-X General Support

2.2.1 HPC-X Requirements

The table below details the HPC-X requirements and platforms.

Platform	Versions
CUDA	12.x
GDRCopy	2.3
MLNX_OFED	23.10
NCCL	2.x
NVIDIA BlueField-2	24.38.1002
NVIDIA BlueField-3	32.38.1002
NVIDIA ConnectX-5/ConnectX-5 Ex	16.35.2000
NVIDIA ConnectX-6	20.38.1900
NVIDIA ConnectX-6 Dx	22.38.1900
NVIDIA ConnectX-6 Lx	26.38.1900
NVIDIA ConnectX-7	28.38.1900
XPMEM	2.7

2.2.2 HPC-X Content

The following communications libraries and acceleration packages are part of this NVIDIA HPC-X® package:

Library/Acceleration Package	Version Number
Open MPI	4.1
NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)	3.6.1
HCOLL	4.8
UCX	1.16.0
UCC	1.3
Open SHMEM specification compliance	1.4[1]
ClusterKit[2]	1.11
nccl-rdma-sharp-plugin[3]	2.5

1. Full Open SHMEM v1.4 support is available only if compiled with C11 Standard (see Rebuilding Open MPI from HPC-X™ Sources).
2. ClusterKit is a multifaceted node assessment tool for high performance clusters.
3. nccl-rdma-sharp plugin enables RDMA and Switch-based collectives (SHARP) with NVIDIA's NCCL library.

2.2.3 Important Note

When HPC-X is launched with Open MPI without a resource manager job environment (slurm,pbs, etc.), or when it is launched from a compute node, the default rsh/ssh-based launcher will be used. This launcher does not propagate environment variables to the compute nodes. Thus, it is important to ensure the propagation of LD_LIBRARY_PATH variable from HPC-x is done as follows.

```
mpirun -x LD_LIBRARY_PATH -np 2 -H host1,host2  $HPCX_MPI_TESTS_DIR/examples/hello_c
```

2.2.4 Supported Platforms and Operating Systems

The following table lists the supported operating systems and CPUs for the latest HPC-X.

Starting from HPC-X v2.9, HPC-X will no longer support PPC architecture.

Operating System	Platforms
RHEL/CentOS/Rocky 7.x	x86_64, aarch64

Operating System	Platforms
RHEL/CentOS/Rocky 8.x	x86_64, aarch64
RHEL/CentOS/Rocky 9.x	x86_64, aarch64
CentOS 8.x Stream	x86_64
CentOS 9.x Stream	x86_64
SLES 12 SP4	x86_64, aarch64
SLES 12 SP5	x86_64
SLES 15 SP2	x86_64
SLES 15 SP3	x86_64
SLES 15 SP4	x86_64
Ubuntu 18.04	x86_64
Ubuntu 20.04	x86_64, aarch64
Ubuntu 22.04	x86_64, aarch64
OpenEuler 20.03	x86_64, aarch64
Kylin 10 SP1	x86_64, aarch64
Kylin 10 SP2	x86_64, aarch64
Debian 10.x	x86_64
Debian 11.x	x86_64

2.3 Bug Fixes in this Version

N/A

2.4 Known Issues

The following is a list of general limitations and known issues of the various components of this HPC-X release.

Reference Number	Issue
3633383	**Description:** When allocating device memory (MEMIC) on some FW versions, by passing `memheap_base_device_nic_mem_seg_size` parameter to SHMEM runner, the process may crash with the error message "failed to allocate 4096 bytes on using md ib". In such cases, avoid using MEMIC.
	Workaround: N/A
	Keywords: MEMIC, SHMEM, Allocation
	Discovered in Version: 2.17.0

Reference Number	Issue
3606732	**Description:** In some cases, when using Cuda buffers for intra-node transfers, the program may crash with an assertion `` `offset <= key->b_len' `` failed in `cuda_ipc`. This happens due to a conflict between `cuda_ipc` and `gdrcopy` memory registration on the same buffer. In other cases, the error message "`gdr_map failed`" can be printed.
	Workaround: N/A
	Keywords: `gdr_copy, cuda_ipc`
	Discovered in Version: 2.17.0
3586369	**Description:** When UD transport is being used explicitly, the MPI or SHMEM job may hang during cleanup or `MPI_Finalize`, while waiting for UCX endpoint flush operation to complete.
	Workaround: Disable adaptive progress optimization by setting the environment variable `UCX_ADAPTIVE_PROGRESS=n`, or don't select UD transport explicitly.
	Keywords: Hang, UD, Flush
	Discovered in Version: 2.17.0
3653404	**Description:** When registering a large memory region `with ucp_mem_map()`, and peer failure handling support is enabled on the UCX endpoint, the process may crash with the error "LRU push returned Unsupported operation" while sending a buffer belonging to that region. The issue happens because multi-threaded registration is being used for large regions, and it does not work well with peer failure support.
	Workaround: Disable multi-thread registration by setting the environment variable "`UCX_REG_MT_THRESH=inf`".
	Keywords: Multi-Threaded, Indirect, Key Registration
	Discovered in Version: 2.17.0
3606445	**Description:** The performance of `osu_mbw_mr` for some message sizes can be worse than the previous release. This can happen because of different default protocol thresholds.
	Workaround: Revert to previous thresholds selection logic by setting the environment variable to `UCX_PROTO_ENABLE=n`
	Keywords: Performance, `osu_mbw_mr`
	Discovered in Version: 2.17.0
-	**Description:** In order to get the best performance when running on ConnectX-7 NDR400 fabric, the following parameter should be set with mpirun. `mpirun -x UCX_MAX_RNDV_LANES=4 -x UCX_RNDV_THRESH=20k …`
	Workaround: N/A
	Keywords: ConnectX-7; UCX; mpirun
	Discovered in Version: 2.11 (UCX 1.13)
2705762	**Description:** UCX job may hang when the DC transport is used.
	Workaround: • Exclude RoCE LAG devices from the list of available devices (managed by UCX_NET_DEVICES environment variable) and make sure UCX_IB_NUM_PATH is set to 1. • Exclude DC from the list of available transports managed by the UCX_TLS environment variable (e.g. set UCX_TLS=sm,self,rc,tcp).

Reference Number	Issue
	Keywords: UCX
	Discovered in Version: 2.9 (UCX 1.11)
-	**Description: Once the** TCP detects a `“Connection reset by a peer”` failure on a connection, it stops sending data, and the MPI/SHMEM application hangs. Error printouts from the UCP/UCT can be seen in the log.
	Workaround: On small scale cases, change the `"UCX_TLS=tcp" to "UCX_TLS=sm,tcp"` parameter. On larger scales this workaround is not applicable.
	Keywords: UCX hang
	Discovered in Version: 2.9 (UCX 1.11)
-	**Description:** NCCL plugin works only with NCCL v2.8 or higher.
	Workaround: Build plugin version v2.0 from the following source. https://github.com/Mellanox/nccl-rdma-sharp-plugins/tree/v2.0.x
	Keywords: NCCL Plugin
	Discovered in Version: 2.7 (NCCL 2.1)
-	**Description**: UD timeout error may appear.
	Workaround: Disable the UD transport and use DC instead. Set `UCX_TLS=dc_x,self,sm`
	Keywords: UD, DC, timeout, UCX
	Discovered in Version: 2.7 (UCX 1.9)
2235234	**Description:** On some platforms, GPUDirect RDMA does not work reliably when the path between HCA and GPU traverses QPI link.
	Workaround: Disable GPUDirect support in UCX by setting UCX_IB_GPU_DIRECT_RDMA=n.
	Keywords: GPUDirect. RDMA, UCX
	Discovered in Version: 2.7 (UCX 1.9)
4549	**Description:** UCX may fail to compile with Clang compiler version 9 if `--dynamic-list-data` flag is used in the compilation. (Github issue: https://github.com/openucx/ucx/issues/4549)
	Workaround: [optional] Compile UCX without using this flag. However, note that ucx_perftest will not be available for usage.
	Keywords: Clang compiler, UCX
	Discovered in Version: 2.6 (UCX 1.8)
-	**Description:** When using GPU memory on an InfiniBand network with GPUDirect enabled yet without gdrcopy library, performance of small messages can be low.
	Workaround: Use the Rendezvous protocol by setting the UCX_RNDV_THRESH parameter to 0.
	Keywords: GPU, GPUDirect, memory
	Discovered in Version: 2.6 (UCX 1.8)
4105	**Description:** Adaptive Routing is not supported when used with OpenSHMEM applications. (Github issue: https://github.com/openucx/ucx/issues/4105)
	Workaround: N/A

Referenc e Number	Issue
	Keywords: Adaptive Routing, AR, OpenSHMEM, OSHMEM
	Discovered in Version: 2.5 (OpenSHMEM 1.4)
-	**Description:** In ConnectX-4 and Connect-IB HCAs, when the DC transport is used on a large scale, “Retry exceeded” messages may be printed from UCX.
	Workaround: Configure SL2VL on your OpenSM in the fabric and make UCX use SL=1 when using the InfiniBand transports via '-x UCX_IB_SL=1'.
	Keywords: UCX, DC transport, ConnectX-4, Connect-IB
	Discovered in Version: 2.1 (UCX 1.3)
-	**Description:** When UCX requires more memory utilization than the memory space defined in /proc/sys/kernel/shmmni file, the following message is printed from UCX: “... total number of segments in the system (%lu) would exceed the limit in /proc/sys/kernel/shmmni (=%lu)... please check shared memory limits by 'ipcs -l”.
	Workaround: Follow the instructions in the error message above and increase the value of shared memory segments in /proc/sys/kernel/shmmni file.
	Keywords: UCX, memory
	Discovered in Version: 2.1 (UCX 1.3)
1162	**Description:** UCX currently does not support canceling send requests. (Github issue: https://github.com/openucx/ucx/issues/1162)
	Workaround: N/A
	Keywords: UCX
	Discovered in Version: 2.0
-	**Description:** UCX job hangs with SocketDirect/MultiHost/SR-IOV.
	Workaround: Set UCX_IB_ADDR_TYPE=ib_global
	Keywords: UCX
-	**Description:** As UCX embedded in the HPC-X is compiled with AVX support, UCX cannot be run on hosts without AVX support. In case the AVX is not available, recompile the UCX that is available in the HPC-X with the option: --with-avx=no
	Workaround: Recompile UCX with AVX disabled: $ ./utils/hpcx_rebuild.sh --rebuild-ucx --ucx-extra-config "--with-avx=no"
	Keywords: UCX

3 Installing and Loading HPC-X

3.1 Installing HPC-X

➢ *To install HPC-X:*

1. Extract hpcx.tbz into your current working directory.

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

2. Update shell variable of the location of HPC-X installation.

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

3.2 Building and Running Applications with HPC-X

HPC-X includes Open MPI v4.1.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 MPI v4.1.x).

➢ *To load Open MPI/OpenSHMEM v4.1.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
```

3.3 Building HPC-X with the Intel Compiler Suite

As of version 1.7, HPC-X builds are no longer distributed 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 above example, 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 suite 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.

For rebuilding HPC-X open-source components, please use the helper script as described in "Rebuilding Open MPI Using a Helper Script" section.

3.4 Loading HPC-X Environment from Modules

➤ *To load Open MPI/OpenSHMEM v4.1.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
```

3.5 HPC-X Environments

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

- HPC-X with CUDA® support - *hpcx*

  Cuda support in SLES 11, RHEL 6 and RHEL OSs lower than 7.4 with PPC arch is no longer available.

  This is the default option which is optimized for best performance for the single-thread mode. This option supports both GPU and non-GPU setups.

  Starting with CUDA 11.0, the minimum recommended GCC compiler is at least GCC 5 due to C++11 requirements in CUDA libraries e.g. cuFFT and CUB.
- 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 for profiling - `hpcx-prof`
  This option enables UCX compiled with profiling information.
- HPC-X for debug - `hpcx-debug`
  This option enables UCX/HCOLL/SHARP compiled in debug mode.
- HPC-X stack - `hpcx-stack`
  This environment contains all the libraries that 'Vanilla HPCX' has, except for OMPI.

When HPC-X is launched with Open MPI without a resource manager job environment (slurm,pbs, etc.), or when it is launched from a compute node, the default rsh/ssh-based launcher will be used. This launcher does not propagate environment variables to the compute nodes. Thus, it is important to ensure the propagation of LD_LIBRARY_PATH variable from HPC-x is done as follows.

```
mpirun -x LD_LIBRARY_PATH -np 2 -H host1,host2  $HPCX_MPI_TESTS_DIR/examples/hello_c
```

Note that only one of the 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).

3.6 HPC-X and Singularity

HPC-X supports Singularity containerization technology, which help 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 post
- $HPCX_HOME/utils/singularity/hpcx-singularity.md file inside HPC-X package

4 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.

4.1 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 NVIDIA®-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'.

4.2 Rebuilding Open MPI

4.2.1 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
```

4.2.2 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 and HCOLL, and use them by default.

If HPC-X is intended to be used with SLURM PMIx plugin, Open MPI should be built 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>
```

4.3 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.

4.4 Running MPI with HCOLL

HCOLL is enabled by default in HPC-X.

- Running with default HCOLL configuration parameters:

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

- Running OSHMEM with HCOLL:

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

4.5 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 '*

5 HCOLL

5.1 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.

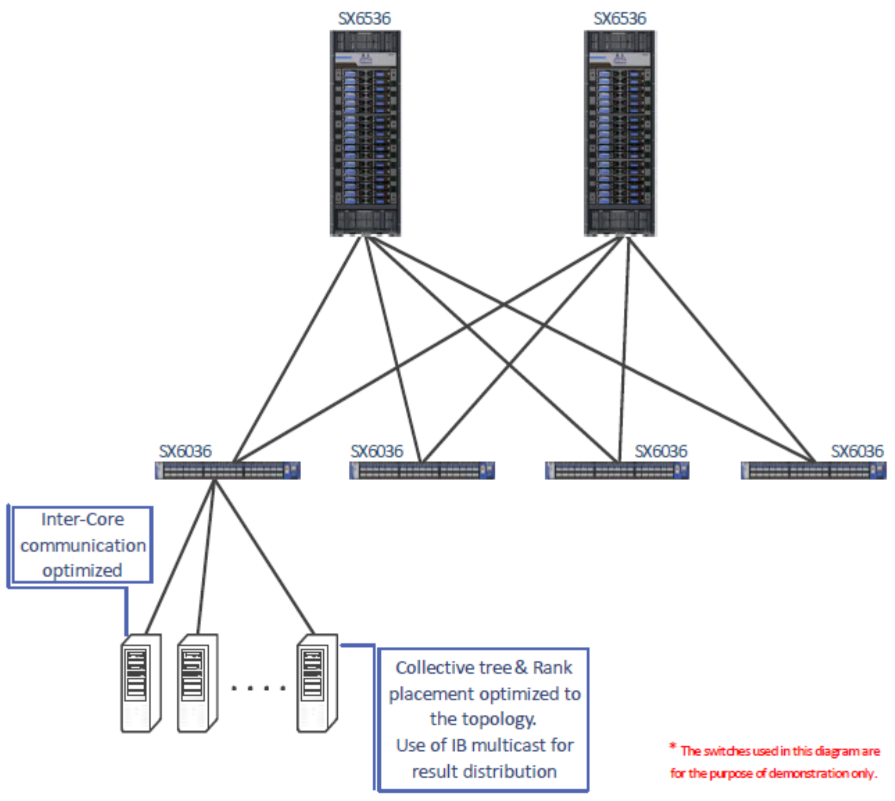
Collective communications execute global communication operations to couple all processes/nodes in the system and therefore must be executed as quickly and as efficiently as possible. Indeed, the scalability of most scientific and engineering applications is bound by the scalability and performance of 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 the meaningful computation.

The two issues of lost CPU cycles and performance loss to the effects of system noise have been addressed by offloading the communications to the host channel adapters (HCAs) and switches. The technologies of SHARP (Scalable Hierarchical Aggregation and Reduction Protocols) and CORE-Direct® (Collectives Offload Resource Engine) provide 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.

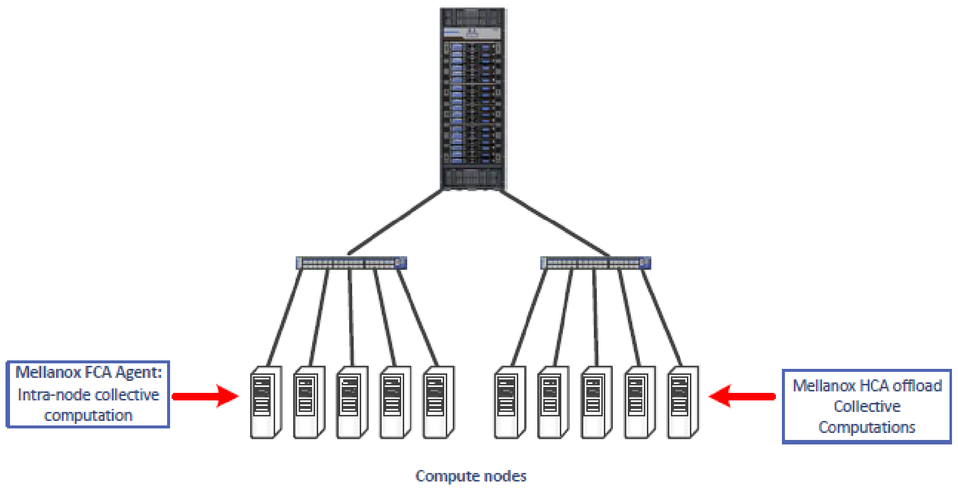
Additionally, HCOLL contains support for building runtime configurable hierarchical collectives. HCOLL leverages hardware multicast capabilities to accelerate collective operations. In HCOLL, the performance and scalability of the UCX point-to-point library in the form of the "ucx_p2p" BCOL is fully taken advantage of. This enables users to leverage NVIDIA hardware offloads transparently and with minimal effort.

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

The following diagram summarizes the HCOLL architecture:


SX6536
SX6536
SX6036
SX6036
SX6036
SX6036
Inter-Core communication optimized
Collective tree & Rank placement optimized to the topology.
Use of IB multicast for result distribution
* The switches used in this diagram are for the purpose of demonstration only.

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


Mellanox FCA Agent: Intra-node collective computation
Mellanox HCA offload Collective Computations
Compute nodes

5.2 Using HCOLL

HCOLL is part of the HPC-X software toolkit and does not require any special installation.

5.2.1 Enabling HCOLL in Open MPI

HCOLL is enabled by default with HPC-X. Users can explicitly disable it using the following MCA parameter.

```
%mpirun -np 32 -mca coll_hcoll_enable 0 ./a.out
```

5.2.2 Tuning HCOLL Setting

The default HCOLL settings should be optimal for most systems. To check the available HCOLL parameters and their default values, run the following command after loading HPC-X.

```
% $HPCX_HCOLL_DIR/bin/hcoll_info -all
```

HCOLL parameters are simply environment variables and can be modified in one of the following ways:

- Modify the default HCOLL parameters as part of the *mpirun* command.

  ```
  % mpirun ... -x HCOLL_ML_BUFFER_SIZE=65536
  ```
- Modify the default HCOLL parameter values from SHELL:

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

5.2.3 Selecting Ports and Devices

➤ *To select the HCA device and port you would like HCOLL to run over:*

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

5.2.4 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
```

The supported non-blocking MPI collectives are:Note that enabling non-blocking MPI collectives will disable multicast acceleration in blocking MPI collectives.

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

5.2.5 Enabling Multicast Accelerated Collectives

HCOLL 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.

5.2.5.1 Configuring IPoIB

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

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)
```

2. Use /sbin/ifconfig to get the address information 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.

5.2.6 Enabling NVIDIA SHARP Software Accelerated Collectives

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

➢ *To enable NVIDIA SHARP acceleration:*

```
-x HCOLL_ENABLE_SHARP=1
```

➢ *To disable NVIDIA SHARP acceleration:*

```
-x HCOLL_ENABLE_SHARP=0
```

➤ *To change the NVIDIA SHARP message threshold:*

```
-x HCOLL_BCOL_P2P_ALLREDUCE_SHARP_MAX=<threshold> ( default: tune based on sharp resources)
```

The maximum small message allreduce algorithm runs through SHARP. Messages with a size greater than the above will use SHARP streaming aggregation or fall back to non-SHARP-based algorithms (multicast based or non-multicast based).

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

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

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

5.2.7 GPU Buffer Support in HCOLL

HCOLL of version >= 4.4 supports collective operations over GPU buffers. The supported GPU HW includes NVIDIA® GPUs starting from Tesla K80.

- Minimal SW requirement: CUDA® (version >= 9.0).

If CUDA runtime is available during MPI job, HCOLL will automatically enable GPU support. Collective operations that support GPU buffers:

- Allreduce
- Bcast
- Allgather

If some other collective operation API of libhcoll is called with GPU buffer, then the call would return HCOLL_ERROR after the buffer type check.

Recommended Additional SW

- NCCL (version >= 2.4).
  It is recommended to install libnccl for better performance. If it is not available, HCOLL will print a warning regarding potentially lower performance. The warning can be suppressed by setting "`-x HCOLL_CUDA_BCOL=ucx_p2p -x HCOLL_CUDA_SBGP=p2p`".
- GPUDirect RDMA nv_peer_mem. If nv_peer_mem module is loaded on all nodes, then Bcast operation over GPU buffers will be optimized with HW Multicast.

The control parameter is `HCOLL_GPU_ENABLE = <0,1,-1>`

where:

Parameter	Description
0	GPU support disabled. The type of the user buffers' pointers is not checked. In such a case, if the user provides the buffer allocated on GPU, the behavior is undefined.

Parameter	Description
1	GPU support enabled. The buffer pointer is checked and HCOLL GPU collectives are enabled. This is the default value if the CUDA runtime is available.
-1	Partial GPU support. The buffer pointer is checked and HCOLL falls back to the runtime in the case of GPU buffer.

Limitations

Not all combinations of (OP, DTYPE) are supported for MPI_Allreduce with GPU buffers.

Supported operations:

- SUM
- PROD
- MIN
- MAX

Supported types:

- INT8,16,32,64
- UINT8,16,32,64
- FLOAT16,32,64

5.2.8 Limitations

- As of v4.1 release, HCOLL 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.

6 Unified Communication - X Framework Library

6.1 Overview

Unified Communication - X Framework (UCX) is an 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 allows 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 is also highly useful for storage, big-data and cloud domains where client-server based applications are used.

UCX supports:

- 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 v5.0 and higher.

    - Accelerated verbs
- Shared Memory communication with support for KNEM, CMA and XPMEM
- RoCE
- TCP
- CUDA

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

6.1.1 Supported CPU Architectures

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

6.2 Configuring UCX

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

6.2.1 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 ...
```

6.2.2 Configuring UCX with XPMEM

By default, UCX library embedded within HPC-X is compiled with an open source version of the XPMEM driver. The recommended version of the XPMEM driver is from: https://github.com/openucx/xpmem.
In order to compile UCX with another version of XPMEM, 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_LIBRARY_PATH for Open MPI to use the new location:

   ```
   % mpirun -mca pml ucx -x LD_LIBRARY_PATH=/path/to/new/ucx/install/lib:$LD_LIBRARY_PATH ...
   ```

   When UCX is compiled from sources, it can be configured for best performance.
   To accomplish this, please compile UCX with:

   ./contrib/configure-release --enable-optimizations

6.3 Tuning UCX Settings

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

➢ *To check the UCX version, run:*

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

UCX parameters can be modified using 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 (when running as part of a resource manager job):

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

  (when running as part of a resource manager job):
- 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.

- Excluding specific transports from the command line:

  ```
  mpirun -mca pml ucx -x UCX_TLS=^rc  ...
  ```
  The above command line will select pml ucx and use all its available transports except for rc. The rc transport will be excluded from usage.

  As of HPC-X v2.5, shared memory has new transport naming. The available shared memory transports are: posix, sysv and xpmem.

  The 'device' name for the shared memory transport is 'memory' (for usage in UCX_SHM_DEVICES).
- When selecting one of the several devices or interfaces in the server, please use the UCX_NET_DEVICES flag to specify which RDMA device you would like to use.

  ```
  $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 DC initiator QPs (DCI) number 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
  ```
  In case the DC transport is not available or disabled on a large scale, UCX will fall back to the UD transport.
  The RC transport is disabled after 256 established connections. The counter of established connections can be overridden using the UCX_RC_MAX_NUM_EPS environmental parameter.

- 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_0:1
      ```

- 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_0:1 -x UCX_TRAFFIC_CLASS=106 -x UCX_IB_GID_INDEX=3
  ```

- 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.

- Setting `UCX_IB_ADDR_TYPE=ib_global` when running on GID-based multi-host setup (see also Single Root IO Virtualization (SR-IOV) section below).
- Enabling various optimizations intended for homogeneous environment. Enabling this mode implies that the local transport resources/devices of all entities that connect to each other are the same.

  ```
  UCX_UNIFIED_MODE=y
  ```

- Using `-x UCX_IB_SUBNET_PREFIX` to filter for the InfiniBand subnet prefix (empty means no filter). This is relevant for IB link layer only. For example, a filter for the default subnet prefix can be specified as follows: fe80:0:0:0.
- Specifying how DC initiator (DCI) is selected by the endpoint with UCX_DC_MLX5_TX_POLICY=<policy> (relevant for DC transport only). The policy options are:

Policy	Description
`dcs`	The endpoint either uses already assigned DCI, or DCI is allocated in a LIFO order and gets released once it has no outstanding operations
`dcs_quota`	Same as "dcs". In addition, the DCI is scheduled for release in case it has sent more than one quota and there are endpoints waiting for a DCI. The DCI is released once it completes all outstanding operations. This policy ensures that there will be no starvation among endpoints
`rand`	Every endpoint is assigned with a randomly selected DCI. Multiple endpoints may share the same DCI

- Using UCX CUDA memory hooks may not work with static building CUDA applications. As a workaround, extend the configuration with the following options:

```
-x UCX_MEMTYPE_CACHE=0 -x HCOLL_GPU_CUDA_MEMTYPE_CACHE_ENABLE=0 -x HCOLL_GPU_ENABLE=1
```

- Disabling GPU memory staging protocols and using only `GPUDirectRDMA`, if possible:

```
-x UCX_RNDV_SCHEME=get_zcopy
```

- Running the application on close NUMA nodes:

```
mpirun -mca rmaps_dist_device <HCA name> -mca rmaps_base_mapping_policy dist:span
```

- The shared memory new transport naming:
  The available shared memory transport names are: posix, sysv and xpmem.
  'sm' and 'mm' will include all the three mentioned above.
  The 'device' name for the shared memory transport is 'memory' (for usage in UCX_SHM_DEVICES)
- To get more information in case of any error (for troubleshooting purposes), please set the following environment parameter:

```
mpirun -mca pml ucx -x UCX_LOG_LEVEL=diag ...
```

- DC full handshake config can be set by the environment variables UCX_DC_MLX5_DCI_FULL_HANDSHAKE, UCX_DC_MLX5_DCI_KA_FULL_HANDSHAKE, UCX_DC_MLX5_DCT_FULL_HANDSHAKE. Possible values are: on / off / auto, with the default being “off”. In auto mode, FH will be used according to the AR config of the SL in use (if the SL is with AR – FH will be used, otherwise – HH).

6.4 UCX Features

6.4.1 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 accelerated RC and DC transports (RC_X and DC_X), and can be enabled in UCX with the following environment parameters:

- For the RC_X transport:

```
UCX_RC_MLX5_TM_ENABLE=y
```

- For the DC_X transport:

  ```
  UCX_DC_MLX5_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_BB_SIZE 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, which is controlled by `UCX_RC_MLX5_TM_MAX_BCOPY` or `UCX_DC_MLX5_TM_MAX_BCOPY` variables (for RC_X and DC_X transports, respectively). 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 Understanding Tag Matching for Developers post.

6.4.2 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.

This feature is supported on ConnectX-5 HCAs and above only.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 Multi-Rail.).
- SRI-OV is not supported with dc and dc_x transports in UCX.

6.4.3 Adaptive Routing

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_x transports.

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

Enabling Adaptive Routing on a certain SL is done according to the following table.

		`UCX_IB_AR_ENABLE=yes`	`UCX_IB_AR_ENABLE=no`	`UCX_IB_AR_ENABLE=try`	`UCX_IB_AR_ENABLE=auto`
`UCX_IB_SL=auto`	`AR enabled on some SLs`	`Use 1st SL with AR`	`Use 1st SL without AR`	`Use 1st SL with AR`	`Use SL=0`
	`AR enabled on all SLs`	`Use SL=0`	`Failure`	`Use SL=0`	`Use SL=0`
	`AR disabled on all SLs`	`Failure`	`Use SL=0`	`Use SL=0`	`Use SL=0`
`UCX_IB_SL=<sl>`	`AR enabled on <sl>`	`Use SL=<sl>`	`Failure`	`Use SL=<sl>`	`Use SL=<sl>`
	`AR disabled on <sl>`	`Failure`	`Use SL=<sl>`	`Use SL=<sl>`	`Use SL=<sl>`

Adaptive routing is not supported for OpenSHMEM applications.

6.4.3.1 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)

6.4.4 CUDA GPU

6.4.4.1 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.

6.4.4.2 Supported Architectures

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

6.4.4.3 System Requirements

- CUDA v8.0 or higher - for information on how to install CUDA, refer to NVIDIA documents for CUDA Toolkit. This version of HPC-X is compiled with CUDA v12.x.
- MLNX_OFED GPUDirect RDMA plugin module - for information on how to install:
  - MLNX_OFED - refer to MLNX_OFED webpage
  - GPUDirect RDMA - refer to MLNX_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 computing 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
    ```
- 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.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

  ➤ *To check whether the GDR COPY module is loaded, run:*
  lsmod | grep gdrdrv

6.4.5 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. When using Socket Direct cards, the Multi-Rail capability becomes essential.

Each process would be able to use up to the first 4 active ports on the host in parallel (this 4 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=4 ...
```

➢ *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=4 ...
```

Possible values for these parameters are 1, 2, 3 and 4. The default values are `UCX_MAX_EAGER_LANES =1,` and `UCX_MAX_RNDV_LANES = 2`.

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

⚠ Starting from HPC-X v2.8, multi-rail is also supported out-of-box for the client-server API. To enable or disable it, use the following environment parameter: `UCX_CM_USE_ALL_DEVICES=y/n`

6.4.6 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 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.

6.4.7 PKey Support

UCX supports the usage of a non-default PKey. In order to specify which PKEY value to use, please set it with the following environment parameter: `UCX_IB_PKEY`.
Valid values are between 0 - 0x7fff.
In an environment where the default PKey is not found, the PKey in index 0 will be used.

6.4.8 Close Protocol

When using the UCX client-server API for connection establishment, it is also possible to have a graceful teardown, i.e a disconnection, between each pair of client and the server it's connected to, at the end of the communication. Either side can be the initiator of the disconnection.

6.4.9 RoCE LAG

UCX now supports RoCE LAG out-of-box.
UCX is now able to detect a RoCE LAG device and automatically create two RDMA connections to utilize the full bandwidth of LAG interface.
For Ethernet packets, the network switch path is usually determined by a hash function on the packet's IP and UDP header fields. In order to force using distinct paths for various switch topologies, it is possible to set “UCX_ROCE_PATH_FACTOR=n” environment variable to influence UDP.source_port field: the first connection will use “UDP.source_port=0xC000”, while the second connection will use “UDP.source_port=0xC000+<n>”.
The default value for UCX_ROCE_PATH_FACTOR is 1. This feature is currently supported for RC transport only.

6.4.10 Flow Control for RDMA Read Operations

This feature is intended to prevent network congestion when many processes send messages to the same destination. To reduce network pressure, the user may limit the number of simultaneously transferred data by setting UCX_RC_TX_NUM_GET_BYTES environment variable to a certain value (e.g. 10MB). In addition, to achieve better pipelining of network transfer and data processing, the user may limit the maximal message size which can be transferred using RDMA Read operation by setting UCX_RC_MAX_GET_ZCOPY environment variable to a certain value (e.g. 64KB).

6.4.11 PCIe Relaxed Ordering Support

UCX supports enabling Relaxed Ordering for PCIe Write transactions in order to improve performance on systems where the PCI bandwidth of relaxed-ordered Writes is higher than that of the default strict-ordered Writes.
The environment variable UCX_IB_PCI_RELAXED_ORDERING can force a specific behavior: “on” enables relaxed ordering; “off” disables it; while “auto” (default) sets relaxed ordering mode based on the system type.

6.4.12 UCX Configuration File

The UCX configuration file enables the user to apply configuration variables set by the user in the $HPCX_UCX_DIR/etc/ucx/ucx.conf file. A configuration file can be created with initial default values by running `"ucx_info -fC > $HPCX_UCX_DIR/etc/ucx/ucx.conf"`.
The values are applied in the following order of precedence:
1. If an environment variable is set explicitly, it overrides the file's configuration.
2. Otherwise, value from $HPCX_UCX_DIR/etc/ucx/ucx.conf is used if it exists.
3. Otherwise, default (compile-time) value is used.

> ⚠ The configuration file applies settings only to the host where it is located.

6.4.13 Instrumentation and Monitoring FUSE-based Tool

This new functionality enables the user to analyze UCX-based applications in runtime. The tool is based on Filesystem in Userspace (FUSE) interface. If the feature is enabled, a directory for each process using UCX will be created in `/tmp/ucx`. The directory name is the PID of the target process. The process directory contains three sub-directories: UCP, UCT, UCS.

> ⚠ This feature requires rebuild of UCX with `"--with-fuse3"` flag in the configure line. UCX inside HPC-X is not built with this option by default.

While building, UCX checks for fuse3 library presence and enables building the tool. Once UCX is built, the `ucx_vfs` binary will be created in the install directory and will be used to launch a daemon process and enable UCX-based applications analysis.

You can use the `UCX_VFS_ENABLE` environment variable to control the feature. It is set to 'y' by default. Setting the variable to 'n' disables creating the service thread in user's UCX application.

6.4.13.1 Requirements

For the feature to function properly, the following is required:

- fuse3 utilities to run the daemon and analyze applications
- fuse3 library to build the tool

6.4.13.2 Limitations

- ucx_vfs daemon must be started before the target processes. Otherwise, if the number of processes exceeds the limit, fs.inotify.max_user_instances are increased.
- If the user starts simultaneously more than the maximum allowed number of processes and then starts the daemon, only the first processes that meet the limit will be monitored by the tool.

6.5 UCX Utilities

6.5.1 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/ucx_perftest -help* .

Examples:

- 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 ucp_am_bw`

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

6.6 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.

- The destination is set by UCX_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>	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:

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 processes 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/StatisticsIt 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:

- 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.

When there are several devices or interfaces in the server, please use the UCX_NET_DEVICES flag to specify which RDMA device you would like to use.

7 Unified Collective Communication (UCC)

Unified Collective Communication (UCC) was codesigned with industry partners for PyTorch-based deep learning recommender model training on multi-rail GPU platforms. UCC has been specifically designed and implemented for high-performance PGAS applications and runtimes. It serves as a drop-in replacement for HCOLL and will gradually assume the role of default collective library once UCC fully implements the range of HCOLL's hierarchical algorithms.

For further information on what UCC is and how to use it, please see https://github.com/openucx/ucc

Please see UCC PyTorch integration layer, Torch_UCC at https://github.com/facebookresearch/torch_ucc

UCC is supported in both MPI and OSHMEM. However, it is not enabled by default.

- To enable it in MPI, set -mca `coll_ucc_enable` to 1.
- To enable it in OSHMEM, set -mca `coll_scoll_enable` to 1.

7.1 TL/UCP Special Service Worker

This feature enables the use of a separate UCX/UCP worker for performing the service collectives, which are invoked internally during setup. For example, service collectives can be set to use TCP only, while regular collectives may use InfiniBand.

The feature can be enabled by setting the UCC environment variable as follows:

```
UCC_TL_UCP_SERVICE_WORKER=1.
```

You may pass the UCX configuration for the service worker using the "UCC_TL_UCP_SERVICE_" prefix. For example:

```
UCC_TL_UCP_SERVICE_NET_DEVICES=mlx5_0:1
```

For further UCC options, run `ucc_info -f`

7.2 Out-Of-Box Native GPU Allreduce

This feature enables UCC library to detect the NVIDIA NVLink topology and select the best GPU-based algorithms for supported collectives (Allgather/v, Reducescatter/v).

To view the NVLink topology, run `nvidia-smi topo -m`

To activate this feature, make sure to enable the hierarchical component in UCC using the UCC_CLS environment variable as follows:

```
UCC_CLS=basic,hier.
```

To view all available UCC items and options, run `ucc_info -f`

7.3 Data Type Support in CUDA Executor Component (EC)

This feature enables out-of-box support for all datatypes and reduction operations for UCC collectives for GPUs.

Supported datatypes: float32, float64, float32_complex, float64_complex, unsinged and signed int8, int16, int32, int64.

Supported reduction operations: sum, prod, avg, min, max, and, bitwise and, or, bitwise or, xor, bitwise xor.

7.4 EC/CUDA One-shot Kernel with Cooperative Launch

This feature improves GPU collective performance by utilizing the CUDA cooperative launch feature. It enables the use of a single CUDA kernel for CUDA operations in UCC GPU collectives.

This feature can be activated by enabling the UCC environment variable UCC_EC_CUDA_USE_COOPERATIVE_LAUNCH as follows:

```
UCC_EC_CUDA_USE_COOPERATIVE_LAUNCH=1
```

8 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 to 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.

8.1 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) and HCOLL, providing an unprecedented level of scalability for SHMEM programs running over InfiniBand.

8.2 Running HPC-X OpenSHMEM

8.2.1 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/

8.2.1.1 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
```

8.2.2 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;
}
```

8.2.3 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 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 benchmarks in order to free memory:

echo 3 > /proc/sys/vm/drop_caches

8.2.3.1 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 supportedMemheap can be allocated with the following methods:
- 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>*

8.2.3.2 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
```

8.2.3.3 OpenSHMEM MCA Parameters for shmem_quiet, shmem_fence and shmem_barrier_all

Default synchronization algorithms of OSHMEM may be tuned by spml_ucx_strong_sync parameter:

0 - don't do strong synchronization (default)

1 - use non-blocking get

2 - use blocking get

3 - use flush operation

8.3 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.

8.3.1 HPC Applications on Intel Sandy Bridge Machines

Intel Sandy Bridge machines have NUMA hardware related limitation which affects the 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>
```

9 ClusterKit

ClusterKit is a multifaceted node assessment tool for high performance clusters. Currently, ClusterKit is capable of testing latency, bandwidth, effective bandwidth, memory bandwidth, GFLOPS by node, per-rack collective performance, as well as bandwidth and latency between GPUs and local/remote memory. ClusterKit employs well known techniques and tests to arrive at these performance metrics and is intended to give the user a general look at the health and performance of a cluster.

9.1 Running ClusterKit

After loading the HPC-X package, and in a job allocation, issue one of the following commands.

➤ *To test a specific network device:*

```
mpirun  -x UCX_NET_DEVICES=mlx5_4:1  $HPCX_CLUSTERKIT_DIR/bin/clusterkit
```

ClusterKit runs by default in pairwise test cases, which requires at least two nodes to run.

➤ ***To allow UCX to choose the network device/devices:***

```
mpirun  $HPCX_CLUSTERKIT_DIR/bin/clusterkit
```

Note that multi-rail is enabled by default.

When not using a job scheduler, the mpirun command line arguments that specify the hosts should be added.

The application will run with the default set of tests. Run with --help to see all command line options. During the program run, interim results for each test are printed, so you can track the progress. This is particularly important for very large clusters, with thousands of nodes.

Towards the end of the program output, you will see the name of the output directory, which is based on the time and date, and should be similar to the following.

```
Output directory: 20190915_061634/
```

The output directory is automatically created, and .json and .txt results are written for each test.

The .txt files are human readable, the .json files are for importing into the UFM-hosted viewer. For small scale, the .txt files generally suffice, but for larger clusters, the UFM-hosted viewer is recommended for viewing the .json files.

9.2 Running ClusterKit via Script

Clusterkit can also be run using the supplied clusterkit.sh convenience script. This script provides a simple interface to configure some internal UCX parameters.

```
./clusterkit.sh [options] <parameters>

        Parameters:
        -v|--verbose                  Set verbose mode
        -f|--hostfile <hostfile>      File with newline separated hostnames to run tests on.
        -r|--hpcx_dir <path>          Path to HPCX installation root folder (or use env HPCX_DIR)

        Options:
        -p|--ppn <number>             Select number of processes per hostname (default: 1)
        -d|--hca_list "string"        Comma separated list of HCAs to use (default: autoselect)
        -t|--transport_list "string"  List of RDMA transports to use (rc,dc,ud) (default: autoselect best)
        -z|--traffic <nn>             Run traffic for 'nn' minutes
        -s|--ssh                      Use ssh for process launching (default: autoselect)
        -h|--help                     Show help message
        -n|--dry-run                  Dry run (do nothing, only print)
        -m|--map-by                   [node|core|socket]  (Used in MPI argument: -- map-by ppr:ppn:map-by)
        -y|--bycore                   Run on ALL cores, not just a single core per node
        -k|--test_intra_node          Run intra-node tests for bandwidth and latency (default: skip intra-node)
        -U|--unidirectional           Run unidirectional bandwidth tests (default: bidirectional)
        -e|--mapper                   shell script that maps local MPI rank to a core and one or more HCAs
                                      e.g. for testing machines with multiple HCAs, where each HCA needs to be
tested
        -g|--gpu                      Run GPU lat/bw/neighbor tests
        -G|--gpudirect                Run GPU tests with GPU-Direct.
        -w|--rdma_write               Use RDMA-write to pass data to the remote host.
        -o|--rdma_read                Use RDMA-read to access data from the remote host.
        -P|--performance              Set CPU scaling governor to 'performance'. Set back to 'powersave' after
execution
        -a|--output                   Generate zip of heatmaps and tgz of JSON files from output. Overrides -k.
        output options:
                -l|--normalize        Normalize latency results                        default: false
                -C|--clean            Erase output cache directory                     default: false
        -x|--exe_opt                  Options for clusterkit.
        -i|--mpi_opt                  Options for mpirun.


    To pass additional MPI options, use the mpi_opt environment variable.
    To pass additional options to the clusterkit executable, use the ext_opt environment variable.

        Examples:
               % ./clusterkit.sh --ssh --hostfile hostfile.txt

               % ./clusterkit.sh --hca_list "mlx5_0:1,mlx5_2:1" --hostfile hostfile.txt

               % exe_opt="--gpudirect     "             ./clusterkit.sh --hca_list "mlx5_0:1,mlx5_2:1" --hostfile
hostfile.txt

               % mpi_opt="-x UCX_RNDV_SCHEME=get_zcopy" ./clusterkit.sh --hca_list "mlx5_0:1,mlx5_2:1" --hostfile
hostfile.txt
```

10 NCCL-RDMA-SHARP Plugins

NCCL-RDMA-SHARP plugins enable RDMA and switch-based collectives (SHARP) with NVIDIA's NCCL library.

10.1 Overview

This plugin replaces the default NCCL internal inter-node communication with RDMA-based transports. It implements both Point-to-Point transport(Net) (IB verbs (default) and UCX), and Collective transport(CollNet) (including SHARP Collective transport).

10.2 NCCL UCX Plugin

NCCL UCX plugin (if enabled) replaces the default NCCL verbs-based inter-node communication routines with UCX-based communication routines.

10.2.1 Running NCCL UCX Plugin

➤ *To use NCCL UCX plugin*:

1. For NCCL to detect the network plugin, make sure to add `plugin_install_dir` to the library search path environment variable, as shown below.

   ```
   # libnccl_net.so is in <plugin_install_dir>/lib
   $ export LD_LIBRARY_PATH=<plugin_install_dir>/lib:$LD_LIBRARY_PATH
   $ <run command>
   ```

2. Enable UCX plugin by defining NCCL_PLUGIN_P2P=ucx environment variable.

   ```
   $ export NCCL_PLUGIN_P2P=ucx
   $ <run command>
   ```

10.2.2 Performance Tuning

To achieve the ultimate performance, various UCX parameters can be used depending on the server's hardware configuration.

10.2.2.1 Example

The below is an example of a hardware configuration where the GPU and the NIC share the same PCIe switch. In such a scenario, GPU Direct RDMA gives the best possible performance.

➤ *To use GPU Direct RDMA for all message sizes in UCX*:

Define the following environment variables as shown.

```
$ export NCCL_UCX_RNDV_THRESH=0
$ export NCCL_UCX_RNDV_SCHEME=get_zcopy
```

```
$ <run command>
```

Note that for servers with multiple NICs available, you need to define the following additional variable.

```
$ export NCCL_UCX_TLS=dc,cuda_copy,cuda_ipc
$ <run command>
```

By default, NCCL is built as a static library to enable portability. In such a case, you may experience plugin-related wrong memory type detection and plugin program failures. In order to avoid this, explicitly disable memory type cache feature in UCX by defining the `UCX_MEMTYPE_CACHE` environment variable as follows.

```
$ export UCX_MEMTYPE_CACHE=n
$ <run command>
```

10.2.2.2 NCCL Tests Benchmark Example

NCCL tests can be used for NCCL-UCX performance benchmarking (visit https://github.com/nvidia/nccl-tests to run the benchmark).

Example:

```
mpirun \
    -np 2 \
    --bind-to socket \
    -x LD_LIBRARY_PATH \
    -x NCCL_UCX_TLS=rc_x,cuda_copy \
    -x NCCL_UCX_RNDV_THRESH=0 \
    -x UCX_MEMTYPE_CACHE=n \
    -x NCCL_COLLNET_ENABLE=0 \
    -x NCCL_PLUGIN_P2P=ucx \
    -x NCCL_DEBUG=info \
    -x NCCL_DEBUG_SUBSYS=NET \
    -x NCCL_IB_HCA=mlx5_0:1 \
    $NCCL_TEST_HOME/build/all_reduce_perf -b 128 -e 128M -f 2 -g 1 -n 50 -w 100 -p 0 -z 0 -t 1 -c 1


# nThread 1 nGpus 1 minBytes 128 maxBytes 134217728 step: 2(factor) warmup iters: 100 iters: 50 validation: 1
#
# Using devices
#   Rank  0 Pid   7198 on  host1 device  0 [0x06] Tesla V100-SXM2-32GB
#   Rank  1 Pid   4890 on  host2 device  0 [0x06] Tesla V100-SXM2-32GB
host1:7198:7198 [0] NCCL INFO NET/IB : Using [0]mlx5_0:1/IB ; OOB ib0:1.1.21.3<0>
NCCL version 2.6.0a0+cuda10.1
host2:4890:4890 [0] NCCL INFO NET/IB : Using [0]mlx5_0:1/IB ; OOB ib0:1.1.21.4<0>
host1:7198:7226 [0] NCCL INFO Thread mode multi is not supported
host1:7198:7226 [0] NCCL INFO Worker address length: 55
host2:4890:4920 [0] NCCL INFO Thread mode multi is not supported
host2:4890:4920 [0] NCCL INFO Worker address length: 55
host2:4890:4920 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host2:4890:4920 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host1:7198:7226 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host1:7198:7226 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host1:7198:7226 [0] NCCL INFO NCCL_COLLNET_ENABLE set by environment to 0.
host1:7198:7226 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host1:7198:7226 [0] NCCL INFO Ring 00 : 1[6000] -> 0[6000] [receive] via NET/UCX/0/GDRDMA
host2:4890:4920 [0] NCCL INFO NCCL_COLLNET_ENABLE set by environment to 0.
host2:4890:4920 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host2:4890:4920 [0] NCCL INFO Ring 00 : 0[6000] -> 1[6000] [receive] via NET/UCX/0/GDRDMA
host1:7198:7226 [0] NCCL INFO Thread mode multi is not supported
host1:7198:7226 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 1
host1:7198:7226 [0] NCCL INFO Ring 00 : 0[6000] -> 1[6000] [send] via NET/UCX/0/GDRDMA
host2:4890:4920 [0] NCCL INFO Thread mode multi is not supported
host1:7198:7226 [0] NCCL INFO Worker address length: 55
host2:4890:4920 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 1
host2:4890:4920 [0] NCCL INFO Ring 00 : 1[6000] -> 0[6000] [send] via NET/UCX/0/GDRDMA
host2:4890:4920 [0] NCCL INFO Worker address length: 55
host2:4890:4920 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host2:4890:4920 [0] NCCL INFO Ring 01 : 0[6000] -> 1[6000] [receive] via NET/UCX/0/GDRDMA
host2:4890:4920 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 1
host2:4890:4920 [0] NCCL INFO Ring 01 : 1[6000] -> 0[6000] [send] via NET/UCX/0/GDRDMA
```

```
host1:7198:7226 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 0
host1:7198:7226 [0] NCCL INFO Ring 01 : 1[6000] -> 0[6000] [receive] via NET/UCX/0/GDRDMA
host2:4890:4920 [0] NCCL INFO Worker address length: 55
host1:7198:7226 [0] NCCL INFO GPU Direct RDMA Enabled for GPU 6000 / HCA 0 (distance 2 <= 3), read 1
host1:7198:7226 [0] NCCL INFO Ring 01 : 0[6000] -> 1[6000] [send] via NET/UCX/0/GDRDMA
host1:7198:7226 [0] NCCL INFO Worker address length: 55
```

10.3 NCCL SHARP Plugin

The following environment variables enable the SHARP aggregation with NCCL when using the plugin.

```
NCCL_COLLNET_ENABLE=1
NCCL_ALGO=CollNet
```

NVIDIA switches allow a limited number of streaming aggregation flows (maximum: 2). On systems with multiple GPUs and multiple HCAs, NCCL creates an aggregation streaming flow (NCCL Ring/Channel) per HCA rail. It is required to build the cluster topology in such a way that leaf level switches are connected to the same HCA rail from each server.

10.3.1 NCCL Test Benchmark Example

The sanity performance of the setup can be verified with NCCL tests. Please refer to NCCL tests here: https://github.com/NVIDIA/nccl-tests.

```
mpirun  -np 1024  -map-by ppr:8:node  -x  NCCL_COLLNET_ENABLE=1 -x NCCL_ALGO=CollNet ./nccl-tests/build/
all_reduce_perf -b 4 -e 2G -f 2 -g 1 -w 50 -n 50

           4             1   float     sum   44.53    0.00    0.00  3e-05    44.21    0.00    0.00  3e-05
           8             2   float     sum   45.42    0.00    0.00  3e-05    45.85    0.00    0.00  3e-05
          16             4   float     sum   46.34    0.00    0.00  3e-05    45.84    0.00    0.00  2e-05
          32             8   float     sum   46.20    0.00    0.00  2e-05    46.56    0.00    0.00  2e-05
          64            16   float     sum   46.00    0.00    0.00  2e-05    48.33    0.00    0.00  2e-05
         128            32   float     sum   48.77    0.00    0.01  2e-05    47.23    0.00    0.01  2e-05
         256            64   float     sum   47.88    0.01    0.01  2e-05    47.85    0.01    0.01  2e-05
         512           128   float     sum   51.44    0.01    0.02  3e-05    48.66    0.01    0.02  3e-05
        1024           256   float     sum   51.27    0.02    0.04  4e-05    51.78    0.02    0.04  4e-05
        2048           512   float     sum   57.93    0.04    0.07  4e-05    56.45    0.04    0.07  4e-05
        4096          1024   float     sum   57.32    0.07    0.14  4e-05    93.51    0.04    0.09  4e-05
        8192          2048   float     sum   106.4    0.08    0.15  4e-05    59.70    0.14    0.27  4e-05
       16384          4096   float     sum   103.0    0.16    0.32  4e-05    58.23    0.28    0.56  4e-05
       32768          8192   float     sum   74.85    0.44    0.87  4e-05    137.8    0.24    0.48  4e-05
       65536         16384   float     sum   96.71    0.68    1.35  4e-05    92.89    0.71    1.41  4e-05
      131072         32768   float     sum   115.6    1.13    2.27  4e-05    120.7    1.09    2.17  4e-05
      262144         65536   float     sum   197.7    1.33    2.65  4e-05    167.6    1.56    3.13  4e-05
      524288        131072   float     sum   222.7    2.35    4.70  4e-05    239.2    2.19    4.38  4e-05
     1048576        262144   float     sum   280.9    3.73    7.46  4e-05    197.7    5.30   10.60  4e-05
     2097152        524288   float     sum   218.0    9.62   19.22  4e-05    213.9    9.81   19.59  4e-05
     4194304       1048576   float     sum   257.6   16.28   32.53  4e-05    254.7   16.47   32.90  4e-05
     8388608       2097152   float     sum   354.3   23.68   47.31  4e-05    523.5   16.02   32.02  4e-05
    16777216       4194304   float     sum   505.9   33.16   66.26  4e-05    484.1   34.66   69.24  4e-05
    33554432       8388608   float     sum   639.2   52.50  104.89  4e-05    678.6   49.45   98.80  4e-05
    67108864      16777216   float     sum  1358.2   49.41   98.72  4e-05   1048.6   64.00  127.87  4e-05
   134217728      33554432   float     sum  1737.2   77.26  154.37  4e-05   1777.6   75.51  150.86  4e-05
   268435456      67108864   float     sum  4359.5   61.58  123.03  4e-05   4262.3   62.98  125.83  4e-05
   536870912     134217728   float     sum  5619.7   95.53  190.88  4e-05   5699.0   94.20  188.22  4e-05
  1073741824     268435456   float     sum   12169   88.23  176.30  4e-05    11508   93.30  186.42  4e-05
  2147483648     536870912   float     sum   22618   94.94  189.70  4e-05    21814   98.44  196.70  4e-05
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 41.2497
```

11 Common Abbreviations

11.1 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

12 User Manual Revision History

Revision	Date	Section	Change
Rev 2.17.1	December 12, 2023	No changes were made to this version.	N/A
Rev 2.17	November 5, 2023	No changes were made to this version.	N/A
Rev 2.16	August 10, 2023	TL/UCP Special Service Worker	Updated
Rev 2.15	May 04, 2023	No changes were made to this version.	N/A
Rev 2.12	July 31, 2022	No changes were made to this version.	N/A
Rev 2.11	May 4, 2022	Unified Collective Communication (UCC)	New section
		IB Router	Removed section
		Important Note	Updated
		Configuring UCX with XPMEM	Updated
		Tuning UCX Settings	Updated
		Adaptive Routing	Updated
		Multi-Rail	Updated
		PCIe Relaxed Ordering Support	Updated
		ucx_perftest	Updated client-side example
		Running ClusterKit via Script	Updated example
Rev 2.10	December 5, 2021	OpenSHMEM MCA Parameters for shmem_quiet, shmem_fence and shmem_barrier_all	New section

13 Release Notes History

13.1 Release Notes Change Log History

13.1.1 HPC-X Toolkit Change Log History

Category	Change
Rev 2.17.0	
HPC-X Content	Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release. • NVIDIA SHARP v3.5.0 • NCCL v2.x • UCX v1.16 • ClusterKit v1.11 • nccl-rdma-sharp-plugin v2.5 Added the following Supported Platforms and OSs: • Debian 10.x • Debian 11.x
Supported Cards	Added support for GH100.
Known Issues	See Known Issues.
Rev 2.16.2	
HPC-X Content	Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release. • NVIDIA SHARP v3.4.1 • UCC v1.3 • ClusterKit v1.10 • nccl-rdma-sharp-plugin v2.4 • NCCL v2.18 • XPMEM v2.7
Supported Cards	All cards up to BlueField-3 and ConnectX-7.
Bug Fixes	See Bug Fixes in this Version.
Rev 2.16	
HPC-X Content	Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release. • NVIDIA SHARP v3.4 • UCC v1.3 • ClusterKit v1.10 • nccl-rdma-sharp-plugin v2.4 • NCCL v2.18 • XPMEM v2.7
Supported Cards	Added support for BlueField-3 cards.
Bug Fixes	See Bug Fixes.
Rev 2.15	

HPC-X Content	Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release. • NVIDIA SHARP v3.3 • UCX v1.15 • ClusterKit v1.9 • nccl-rdma-sharp-plugin v2.3 • GDRCopy v2.3 • NCCL v2.17.1-1 • CUDA v12.1
Bug Fixes	See Bug Fixes.
Rev 2.14	
TL/UCP Special Service Worker	Added support for having a separate UCX UCP worker use UCC service collectives. For further information, please see TL/UCP Special Service Worker section.
Data Type Support in CUDA Executor Component (EC)	Added out-of-box support for all datatypes and reduction operations for UCC collectives for GPUs. For further information, please see Data Type Support in CUDA Executor Component section.
EC/CUDA One-shot Kernel with Cooperative Launch	Added support for using a single CUDA kernel for CUDA operations in UCC GPU collectives. For further information, please see EC/CUDA One-shot Kernel with Cooperative Launch section.
Out-Of-Box Native GPU Allreduce	Added support for the UCC library to detect the NVIDIA NVLink topology and select the best GPU-based algorithms for supported collectives (Allgather/v, Reducescatter/v). For further information, please see Out-Of-Box Native GPU Allreduce section.
Bug Fixes	See Bug Fixes.
Rev 2.13.1 LTS	
Operating System	Added support for Ubuntu v20.04 and v20.10.
Rev 2.13	
HPC-X Content	Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release. • NVIDIA SHARP v3.1 • HCOLL v4.8 • UCC v1.2 • ClusterKit v1.8 • nccl-rdma-sharp-plugin v2.2
NCCL-RDMA-SHARP-PLUGIN	Added support for NCCL plugin API v5.
SHARP	Added support for SHARP on NDR.
Bug Fixes	See Bug Fixes section.
Rev 2.12	
UCX	Added a method to set RoCE ECE value from UCX configuration. For example: UCX_IB_ECE=auto will use maximal ECE value, and UCX_IB_ECE= will use a specific numeric ECE value.
HPC-X Content	Updated the version of the UCX communication library to v1.14.
Rev 2.11	

Adapter Cards	Added support NVIDIA ConnectX-7 adapter card with with 400 Gb/s speed.
SHARPD	sharpd daemon process has been removed. sharpd-related activity is now performed from the user application process
HPC-X Content	Updated the versions of the following communication libraries. • UCX version 1.13 • ClusterKit 1.6
	Added support for UCC, a collective communication operations API and library in HPC-X. UCC is now part of the HPC-X package. For further information on UCC, pleased see Unified Collective Communication (UCC) section.
Rev 2.10	
UCX	Added support for atomics on GPU memory target
OpenSHMEM	Added support for reducing memory overhead on scale
Rev 2.9	
UCX Configuration File	The UCX configuration file enables the user to apply configuration variables set by the user in the /etc/ucx/ucx.conf file. For further information see UCX Configuration File.
Instrumentation and Monitoring FUSE-based Tool	This new functionality enables the user to analyze UCX-based applications in runtime. The tool is based on Filesystem in Userspace (FUSE) interface. If the feature is enabled, a directory for each process using UCX will be created in /tmp/ucx. For further information see Instrumentation and Monitoring FUSE-based Tool.
OS Architecture	HPC-X v1.9 onwards will no longer support PPC architecture in its releases.
Bug Fixes	Bug Fixes in this Version
Rev 2.8	
HPC-X Content	Updated the following communication libraries and acceleration packages versions: • Open MPI version 4.1.x • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 2.4.x • HCOLL version 4.7 • UCX version 1.10 • ClusterKit version 1.3 • nccl-rdma-sharp-plugin version 2.1
UCX	Added support for Multi-interface for cloud (client-server) applications.
	Added support for using Adaptive-Routing (out-of-order) on an SL that supports it.
	Added support for UCP Active-Messages API with Rendezvous.
	Added support for Keepalive functionality on the UCT layer.
	Performed several error handling enhancements.
	Added support for GPU-NIC locality discovery.
NCCL-RDMA-SHARP-PLUGIN	Added support for NCCL Plugin API v4.
	Added support for PCIe Relaxed Ordering.
	Added support for Adaptive Routing.
Rev 2.7	

UCX	Added a new request API. For further information on this request API, please refer to UCX API documentation.
	Added support for PCIe Relaxed Ordering.
	Added out-of-box support for RoCE LAG.
	Added Flow Control support for RDMA Read operations.
	AMD Rome optimizations: Optimized IB connection establishment procedures to reduce system noise.
Rev 2.6	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 2.1.0 • HCOLL version 4.5 • UCX version 1.9
UCX	Added support in UCX for communication between containers configured to share the memory namespaces.
	Added strided Receive queue support for hardware tag matching.
	Made the following performance improvements on AMD EPYC servers. • 8-16 KB message: Improved latency by up to 6.4%, bandwidth by up to 20%, and bidirectional bandwidth by up to 96% • IMB/multiPingPong and osu_mbw_mr for messages up to 32B on full ppn on MLNX_OFED 5.0. **Note**: To enjoy this performance optimization, make sure to enable hardware tag-matching by setting `UCX_RC_TM_ENABLE=y`
	Added support for multithreaded memory region in Open SHMEM (OSHMEM) applications to improve performance in job startup and teardown latencies. The multithreaded MR enables a more efficient use of the CPU resource during registration of memory regions larger than 4GB.
Cuda	Removed Cuda support in SLES 11 and RHEL 6 OSs.
Rev 2.5	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 2.0 • HCOLL version 4.4 • UCX version 1.7
	Removed CUDA init script (hpcx-init-cuda.sh), and environmental module (modules/hpcx-cuda) from HPC-X. Up until HPC-X v2.4, these files used to point to the default files hpcx-init.sh and modules/hpcx. Now, these CUDA files no longer exist, and users can only use the default init script and environmental module for enabling CUDA support.
CUDA	Unified Vanilla and CUDA environments. CUDA v10.0 is supported out of the box with standard init script or environmental module. **Note**: HPC-X is compiled against CUDA version 10.0, which does not support GCC versions newer than v8. Therefore, HPC-X built on systems with GCC versions above v8 will not have CUDA support.
UCX	Made performance optimizations.
	Added full support for rdma-core.
	Added support for CUDA v10.1.

Rev 2.4	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.8 • HCOLL version 4.3 • UCX version 1.6
	Removed `rc.local_mellanox` script. HPC-X became more stable and this script is no longer required.
CUDA	Unified Vanilla and CUDA environments. CUDA v9 is supported out of the box with standard init script or environmental module. **Note**: HPC-X is compiled against CUDA version 9, which does not support GCC versions newer than v7. Therefore, HPC-X built on systems with GCC versions above v7 will not have CUDA support.
UCX	Enabled HDR, SocketDirect and MultiRail features out-of-box.
	UCX Random DCI is now at GA level.
	Implemented a number of job startup optimizations.
	Added support from PCIe atomic operations feature.
HCOLL	Added support for performing floating point 16 bit operations for machine learning scenarios.
OpenMPI	Added multi threading support to OpenMPI OSC UCX.
General	HPC-X is now available through the EasyBuild framework: https://easybuild.readthedocs.io/en/latest/
Rev 2.3	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • Open MPI version 4.0.x • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.7.2 • HCOLL version 4.2 • UCX version 1.5 • OpenSHMEM version 1.4
UCX	UCX is now compiled without JAVA bindings.
	Added support for running UCX over rdma-core, for DC transport and direct verbs.
	Emulation layer: Added the ability to run UCX over software emulation of remote memory access and atomic operations. This provides full support of SHMEM and MPI-RMA over shared memory, TCP, and older RDMA hardware, such as ConnectX-3 HCA.
HCOLL	HCOLL and NVIDIA SHARP are now compiled with CUDA support.
	Added support for CUDA buffers over SRA allreduce algorithm.
MXM	Removed support for MXM library.
OpenMPI	Added the following configuration options to OMPI: • `--with-libevent=internal` • `--enable-mpi1-compatibility`
	Updated the configuration file platform/mellanox/optimized config in OMPI upstream by removing BTL OpenIB and UCT support and removing links to MXM/FCA usage.
	Removed PMI2 support.

Rev 2.2	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.7 • HCOLL version 4.1 • UCX version 1.4
	Added support for Singularity containerization. For further information, please refer to HPC-X User Manual.
	“osc ucx” is no longer the default one-sided-component in OpenMPI.
	Removed KNEM library from HPC-X package. UCX will use the KNEM available in MLNX_OFED.
MXM Support	Open MPI and HCOLL are not compiled with MXM anymore. Both are compiled with UCX only and use it by default.
UCX	Added support for the following UCX features: • New API for establishing client-server connection. • Out-of-box support for Memory In Chip (MEMIC) on ConnectX-5 HCAs.
HPC-X Setup	Added support for HPC-X to work on Huawei ARM architecture.
HCOLL	Improved performance by utilizing zero-copy messaging for MPI Bcast.
Rev 2.1	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • Open MPI version 3.1.x • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.5 • HCOLL version 4.0 • MXM version 3.7 • UCX version 1.3 • OpenSHMEM v1.3 specification compliant
UCX	• UCX is now the default pml layer for Open MPI, default spml layer for OpenSHMEM, and default OSC component for MPI RMA. • Added the following UCX features: • Added support for GPU memory in UCX communication libraries • Added support for Multi-Rail protocol
MXM	The UD_RNDV_ZCOPY parameter is set to ‘no’ by default. This means that the zcopy mechanism for the UD transport is disabled when using the Rendezvous protocol.
HCOLL	• UCX is now the default p2p transport in HCOLL • Improved multi-threaded performance • Improved shared memory performance • Added support for NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v1.5 • Added support for NVIDIA SHARP software multi-channel/multi-rail capable algorithms • Improved Allreduce large message algorithm • Improved AlltoAll algorithm
Profiling IB verbs API (ibprof)	Removed ibprof tool from HPC-X toolkit.
UPC	Removed UPC from HPC-X toolkit.
Rev 2.0	

HPC-X Content	Updated the following communications libraries and acceleration packages versions: • OpenMPI version 3.0.0 • Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.4 • HCOLL version 3.9 • UCX version 1.3
UCX	• UCX is now at GA level. • Added the following UCX features: • **[ConnectX-5 only]** Added support for hardware Tag Matching with DC transport. • **[ConnectX-5 only]** Added support for Out-of-order RDMA RC and DC to support adaptive routing with true RDMA. • Added UCX datatypes - community approved datatype support. • Added UCX support to Inbox RHEL. • Added GPU Direct RDMA support. • Hardware Tag Matching (See section *Hardware Tag Matching* in the User Manual) • SR-IOV Support (See section *SR-IOV Support* in the User Manual) • Adaptive Routing (AR) (See section *Adaptive Routing* in the User Manual) • Error Handling (See section *Error Handling* in the User Manual)
HCOLL	• Added support for Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v1.4 • Added support for NCCL on-host GPU based collectives. • Added support for Hierarchical GPU based allreduce using NCCL for scale-in and MXM/UCX for scale-out. • Improved shared memory performance for allreduce, barrier, and broadcast. Targeting high thread count systems, e.g. Power9. • Improved large message allreduce (multi-radix, zero-copy fragmentation, CPU vectorization.) • Added new and improved AlltoAllv algorithm - hybrid logarithmic pair-wise exchange. • Added support for on-demand HCOLL memory. Improves HCOLL's memory footprint on high thread count system e.g. Power9. • Added a high performance multithreaded implementation to support MPI_THREAD_MULTIPLE applications. Designed specifically for high thread count systems, e.g. Power9. • HCOLL startup improvements.
Open MPI / OpenSHMEM	• Added support for Open MPI 3.0.0. • Added support for xpmem kernel module. • Added a high performance implementation of shmem_ptr() with UCX SPML. • Added a UCX allocator. The UCX allocator optimizes intra-node communication by allowing direct access to memories of processes on the same node. The UCX allocator can only be used with the UCX SPML. • Added a UCX one-sided component to support MPI RMA operations.
Rev 1.9.7	
Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)	Bug Fixes, see Section 4, “Bug Fixes History”, on page 11
Rev 1.9	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • OpenMPI version 2.1.2a1 • Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.3.1 • HCOLL version 3.8.1652 • MXM version 3.6.3103 • UCX version 1.2.2947

UCX	Point-to-point communication API, with tag matching, remote memory access, and atomic operations. This can be used to implement MPI, PGAS, and Big Data libraries and applications- IB transport
	A cleaner API with lower software overhead which provides better performance especially for small messages.
	Support for multitude of InifiniBand transports and NVIDIA offloads to optimize data transfer performance: • RDMA • DC • Out-of-order • HW tag matching offload • Registration cache • ODP
	Shared memory communications for optimal intra-node data transfer: • SysV • posix • knem • CMA • xpmem
MXM	Enabled Adaptive Routing for all the transport layers (UD/RC/DC).
	Memory registration optimization.
Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)	Improved the Out-of-the-box performance of Scalable Hierarchical Aggregation and Reduction Protocol (SHARP).
Shared memory	Improved the intranode performance of allreduce and barrier.
Configuration	Changed many default parameter setting in order to achieve best out-of-the-box experience for several applications including - CP2K, miniDFT, VASP, DL-POLY, Amber, Fluent, GAMES-UK, and LS-DYNA.
FCA	As of HPC-X v1.9, FCA v2.5 is no longer included in the HPC-X package.
	Improved AlltoAllv algorithm.
	Improved large data allreduce.
	Improved UCX BCOL.
OS architecture	Added support for ARM architecture.
Rev 1.8.2	
MXM	Updated MXM version to 3.6.2098 which includes memory registration optimization.
Rev 1.8	
Cross Channel (CC)	Added Cross Channel (CC) AlltoAllv
	Added CC zcpy Ring Bcas
Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)	Added Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) non-blocking collectives
Shared memory POWER	Added shared memory POWER optimizations for allreduce
	Added shared memory POWER optimizations for Barrier

Mixed data types	Added support for mixed data types
Non-contiguous Bcast	Added support for non-contiguous Bcast with UMR or SGE in CC
UMR	Added UMR support in CC bcol
Unified Communication - X Framework (UCX)	A new acceleration library, integrated into the Open MPI (as a pml layer) and available as part of HPC-X. It is an open source communication library designed to achieve the highest performance for HPC applications.
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • HCOLL updated to v3.7. Open MPI updated to v2.10
FCA	FCA 2.x is no longer the default FCA used in HPC-X. As of HPC-X v1.8, FCA 3.x (HCOLL) is the default FCA used and it replaces FCA v2.x.
Bug Fixes	See Section 4, “Bug Fixes History”, on page 11
Rev 1.7	
MXM	Updated MXM version to 3.6
FCA Collective	Added Cross-Channel based Allgather, Bcast, 8-byte Allreduce.
FCA	Added MPI datatype support.
	Added optimizations for PPC platforms.
	Added support for multiple NVIDIA SHARP technology leaders on a single host.
	Added support for collecting NVIDIA SHARP technology usage statistics.
	Exposed cross-channel non-blocking collectives to the MPI level.
Rev 1.6	
MXM v3.5	See Section 5.3, “MXM Change Log History”, on page 23
IB-Router	Allows 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. **Note:** When using 'openib btl', RoCE and IB router are mutually exclusive. The Open MPI inside HPC-X 1.6 is not compiled with ib-router support, therefore it supports RoCE out-of-the-box.
FCA v3.5	See Section 5.2, “FCA Change Log History”, on page 21
Rev 1.5	
HPC-X Content	Updated the following communications libraries and acceleration packages versions: • Open MPI updated to v1.10 • UPC update to 2.22.0 • MXM updated to v3.4.369 • FCA updated to v3.4.799
MXM v3.4.369	See Section 5.3, “MXM Change Log History”, on page 23
FCA v3.4.799	See Section 5.2, “FCA Change Log History”, on page 21
Rev 1.4	
FCA v3.3	See Section 5.2, “FCA Change Log History”, on page 21
MXM v3.4	See Section 5.3, “MXM Change Log History”, on page 23
Rev 1.3	
MLNX_OFED	Added support for OFED Inbox drivers

CPU Architecture	Added support for PPC architecture
LID Mask Control (LMC)	Added support for multiple LIDs usage when the LMC in the fabric is higher than zero. MXM will use multiple LIDs to distribute traffic across multiple links and achieve better resource utilization.
Performance	Performance improvements for all transport layers.
Adaptive Routing	Enhanced support for Adaptive Routing for the UD transport layer. For further information, please refer to the HPC-X User Manual section *“Adaptive Routing for UD Transport”*.
UD zero copy	UD zero copy support on receiver side to achieve better bandwidth utilization and reduce CPU usage.

13.1.2 FCA Change Log History

Category	Change
Rev 3.5	
FCA Collective	Added MPI Allgatherv and MPI reduce
FCA	Added support for NVIDIA SHARP library (including SHARP allreduce, reduce and barrier)
	Enhanced scalability for CORE-Direct based collectives
	Added support for complex data types
Rev 3.4	
General	UCX support
	Communicator caching scheme with eviction: improves jobstart and communicator creation time
Collectives	Collectives: Added Alltoallv and Alltoall small message algorithms.
Rev 3.3	
General	Ported to PowerPC
	Thread safety added
Collectives	Improved large message allreduce algorithm (Enabled by default)
	Beta version of network topology awareness (Enabled by default)
Rev 3.0	

Collectives	Offload collectives communication from MPI process onto NVIDIA interconnect hardware.
	Efficient collectives communication flow optimized to job and topology
MPI collectives	Significantly reduce MPI collectives runtime
MPI-3	Native support for MPI-3
Blocking and Non-blocking collectives	Support for blocking and nonblocking collectives
HCOLL	Supports hierarchical communication algorithms (HCOLL)
Collective algorithm	Supports multiple optimizations within a single collective algorithm
Performance	Increase CPU availability and efficiency for increased application performance
MPI libraries	Seamless integration with MPI libraries and job schedulers
Rev 2.5	
Multicast Group	Added MCG (Multicast Group) cleanup tool
Performance	Performance improvements
Rev 2.2	
Performance	Performance improvements
Dynamic offloading rules	Enabled dynamic offloading rules configuration based on the data type and reduce operations
Mixed MTU	Added support for mixed MTU
Rev 2.1.1	
AMD/Interlagos CPUs	Added support for AMD/Interlagos CPUs
Rev 2.1	
Core-Direct®	Added support for Core-Direct® technology (enables offloading collective operations to the HCA.)
Non-contiguous data layouts	Added support for non-contiguous data layouts
PGI compilers	Added support for PGI compilers

13.1.3 HPC-X™ Open MPI/OpenSHMEM Change Log History

Category	Change
Rev 2.2	
Performance	Added Sandy Bridge performance optimizations.
memheap	Allocated memheap using contiguous memory provided by the HCA.
ptmalloc allocator	Replaced the buddy memheap by the ptmalloc allocator.

multiple pSync arrays	Added the option of using multiple pSync arrays instead of barrier synchronization between collective routines (fcollect, reduction routines)
spml yoda	Optimized small size puts
Performance	Performance optimization
Memory footprint optimizations	Added memory footprint optimizations
Rev 1.8.2	
Acceleration Packages	Added support for new MXM, FCA, HCOLL versions
Job start optimization	Added job start optimization
Performance	Performance improvements

13.2 Bug Fixes History

Internal Reference Number	Issue
3436244	**Description:** On rare occasions, a 'group join' request may reach a timeout.
	Keywords: NDR Switch, SHARP
	Discovered in Version: 2.16
	Fixed in Version: 2.16.2
3479712	**Description:** In virtualized environments, the performance of large messages can drop due to repeated failures to create indirect-atomic key (KSM).
	Keywords: Virtualized Environments; Failure; Indericet-atomic Key; KSM;
	Discovered in Version: 2.15
	Fixed in Version: 2.16
3268964	**Description:** Improved performance in MPI_Bcast on AMD Genoa. **Note**: To make use of these improvements, make sure UCC is explicitly enabled using: `--mca coll_ucc_enable 1 --mca coll_ucc_priority 99 --mca coll ucc,basic,libnbc --mca coll_ucc_cls basic,hier`
	Keywords: MPI_Bcast; AMD Genoa; UCC
	Discovered in Version: 2.14
	Fixed in Version: 2.15
3255925	**Description:** Fixed the issue where mpi_init was creating an internal CUDA context on GPU0, which could have an impact on CUDA applications behavior.
	Keywords: CUDA; MPI
	Discovered in Version: 2.13
	Fixed in Version: 2.14

Internal Reference Number	Issue
3223214	**Description:** Fixed the issue where shmem_ulong_wait_until() unsigned comparison was not working as expected.
	Keywords: SHMEM
	Discovered in Version: 2.13
	Fixed in Version: 2.14
3261844	**Description:** Fixed the issue of when TCP transport was used on RDMA-capable setup, this led to lower performance and occasional hangs during mpi_finalize.
	Keywords: TCP; RDMA; MPI; performance
	Discovered in Version: 2.13
	Fixed in Version: 2.13.1 LTS
3139906	**Description:** Port counters were not updated for UCX traffic when creating QP with DevX.
	Keywords: UCX; QP; DevX
	Discovered in Version: 2.13
	Fixed in Version: 2.13.1 LTS
3084053	**Description:** Fixed the issue where performance of some applications was lower compared with HPC-X v2.10 and earlier.
	Keywords: Performance
	Discovered in Version: 2.12
	Fixed in Version: 2.13
3163697	**Description:** Fixed the issue of when the client application used more than 1024 file descriptors (range limit defined by FD_SETSIZE), libsharp was prevented from using any more file descriptors. Using poll() instead of select() enables using the full range of allowed file descriptors by Linux.
	Keywords: File descriptor; libsharp; HCOLL; HPC-X
	Discovered in Version: 2.12
	Fixed in Version: 2.13
3208615	**Description:** Fixed Data Integrity failure in Broadcast when using sparse subarray data type in OMPI with hcoll library by using the TRUE extent of the datatype, which includes any additional padding the datatype may require.
	Keywords: OMPI; HCOLL; data integrity
	Discovered in Version: 1.12
	Fixed in Version: 2.13
4549	**Description:** Fixed the issue where UCX may have failed to compile with Clang compiler version 9 if `--dynamic-list-data` flag was used in the compilation. (Github issue: https://github.com/openucx/ucx/issues/4549)
	Keywords: Clang compiler, UCX
	Discovered in Version: 2.6 (UCX 1.8)

Internal Reference Number	Issue
	Fixed in Version: 2.11 (UCX 1.13)
-	**Description:** DevX does not work on architectures without "Write combining" support, such as some flavors of ARM, prompting the following error message. `UCX ERROR mlx5dv_devx_alloc_uar() failed: Operation not supported`
	Keywords: DevX, UCX, ARM
	Discovered in Version: 2.8 (UCX 1.10)
	Fixed in Version: 2.9 (UCX 1.11)
-	**Description:** NVIDIA SHARP library is not available in HPC-X for the Community OFED and Inbox OFED.
	Keywords: NVIDIA SHARP library
	Discovered in Version: 2.0
	Fixed in Version: 2.9 (UCX 1.11)
2190337	**Description:** Fixed the issue where errors from the UCX TCP transport about refused connection may have appeared.
	Keywords: UCX_TLS, UCX, TCP
	Discovered in Version: 2.7 (UCX 1.9)
	Fixed in Version: 2.8 (UCX 1.10)
2131893	**Description**: Fixed the issue where OpenSHMEM or MPI applications may have failed with the following error: `“Fatal: endpoint reconfiguration not supported yet”` This could happen when running in heterogeneous environment, such as when different nodes in the job had different types of HCAs or PCI atomics configuration.
	Keywords: OpenSHMEM, UCX, MPI
	Discovered in Version: 2.7 (UCX 1.9)
	Fixed in Version: 2.8 (UCX 1.10)
2084450	**Description:** Fixed the issue where the osu_ialltoallw and osu_iallgather benchmarks may have not performed well over RoCE with the ud_x transport starting messages of 8192 bytes.
	Keywords: osu_ialltoallwת osu_iallgather, ud_x transport, RoCE, UCX
	Discovered in Version: 2.6 (UCX 1.8)
	Fixed in Version: 2.8 (UCX 1.10)

Internal Reference Number	Issue
1886580	**Description:** Fixed the issue where the below error messages might have been received when running OMPI with 'direct modex', i.e. when the following command line parameters were used: `-mca pmix_base_async_modex 1 -mca mpi_add_procs_cutoff 0 -mca pmix_base_collect_data 0` Error messages: • `PMIX ERROR: NOT-FOUND in file server/pmix_server_get.c at line 751` • `PMIX ERROR: NOT-FOUND in file client/pmix_client_get.c at line 334`
	Keywords: OMPI, pmix, direct modex, full modex
	Discovered in Version: 2.5 (OpenMPI 4.0.x)
	Fixed in Version: 2.7 (OpenMPI 4.0.x)
4710	**Description:** Fixed the issue of when using UCX with XPMEM module on Kernels 4.10 and above, there might have been a "Bus error" due to an issue in the XPMEM driver. (Github issue: https://github.com/openucx/ucx/issues/4710)
	Keywords: UCX, XPMEM
	Discovered in Version: 2.6 (UCX 1.8)
	Fixed in Version: 2.7 (UCX 1.9)

2096036	**Description:** Fixed the issue where the verifier test may have failed with the following error when using the ud_x transport:
	`ib_mlx5_log.c:139 Local QP operation on mlx5_0:1/IB (synd 0x2 vend 0x68 hw_synd 0/66)`
	`ib_mlx5_log.c:139 UD QP 0x37161 wqe[368]: SEND --- [rqpn 0x36a01 rlid 93] [inl len 16]`

	Keywords: ud_x transport, UCX
	Discovered in Version: 2.6 (UCX 1.8)
	Fixed in Version: 2.7 (UCX 1.9)
2095618	**Description:** Fixed the issue where the host may have run out of memory when enabling Hardware Tag-Matching.
	Keywords: Hardware Tag-Matching, UCX
	Discovered in Version: 2.6 (UCX 1.8)
	Fixed in Version: 2.7 (UCX 1.9)
3758	**Description:** Fixed the issue of when running UCX with TCP transport on more than 16 hosts with full PPN (processes per node), the following error message might have appeared. `sock.c:228  UCX  ERROR recv(fd=1377) failed: 104` (Github issue: https://github.com/openucx/ucx/issues/3758)
	Keywords: TCP, UCX, backlog
	Discovered in Version: 2.5 (UCX 1.7)
	Fixed in Version: 2.6 (UCX 1.8)

Internal Reference Number	Issue
1582208	**Description:** Fixed the issue where sending data over multiple SHMEM contexts may lead to memory corruption or segmentation fault.
	Keywords: Open SHMEM, segmentation fault
	Discovered in Version: 2.3 (Open MPI v4.0.x, OpenSHMEM v1.4)
	Fixed in Version: 2.5 (Open MPI v4.0.x, OpenSHMEM v1.4)
2934	**Description:** Fixed the issue where OpenMPI and OpenSHMEM applications may hang with DC transport. (Github issue: https://github.com/openucx/ucx/issues/2934)
	Keywords: UCX, Open MPI, DC
	Discovered in Version: 2.3 (Open MPI v4.0.x, OpenSHMEM v1.4)
	Fixed in Version: 2.5 (Open MPI v4.0.x, OpenSHMEM v1.4)
1307243	**Description:** Fixed the issue where one-sided tests may fail with a segmentation fault.
	Keywords: OSC UCX, Open MPI, one-sided
	Discovered in Version: 2.1 (Open MPI 3.1.x)
	Fixed in Version: 2.5 (Open MPI 4.0.x)
-	**Description:** Fixed the issue where OpenSHMEM atomic operations AND/OR/XOR for datatypes int32/int64/uint32/uint64 were not implemented, which might have caused build failures.
	Keywords: OpenSHMEM atomic, Open MPI
	Discovered in Version: 2.3 (Open MPI v4.0.x, OpenSHMEM v1.4)
	Fixed in Version: 2.4 (Open MPI v4.0.x, OpenSHMEM v1.4)
2226	**Description:** Fixed the issue where the following assertion may have failed in certain cases: Assertion `ep->rx.ooo_pkts.head_sn == neth->psn' failed (Gihub issue: https://github.com/openucx/ucx/issues/2226)
	Keywords: UCX, assertion
	Discovered in Version: 2.1 (UCX 1.3)
	Fixed in Version: 2.4 (UCX 1.6)
-	**Description:** Fixed the issue where zero-length OpenSHMEM collectives might have failed due to incomplete implementation.
	Keywords: OpenSHMEM atomic, Open MPI
	Discovered in Version: 2.3 (Open MPI v4.0.x, OpenSHMEM v1.4)
	Fixed in Version: 2.4 (Open MPI v4.0.x, OpenSHMEM v1.4)
-	**Description:** Fixed the issue where OSC UCX module was not selected by default on ConnectX-4/ConnectX-5 HCAs.
	Keywords: OSC UCX, one-sided, Open MPI
	Discovered in Version: 2.3 (Open MPI v4.0.x, OpenSHMEM v1.4)
	Fixed in Version: 2.4 (Open MPI v4.0.x, OpenSHMEM v1.4)
-	**Description**: Fixed the issue where using UCX on ARM hosts may result in hangs due to a known issue in Open MPI when running on ARM.

Internal Reference Number	Issue
	Keywords: UCX
	Discovered in Version: 1.3 (Open MPI 1.8.2)
	Fixed in Version: 2.3 (Open MPI 4.0.x)
-	**Description**: MCA options *rmaps_dist_device* and *rmaps_base_mapping_policy* are now functional.
	Keywords: Process binding policy, NUMA/HCA locality
	Discovered in Version: 2.0 (Open MPI 3.0.0)
	Fixed in Version: 2.3 (Open MPI 4.0.x)
2111	**Description**: Fixed the issue of when UCX was used in the multi-threaded mode, it might have taken the *osu_latency_mt* test a long time to be completed. (Github issue: https://github.com/openucx/ucx/issues/2111)
	Keywords: UCX, multi-threaded
	Discovered in Version: 2.1 (UCX 1.3)
	Fixed in Version: 2.3 (UCX 1.5)
2267	**Description**: Fixed the issue where the following error message might have appeared when running at the scale of 256 ranks with the RC transport, when UD is used for wireup only: " *Fatal: send completion with error: Endpoint timeout* ". (Github issue: https://github.com/openucx/ucx/issues/2267)
	Keywords: UCX
	Discovered in Version: 2.1 (UCX 1.3)
	Fixed in Version: 2.3 (UCX 1.5)
2702	**Description**: Fixed the issue of when using the Hardware Tag Matching feature, the following error messages may have been printed: • *"rcache.c:481 UCX WARN failed to register region 0xdec25a0 [0x2b7139ae0020..0x2b7139ae2020]: Input/output error"* • *"ucp_mm.c:105 UCX ERROR failed to register address 0x2b7139ae0020 length 8192 on md[1]=ib/mlx5_0: Input/output error"* • *"ucp_request.c:259 UCX ERROR failed to register user buffer datatype 0x20 address 0x2b7139ae0020 len 8192: Input/output error"* (Github issue: https://github.com/openucx/ucx/issues/2702)
	Keywords: Hardware Tag Matching
	Discovered in Version: 2.2 (UCX 1.4)
	Fixed in Version: 2.3 (UCX 1.5)
2454	**Description**: Fixed the issue where some one-sided benchmarks may have hung when using "osc ucx". For example: osu-micro-benchmarks-5.3.2/osu_get_acc_latency (Latency Test for accumulate with Active/Passive Synchronization). (Github issue: https://github.com/openucx/ucx/issues/2454)

Internal Reference Number	Issue
	Keywords: UCX, one_sided
	Discovered in Version: 2.2 (UCX 1.4)
	Fixed in Version: 2.3 (UCX 1.5)
2670	**Description**: Fixed the issue of when enabling the Hardware Tag Matching feature on a large scale, the following error message may have been printed due to the increased threshold for BCOPY messages: *"mpool.c:177 UCX ERROR Failed to allocate memory pool chunk: Out of memory."* (Github issue: https://github.com/openucx/ucx/issues/2670)
	Keywords: Hardware Tag Matching
	Discovered in Version: 2.2 (UCX 1.4)
	Fixed in Version: 2.3 (UCX 1.5)
1295679	**Description**: Fixed the issue where OpenSHMEM group cache had a default limit of 100 entries, which might have resulted in OpenSHMEM application exiting with the following message: " *group cache overflow on rank xxx: cache_size = 100* ".
	Keywords: OpenSHMEM, Open MPI
	Discovered in Version: 2.1 (Open MPI 3.1.x)
	Fixed in Version: 2.2 (Open MPI 3.1.x)
-	**Description**: Fixed the issue where UCX did not work out-of-the-box with CUDA support.
	Keywords: UCX, CUDA
	Discovered in Version: 2.2 (UCX 1.4)
	Fixed in Version: 2.1 (UCX 1.3)
1926	**Description**: Fixed the issue of when using multiple transports, invalid data was sent out-of-sync with Hardware Tag Matching traffic. (Github issue: https://github.com/openucx/ucx/issues/1926)
	Keywords: Hardware Tag Matching
	Discovered in Version: 2.1 (UCX 1.3)
	Fixed in Version: 2.2 (UCX 1.4)
1949	**Description**: Fixed the issue where Hardware Tag Matching might not have functioned properly with UCX over DC transport. (Github issue: https://github.com/openucx/ucx/issues/1949)
	Keywords: UCX, Hardware Tag Matching, DC transport
	Discovered in Version: 2.0
	Fixed in Version: 2.1
-	**Description**: Fixed job data transfer from SD to libsharp.
	Keywords: NVIDIA SHARP library
	Discovered in Release: 1.9

Internal Reference Number	Issue
	Fixed in Release: 1.9.7
884482	**Description:** Fixed internal HCOLL datatype mapping.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
884508	**Description:** Fixed internal HCOLL datatype lower bound calculation.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
884490	**Description:** Fixed allgather unpacking issues.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
885009	**Description:** Fixed wrong answer in alltoallv.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
882193	**Description:** Fixed mcast group leak in HCOLL.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
-	**Description:** Added IN_PLACE support for alltoall, alltoallv, and allgatherv.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
-	**Description:** Fixed an issue related to multi-threaded MPI_Bcast.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
Salesforce: 316541	**Description:** Fixed a memory barrier issue in MPI_Barrier on Power PPC systems.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
Salesforce: 316547	**Description:** Fixed multi-threaded MPI_COMM_DUP and MPI_COMM_SPLIT hanging issues.

Internal Reference Number	Issue
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
894346	**Description**: Fixed Quantum Espresso hanging issues.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
898283	**Description**: Fixed an issue which caused CP2K applications to hang when HCOLL was enabled.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.7.405
	Fixed in Release: 1.7.406
906155	**Description**: Fixed an issue which caused VASP applications to hang in MPI_Allreduce.
	Keywords: HCOLL, FCA
	Discovered in Release: 1.6
	Fixed in Release: 1.7.406