# NVIDIA HPC-X Software Toolkit Rev 2.50

# Table of Contents

| pdf.download |  |
|---|---|

This version is not intended for GB/B customers. GB/B customers should use the designated release package to ensure full compatibility, qualification, and support alignment.

## 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

## Software Download

Please visit NVIDIA HPC-X

## Document Revision History

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

## Related Documentation

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

# 1 Release Notes

Release Notes Update History

| Revision | Date | Description |
|---|---|---|
| 2.50 | June 2026 | Initial release of this document. |

## 1.1 Changes and New Features

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

| Category | Change Description |
|---|---|
| **HPC-X Content** | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release:<br>• IMB Benchmarks v2021.10<br>• nccl_spcx_plugin v1.4.0<br>• NVIDIA SHARP v3.15<br>• UCX v1.21<br>• UCC v1.9.0 |
| **UCC: TL/UCP Strided Active Set Broadcast** | Added support for broadcast collectives over strided active sets larger than two ranks. This allows for flexible rank subsetting using start, stride, and size parameters. For further information, please see TL/UCP Strided Active Set. |
| **UCC** | Starting with HPC-X v2.50, UCC is the default collective framework for Open MPI collectives. HCOLL is included in HPC-X but is disabled by default. |
| | Improved UCC correctness on NVIDIA Grace Hopper systems by strengthening AArch64 memory ordering and shared-memory collective synchronization. |
| **Open MPI** | Open MPI v5.0.x is now the default MPI implementation in HPC-X. Open MPI v4.1.x is available and can be set using an environment variable - see Installing and Loading HPC-X for further details. |
| **Benchmarks** | **IMB Benchmarks:** Upgraded to v2021.10.<br>• Improved GPU startup by initializing CUDA prior to `MPI_Init`. This aligns with standard CUDA-aware MPI application behavior and prevents late-initialization errors. |

## 1.2 HPC-X General Support

### 1.2.1 HPC-X Requirements

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

| Platform | Versions |
|---|---|
| CUDA | 12.3+ |
| GDRCopy | 2.3+ |
| DOCA-Host | 3.3.0 |
| NCCL | 2.22+ |

| Platform | Versions |
|---|---|
| NVIDIA BlueField-3 | 32.49.1014 |
| NVIDIA ConnectX-5/ConnectX-5 Ex | 16.35.2000 |
| NVIDIA ConnectX-6 | 20.43.1014 |
| NVIDIA ConnectX-6 Dx | 22.49.1014 |
| NVIDIA ConnectX-6 Lx | 26.49.1014 |
| NVIDIA ConnectX-7 | 28.49.1014 |
| NVIDIA ConnectX-8 | 40.49.1014 |
| XPMEM | 2.7 |
| Grace-Hopper - GH200 | N/A |

### 1.2.2 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 |
| Open MPI[1] | 5.0 |
| NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) | 3.15 |
| HCOLL[2] | 4.8 |
| UCX | 1.21 |
| UCC | 1.9.0 |
| ClusterKit[3] | 1.15 |
| nccl-rdma-sharp-plugin[4] | 2.7 |
| nccl_spcx_plugin | 1.4.0 |
| OSU Micro-Benchmarks | 7.4 |
| Intel MPI Benchmarks (IMB) | 2021.10 |

1. Open MPI v5.0.x is now the default MPI implementation in HPC-X.
2. HCOLL is not supported on GB200 and GB300 systems, with no plans for future support.
3. ClusterKit is a multifaceted node assessment tool for high performance clusters.
4. nccl-rdma-sharp plugin enables RDMA and Switch-based collectives (SHARP) with NVIDIA's NCCL library.

### 1.2.3 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
```

### 1.2.4 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 8.x | x86_64, aarch64 |
| RHEL/CentOS/Rocky 9.x | x86_64, aarch64 |
| RHEL/CentOS/Rocky 10.x | x86_64, aarch64 |
| CentOS 8.x Stream | x86_64 |
| CentOS 9.x Stream | x86_64 |
| SLES 15 SP4 | x86_64 |
| SLES 15 SP5 | x86_64 |
| SLES 15 SP6 | x86_64 |
| Ubuntu 22.04 | x86_64, aarch64 |
| Ubuntu 24.04 | x86_64, aarch64 |
| Kylin 10 SP1 | x86_64, aarch64 |
| Kylin 10 SP2 | x86_64, aarch64 |
| Debian 12.x | x86_64 |
| CTyunOS 25.07 | x86_64, aarch64 |

## 1.3 Bug Fixes in this Version

N/A

## 1.4 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 |
|---|---|
| 4803211 | **Description:** Using GPU buffers with the `ireduce_scatter` collective may result in a segmentation fault. |
| | **Workaround:** Use the HPC-X Open MPI v5.0.x stack by setting `HPCX_ENABLE_OMPI5=1` before loading the HPC-X environment/modules. |
| | **Keywords:** UCX; ireduce_scatter; segfault; segmentation fault; GPU |
| | **Discovered in Version:** 2.26 |
| 4693674 | **Description**: Performance degradation may occur when running `MPI_Alltoall` with two processes on systems using ConnectX-8 HCAs. |
| | **Workaround:** Set the environment variable `UCX_CUDA_COPY_BW=9500MBps` to restore performance. |
| | **Keywords:** Performance; MPI: collectives; GPU |
| | **Discovered in Version:** 2.25 |
| 4662365 | **Description**: In certain cases, performance degradation may occur with `MPI_Alltoall`, `MPI_Alltoallv`, `MPI_Bcast`, and `MPI_Allgather` operations. |
| | **Workaround:** Set the environment variable `UCX_RNDV_THRESH=auto,inter:12kb` to restore performance. |
| | **Keywords**: Performance; MPI; Collectives |
| | **Discovered in Version:** 2.25 |
| 4546016 | **Description:** HCOLL is not supported on GB200 and GB300 systems, with no plans for future support |
| | **Workaround:** N/A |
| | **Keywords:** HCOLL; GB200; GB300 |
| | **Discovered in Version:** 2.24 |
| 4422894 | **Description:** When sending relatively large amounts of data, the following error may occur: rc_verbs_iface.c:128  send completion with error: remote access error |
| | **Workaround:** To avoid this issue, exclude the `rc_v` transport from the list of available transports by setting the environment variable as follows: `UCX_TLS=^rc_v` |
| | **Keywords:** UCX; remote access error |
| | **Discovered in Version:** 2.23 |
| 4177839 | **Description**: When using a bidirectional traffic pattern in setups with 4 NICs, each connected to a separate NUMA node, a 10% degradation in bandwidth performance is observed compared to the full wire speed (FWS). |

| Reference Number | Issue |
|---|---|
| | **Workaround:** Use a single lane for the RNDV operation instead of the default two lanes by setting the following environment variable: `UCX_MAX_RNDV_LANES=1` |
| | **Keywords:** NUMA; BW; FWS |
| | **Discovered in Version:** 2.22.x |
| 3995982 | **Description:** GPU device variables (obtained from `cudaGetSymbolAddress`) are not supported for MPI send/receive operations. Passing a pointer to a device variable may lead to a segmentation fault. |
| | **Workaround:** Copy the contents of the device buffer to a bounce buffer allocated by `cudaMalloc`, and use that bounce buffer for communication. |
| | **Keywords:** `cudaGetSymbolAddress`; `cudaMalloc`; segmentation fault; bounce buffer |
| | **Discovered in Version:** 2.21.0 |
| 4050321 | **Description:** Significant bandwidth degradation occurs when the Global VA feature is enabled (by setting `UCX_GVA_ENABLE=y`), due to the failure to utilize PCIe relaxed ordering. |
| | **Workaround:** Avoid setting UCX_GVA_ENABLE=y to prevent potential bandwidth degradation. |
| | **Keywords:** Global VA; GVA; ODP |
| | **Discovered in Version:** 2.21.0 |
| 4097336 | **Description:** Enabling HW DCS (by setting `UCX_DC_MLX5_TX_POLICY=dcs_hybrid`) may cause the application to hang due to an issue with scheduling work on DC initiator QPs. |
| | **Workaround:** Avoid setting `UCX_DC_MLX5_TX_POLICY=dcs_hybrid` to prevent potential application hangs. |
| | **Keywords:** DC; DCS; hang |
| | **Discovered in Version:** 2.21.0 |
| 4139280 | **Description:** Asynchronously allocated CUDA memory may not work correctly with the gdr_copy transport, potentially resulting in an error such as: `gdr_copy_md.c:139 UCX ERROR gdr_pin_buffer failed.` |
| | **Workaround:** Set the `UCX_TLS=^gdr_copy` environment variable to disable gdr_copy transport. |
| | **Keywords:** gdr_copy; memory registration; Stream Ordered CUDA Allocator |
| | **Discovered in Version:** 2.21.0 |
| 4026461 | **Description:** UCX atomic operations on Grace CPU may fail with Remote Access error. |
| | **Workaround:** Disable DevX and KSM memory registration by setting `UCX_IB_MLX5_DEVX=no` |
| | **Keywords:** Atomic; Grace |
| | **Discovered in Version:** 2.20.0 |
| 3884209 | **Description:** In certain scenarios, a significant performance degradation can be observed due to excessive memory registrations. |

| Reference Number | Issue |
|---|---|
| | **Workaround:** Switch back to legacy protocols implementation by setting `UCX_PROTO_ENABLE=n` |
| | **Keywords:** UCC, Performance |
| | **Discovered in Version:** 2.19.0 |
| 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.<br>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.<br>`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) |

| Reference Number | Issue |
|---|---|
| - | **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.<br>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.<br>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**:<br>1. Set **UCX_UD_TIMEOUT=120** (the default is 30 seconds)<br>2. 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) |
| - | **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) |
| 3672903/ Github 4105 | **Description:** Adaptive Routing is not supported when used with OpenSHMEM applications. (Github issue: https://github.com/openucx/ucx/issues/4105) |
| | **Workaround:** Enable strong synchronization by adding **-mca spml_ucx_strong_sync 2** parameter to **oshrun** command. |
| | **Keywords:** Adaptive Routing, AR, OpenSHMEM, OSHMEM |
| | **Discovered in Version:** 2.5 (OpenSHMEM 1.4) |
| - | **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:<br>"... 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.<br>(Github issue: https://github.com/openucx/ucx/issues/1162) |

| Reference Number | Issue |
|---|---|
| | **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.<br>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:<br>$ ./utils/hpcx_rebuild.sh --rebuild-ucx --ucx-extra-config "--with-avx=no" |
| | **Keywords:** UCX |

# 2 Installing and Loading HPC-X

## 2.1 Installing HPC-X

*To install HPC-X:*

1. Extract HPC-X tarball into your current working directory. The general naming scheme is: **hpcx-<version>-<compiler>-<doca>-<distro>-<cuda>-<arch>.tbz** .

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

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

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

## 2.2 Building and Running Applications with HPC-X

HPC-X includes Open MPI v4.1.x and Open MPI v5.0.x.

The symbolic links *hpcx-init.sh* and *modulefiles/hpcx* point to the default version (Open MPI v5.0.x).

*To load OpenMPI/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
```

*To load OpenMPI/OpenSHMEM v4.1.x based package:*

```
% export HPCX_ENABLE_OMPI4=1
% source $HPCX_HOME/hpcx-init.sh
% hpcx_load
% env | grep HPCX
% mpicc $HPCX_MPI_TESTS_DIR/examples/hello_c.c -o $HPCX_MPI_TESTS_DIR/examples/hello_c
% mpirun -np 2$HPCX_MPI_TESTS_DIR/examples/hello_c
% oshcc $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c.c -o $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c
% oshrun -np 2$HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c
% hpcx_unload
```

## 2.3 Building HPC-X with the Intel Compiler Suite

As of version 1.7, 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.

## 2.4 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
```

*To load Open MPI/OpenSHMEM v5.0.x based package:*

```
% export HPCX_ENABLE_OMPI5=1
% module use $HPCX_HOME/modulefiles
% module load hpcx
% mpicc $HPCX_MPI_TESTS_DIR/examples/hello_c.c -o $HPCX_MPI_TESTS_DIR/examples/hello_c
% mpirun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_c
% oshcc $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c.c -o $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c
% oshrun -np 2 $HPCX_MPI_TESTS_DIR/examples/hello_oshmem_c
% module unload hpcx
```

## 2.5 HPC-X Environments

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. Use this module for multi-threaded MPI 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.

Unable to render include or excerpt-include. Could not retrieve page.

> ⚠ 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 Running, Configuring and Rebuilding HPC-X

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

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

## 3.1 Profiling MPI API Application with IPM

*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'.

## 3.2 Rebuilding Open MPI

### 3.2.1 Rebuilding Open MPI Using a Helper Script

The $HPCX_HOME/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_HOME/utils/hpcx_rebuild.sh --help
```

### 3.2.2 Rebuilding Open MPI from HPC-X Sources

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

*To build Open MPI from sources:*

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

Open MPI and OpenSHMEM are pre-compiled with UCX 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>
```

## 3.3 Running MPI with HCOLL

HCOLL is disabled 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 <...>
```

## 3.4 Direct Launch of Open MPI and OpenSHMEM using SLURM 'srun'

The default HPC-X is not built with SLURM support. In order to use a direct launch with srun, rebuild the OpenMPI or HPC-X with the slurm version installed on the system:

- 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 '*

## 3.5 Process and Memory Affinity with Open MPI v5.0

Starting with Open MPI v5.0, the runtime environment transitioned from ORTE to the PMIx Reference RunTime Environment (PRRTE). Processor and memory affinity management is now handled by PRRTE, which operates as an independent submodule.

While mapping and binding still utilize hwloc and the familiar **`--map-by <object>`** and **`--bind-to <object>`** directives, the default mapping policy has changed from map-by-socket to map-by-core.

Processor affinity must be enabled for memory affinity to function. An unbound process may migrate off its local NUMA memory and lose locality performance benefits.

### 3.5.1 Default Mapping and Binding Policy: OMPI v4.1.x vs. v5.0.x

| Feature / Behavior | OMPI v4.1.x (ORTE) | OMPI v5.0.x (PRRTE 3.x) |
|---|---|---|
| **Default Mapping** | **`--map-by numa`** *(falls back to socket, then slot)* | **`--map-by core`** |
| **Default Binding** | **`--bind-to numa`** | **`--bind-to core`** |
| **Placement (N ≪ cores)** | Ranks round-robin across NUMA domains—load is spread across sockets. | Ranks placed sequentially on consecutive cores—N ranks all land on socket 0 until that socket fills. |

### 3.5.2 Practical Impact

For multi-threaded MPI codes, the v5.0.x default packs all ranks on the first socket. As a result, threads of every rank will contend for the exact same memory subsystem. Conversely, the v4.1.x default spreads one rank per NUMA domain, leaving each rank a full NUMA's worth of cores for its respective threads.

### 3.5.3 Recovering v4.1.x-Equivalent Behavior Under v5.0.x

To recover the legacy behavior, pass the mapping and binding rules explicitly.

For NUMA-granularity placement:

```
mpirun --map-by numa --bind-to numa -n N <app>
```

For socket-granularity placement:

```
mpirun --map-by socket --bind-to socket -n N <app>
```

Passing only `--bind-to core` does not restore spread placement. When a binding is specified without an explicit mapping, v5.0.x automatically sets the mapping to match the binding object ( `--bind-to core` $\Rightarrow$ `--map-by core` ), producing the same packed layout on socket 0.

Always pair it with an explicit `--map-by ...` flag if you need a specific rank distribution across your hardware topology.

### 3.5.4 References

- Open MPI Affinity Documentation: https://docs.open-mpi.org/en/main/tuning-apps/affinity.html
- PMIx and PRRTE Launching: https://docs.open-mpi.org/en/v5.0.x/launching-apps/pmix-and-prrte.html
- Up-to-date PRRTE Syntax: Run `prterun --help map-by` in your environment.

# 4 HCOLL

## 4.1 Overview

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

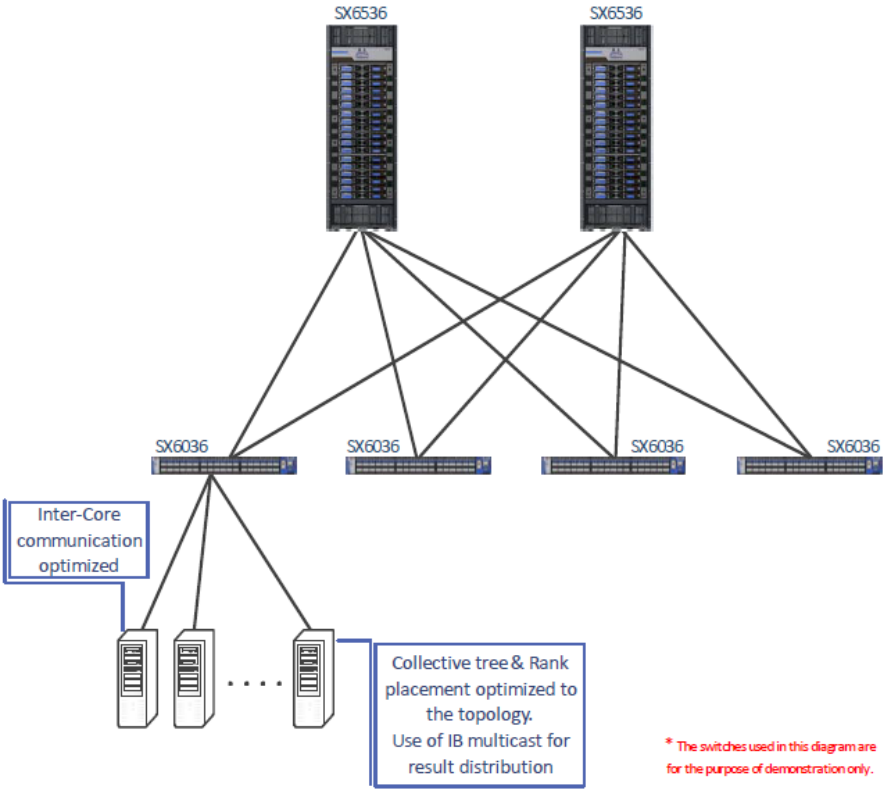
Collective communications execute global communication operations to couple all processes/nodes in the system and therefore must be executed as quickly and as efficiently as possible. Indeed, the scalability of most scientific and engineering applications is bound by the scalability and performance of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

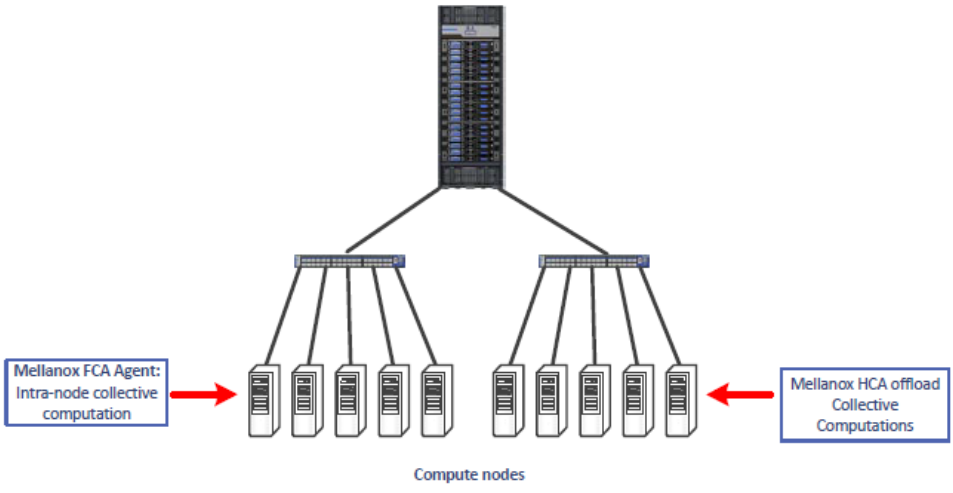
Additionally, HCOLL contains support for building runtime configurable hierarchical collectives. HCOLL leverages hardware multicast capabilities to accelerate collective operations. In HCOLL, the performance and scalability of the UCX point-to-point library in the form of the "ucx_p2p" BCOL is fully 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:




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



## 4.2 Using HCOLL

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

### 4.2.1 Enabling HCOLL in Open MPI

Starting with HPC-X v2.50, UCC is the default collective framework for Open MPI and Open SHMEM collectives. HCOLL is included in HPC-X but is disabled by default.

To run with HCOLL, use:

```
--mca coll_hcoll_enable 1
```

To force HCOLL instead of UCC, use:

```
--mca coll_hcoll_enable 1 --mca coll_ucc_enable 0
```

### 4.2.2 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 ...
  ```

### 4.2.3 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>
```

### 4.2.4 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)

### 4.2.5 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.

#### 4.2.5.1 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.

### 4.2.6 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>
```

### 4.2.7 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. |
| 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

## 4.2.8 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.

# 5 Unified Communication - X Framework Library

## 5.1 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:
  - Reliable Connected (RC)
  - Unreliable Datagram (UD)
  - Dynamically Connected (DC)
  - Accelerated verbs
  - DPU transport (GGA)

    ⚠ GGA is supported on BlueField-3 SuperNICs and above.
- Shared memory communication, including CMA and XPMEM
- RoCE
- TCP
- CUDA/gdrcopy

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

### 5.1.1 Supported CPU Architectures

Unified Communication - X Framework (UCX) supported CPU architectures are: x86_64 and aarch64.

## 5.2 Configuring UCX

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

### 5.2.1 Using UCX with OpenMPI

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

*To use UCX with Open MPI explicitly:*

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

*To use UCX with OpenSHMEM explicitly:*

```
$oshrun --mca spml ucx -mca atomic ucx ...
```

### 5.2.2 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 compiled with tuning for the specific CPU on the system (instead of the default build, which aims to be compatible with a broad range of CPU models).
To accomplish this, please compile UCX with:

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

## 5.3 Tuning UCX Settings

The default UCX settings are already optimized for out-of-box performance. 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 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 |
| **hw_dcs** | A single DCI that operates as a HW DCS queue. The channels are assigned in a round-robin fashion |
| **dcs_hybrid** | Same as "dcs_quota". However, when there are no DCIs available, a dedicated HW DCI is used in the same manner as in "hw_dcs" policy |

- 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).

## 5.4 UCX Features

### 5.4.1 Hardware Tag Matching

⚠ This feature is in maintenance support mode for adapter cards from ConnectX-5 through ConnectX-7 and is unsupported on ConnectX-8 NICs and newer.

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 RoCE is not supported.

For further information, refer to Understanding Tag Matching for Developers post.

### 5.4.2 Single Root IO Virtualization (SR-IOV)

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

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.

### 5.4.3 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.

For RoCE adapters, the option **UCX_IB_AR_ENABLE=<yes/try/auto/no>** enables adaptive routing if the hardware configuration supports it. If set to **yes** and the network does not support it, an error will occur. If set to **try** , any lack of support will be silently ignored. Setting to **no** will force disable it or return an error.

For InfiniBand adapters, the activation is performed with **UCX_IB_SL** and **UCX_IB_AR_ENABLE** options as listed in the table below.

⚠ 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.

#### 5.4.3.1 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 (for example, the ucp_ep_create API function needs to be addressed).

### 5.4.4 CUDA GPU

#### 5.4.4.1 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.

System Requirements

- CUDA v12.0 or higher. For information on how to install CUDA, please refer to NVIDIA documents for *CUDA Toolkit.*
- For GPUDirect support, need either a recent kernel with dmabuf support or a GPUDirect RDMA driver. For more information, please refer to DOCA-Host documentation.
- To install GPUDirect RDMA driver with DOCA-Host. For more information, please refer to DOCA-Host documentation.
- The optional **gdrcopy** driver optimizes the transfer latency of small messages residing in GPU memory. For information on how to install GDR COPY, refer to its *GitHub webpage*.

  It is important to make sure that the **gdrcopy** driver is installed properly on each of the compute nodes taking part in the MPI job.

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

```
lsmod | grep gdrdv
```

### 5.4.5 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**

### 5.4.6 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.

### 5.4.7 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.

### 5.4.8 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.

### 5.4.9 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.

### 5.4.10 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).

### 5.4.11 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.

### 5.4.12 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.

### 5.4.13 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.

⚠ UCX inside HPC-X is built with **--with-fuse3** 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.

#### 5.4.13.1 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

### 5.4.13.2 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.

## 5.4.14 On-demand Paging (ODP)

On-demand paging mitigates the limitations of memory registration by allowing applications to avoid pinning down physical pages of the address space and tracking mapping validity.

Instead, the Host Channel Adapter (HCA) requests updated translations from the operating system when pages are absent, and the OS invalidates translations affected by non-present pages or mapping alterations.

With ODP, system memory is not locked and therefore is allowed to be swapped out if necessary.

The feature is controlled by `UCX_REG_NONBLOCK_MEM_TYPES` environment variable. UCX supports both ODP versions - ODPv1 and ODPv2. Which ODP version can be used depends on FW version and configuration.

To enable ODPv2 in UCX it is enough to specify `UCX_REG_NONBLOCK_MEM_TYPES=host`.

To enable ODPv1, in addition to setting `UCX_REG_NONBLOCK_MEM_TYPES=host`, devx objects creation must be disabled using `UCX_IB_MLX5_DEVX_OBJECTS=""`.

⚠ This feature is enabled by default on Grace platforms. On all other platforms it is disabled by default and can be activated using environment variables described above.

## 5.4.15 Multi-Node NVLINK (MNNVL)

Multi-Node NVLINK (MNNVL) enables NVLINK communication between processes located on different nodes. MNNVL support is disabled by default in UCX. To enable this feature, set the environment variable `UCX_CUDA_IPC_ENABLE_MNNVL=y`.

Additionally, for applications that create endpoints with error handling support, `UCX_RNDV_PIPELINE_ERROR_HANDLING=y` needs to be set.

⚠ Setting `UCX_RNDV_PIPELINE_ERROR_HANDLING=y` may result in data corruption during the error handling flow. Once all missing parts of the error handling flow are implemented, this environment variable will be deprecated.

## 5.5 DPU Transport (GGA)

The GGA Transport enables High-Speed Data Transfer between host memory and the DPU's internal memory utilizing the Memory-to-Memory Offload (MMO) acceleration engine.

## 5.6 Global Virtual Address

The Global VA (Global Virtual Address) feature introduces a new memory registration model that registers the entire virtual address space of a process for a given transport. This is particularly useful for InfiniBand transports, where it is implemented using Implicit On-Demand Paging (ODP). Traditionally, RDMA operations require explicit memory registration, which can lead to significant overhead in large-scale applications. With Global VA, the entire process's virtual address space is registered once, allowing any memory within the process to be accessed without further registrations. Implicit ODP enables the hardware to dynamically handle page faults and map memory as needed, eliminating the need to pin down physical memory. The primary goal of this feature is to reduce the number of memory keys used for RDMA operations, which improves efficiency and scalability.

## 5.7 UCX Utilities

### 5.7.1 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.

## 5.8 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/Statistics*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:

- 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.

# 6 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



Starting with HPC-X v2.50, UCC is the default collective framework for Open MPI and Open SHMEM collectives.

## 6.1 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**

## 6.2 TL/UCP Strided Active Set

UCC now supports broadcast collectives over strided active sets larger than two ranks within the TL/UCP transport. This feature allows applications and runtimes to define a specific subset of participating ranks - using the **active_set.start**, **active_set.stride**, and **active_set.size** fields - which UCC then maps into a subset rank space to execute the broadcast only across those selected participants. This support is available in the TL/UCP broadcast path, including the knomial and double-binary-tree broadcast algorithms, and removes the previous limitation where active-set broadcast was restricted to a maximum of two ranks.

## 6.3 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**.

## 6.4 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.

## 6.5 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
```

## 6.6 CPU/GPU Bcast

This feature implements the MCAST Bcast algorithm in UCC, which is disabled by default. To activate the algorithm, users must configure the following environment variables:

- **-x UCC_TL_MLX5_MCAST_NET_DEVICE=<HCA>** (e.g., mlx5_0:1)
- **-x UCC_TL_MLX5_MCAST_ENABLE=1** (Enables MCAST algorithms in TL_MLX5)
- **-x UCC_TL_MLX5_MIN_TEAM_SIZE=N** (Where N is greater than or equal to 2 and less than or equal to the number of processes in the job)
- **-x UCC_TL_MLX5_TUNE=inf** (Sets the maximum priority for all MLX5 algorithms)

Additionally, users should adjust the following Open MPI variables:

- **-x OMPI_UCC_CL_BASIC_TLS=^sharp,nccl**
- **-x OMPI_UCC_CL_HIER_NODE_LEADERS_SBGP_TLS=^sharp,nccl,shm,cuda**

Alternatively, users can customize the algorithm tuning for specific memory types by configuring the **UCC_TL_MLX5_TUNE** variable:

- **-x UCC_TL_MLX5_TUNE=bcast:host:inf#cuda,cuda_managed:0** (Sets maximum priority for Bcast algorithms for host memory and disables MLX5 for cuda and cuda managed memory).

# 7 PGAS Shared Memory Access

## 7.1 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.

## 7.2 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 (DOCA-Host), 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.

# 7.3 Running HPC-X OpenSHMEM

## 7.3.1 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)
  - 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/

### 7.3.1.1 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
```

## 7.3.2 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;
}
```

### 7.3.3 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*

#### 7.3.3.1 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>*

#### 7.3.3.2 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
```

#### 7.3.3.3 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 ClusterKit

## 8.1 Introduction

ClusterKit is a multipurpose node assessment tool for high-performance clusters, aimed at conducting the following tests:

- General Assessments: Latency, bandwidth, effective bandwidth, memory bandwidth, ordered ring bandwidth, and random ring bandwidth
- GPU Communication Tests: Memory bandwidth, GPU-GPU latency and bandwidth, GPU-Host latency and bandwidth, and NCCL bandwidth and latency
- Collective Evaluations: Barrier, allreduce, broadcast, alltoall, and NCCL
- Bisectional Bandwidth
- CPU/GPU Stress

## 8.2 ClusterKit Requirements

- It is recommended to install ClusterKit on a shared directory.
- If such directory does not exist - make sure that all scripts are available on all the hosts in the exact same directory.
- SLURM or passwordless ssh connectivity across the hosts.

## 8.3 Running ClusterKit

### 8.3.1 Selecting Nodes

#### 8.3.1.1 With SLURM

No additional action required.

#### 8.3.1.2 Without SLURM

- Create a hostfile with hostnames or IP addresses of the servers.
- The **hostfile.txt** should contain an ordered, newline-separated list of all host names as used in ssh - see mpirun manpage for more details

### 8.3.2 Basic Options

The primary method for running ClusterKit is through the wrapper script clusterkit.sh. HPCX_CLUSTERKIT_DIR environment variable should point to the ClusterKit directory, which should happen automatically if HPC-X is correctly initialized:

```
$HPCX_CLUSTERKIT_DIR/bin/clusterkit.sh [-f hostfile] [--ssh] [--hca_list "HCAs to use"] [--ppn <num>] [--map-by
node|core|socket] [--mpi_opt "mpirun options"] [--exe_opt "clusterkit options"]
```

where:

| | |
|---|---|
| **-f** | hostfile for running w/o SLURM |
| **--ssh** | Used for launching processes (default autoselect) |
| **--hca_list** | Provides a list of HCAs to use, passed to UCX (see DOCA documentation) |
| **--ppn** | Launches <num> number of processes per node\|core\|socket (see **--map-by**) |
| **--map-by node\|core\|socket** | Runs given number of processes per resource |
| **--mpi_opt** | Passes additional options to mpirun (see mpirun manpage) |
| **--exe_opt** | Passes additional options to clusterkit binary (run **$HPCX_CLUSTERKIT_DIR/bin/clusterkit --help** for possible options) |

The default is to run bandwidth and latency pairwise tests. To select a specific test, use --exe_opt "-d <test1> -d <test2> ..."

Additional options mentioned below should be passed directly to clusterkit binary with --exe_opt option of clusterkit.sh wrapper.

### 8.3.3 ClusterKit Evaluation Logic for *Pairwise* Tests

Certain tests in ClusterKit are pairwise, meaning there is a specific logic for processing their results, as described below.

For each particular type of test, ClusterKit repeats it a number of times across appropriate population of ranks, using given message sizes (default or specified) and collects performance statistics.

In *pairwise* experiments and *FULL* test mode, ClusterKit repeats the experiment on pairs it selects in each round.

- For a system with `n` nodes, there can be up to `n(n-1)/2` distinct pairs, allowing for `n/2` communicating pairs to be tested simultaneously.
- ClusterKit operates over `n-1` rounds, selecting a different destination for the same source in each pair.

A 'tolerance' is a specified percentage derived from the extreme values of the observed performance distribution, used to characterize a component's performance as 'unacceptable.'

- Message latencies that are 2.1 times (by default) above the minimum are considered 'bad,'
- Message bandwidths (BWs) less than 93% (by default) of the maximum are also deemed 'bad.'
- Different values can be supplied.

Pairs whose performance falls outside the tolerance are selected to participate in subsequent evaluation rounds for re-testing. In each subsequent round, the total number of pairs is halved. In the final round, pairs that continue to perform outside the tolerance are declared definitively 'bad.'

The `-x` option for the ClusterKit binary disables the retesting logic, causing ClusterKit to test all possible pairs once and then stop.

In pairwise experiments conducted in QUICK mode, **n/2** communicating pairs go through one round of evaluation. To run in QUICK mode, pass **-q** ( **--quick** ) to the ClusterKit binary.

In CUSTOM mode, we specify the pairs to test in each round using an input file.

#### 8.3.3.1 *CUSTOM* Mode

To run a *pairwise* test in *CUSTOM* mode, pass **-f <file> ( --fromfile=<file>)** to the ClusterKit binary.

The file consists of lines, with each line formatted as:

```
<round_num> <node_1> <node_2>
```

All pairs (links between **<node_1>** and **<node_2>** ) with the same **<round_num>** will be tested in parallel. **<round_num>** should be in non-descending order. For example:

```
1 machine02 machine10
1 machine03 machine07
2 machine02 machine03
```

This will test (machine02, machine10) and (machine03, machine07) in round 1 and (machine02, machine03) in round 2.

### 8.3.4 Accounting for Oversubscription

While performing a bandwidth test (a pairwise test), ClusterKit assesses each link to determine if it is 'bad' (i.e., has too low bandwidth) and needs to be retested. This assessment is made by comparing the measured bandwidth of a specific link with the highest bandwidth among all links. In a network topology with oversubscription, the bandwidth of links going through multiple TOR switches may be lower than that of a link within a single TOR. This occurs when the total uplink bandwidth is less than the total bandwidth from the TOR switch to the connected nodes. The total bandwidth available for communication between nodes connected to different TORs is limited by the lowest uplink bandwidth of the two TOR switches involved. The ratio of total downlink bandwidth to total uplink bandwidth is referred to as the oversubscription ratio.

ClusterKit can account for oversubscription using a topology information file. Use the **--topo-file <topofile>** option with the ClusterKit binary to enable this feature. The format of the topology information file is:

```
<scope_name>,<scope_num>,<oversubscription_factor>
```

**<scope_name>** and **<scope_num>** should match those in the scope_info file (see SCOPED tests). The **<oversubscription_factor>** is a floating-point number.

When assessing whether a link is 'bad' with this feature enabled, the link is considered 'bad' if its measured bandwidth is lower than the maximum measured bandwidth divided by the oversubscription ratio.

### 8.3.5 *SCOPED* Tests

Several tests (including collective tests and the bisectional bandwidth test) utilize the concept of a scope. A scope is defined as a set of nodes, and the **scope_info** file is used to define these scopes. The purpose of scoped tests is to analyze how similar sets of nodes (such as all nodes in a single rack) behave and whether there are differences among them.

The **-S <file>** or **--scope_info=<file>** option for the ClusterKit binary is used to specify the **scope_info** file. The format of this file consists of comma-separated lines with three fields:

```
host,scope_name,scope_num
```

The **<scope_name>** and **<scope_num>** pairs should be consistent throughout the entire file. For example:

```
node01,scope1,1
node02,scope1,1
node03,scope2,2
node04,scope2,2
```

This **scope_info** file specifies two scopes, each containing two nodes: node01 and node02 belong to scope1, while node03 and node04 belong to scope2.

### 8.3.6 Running Stress Tests

This feature enables stress testing of the CPU and GPU while conducting other ClusterKit tests. Stress testing means placing a potentially high load (and thus high power consumption) on the CPU and/or GPU during these tests.

To run stress tests, use the **--with-stress[=<STRESS_TYPES>]** option with the ClusterKit binary. **<STRESS_TYPES>** (optional) is a comma-separated list of "cpu," "gpu," or "all." The default is "all," which stresses both the CPU and GPU if available. The **-Y <TEST_TIME>** option specifies the time in minutes for the tests. If the tests finish earlier than the specified **<TEST_TIME>**, they will re-run. If the **-Y** option is not provided, each test will execute only once.

When executing tests alongside stress tests, results may vary significantly due to the load on the CPU and GPU caused by stress testing.

CPU stress testing on RHEL 7 does not support AVX512; therefore, stress testing CPUs that support AVX512 on RHEL 7 may not be fully effective. A corresponding warning will be issued when starting ClusterKit on RHEL 7.

## 8.4 Test Descriptions and Options

> ⚠ All command-line options mentioned in the test descriptions are applicable to the ClusterKit binary (see Running ClusterKit).

### 8.4.1 *Pairwise* Tests

#### 8.4.1.1 Bandwidth Test (-d bw)

The bandwidth test utilizes nonblocking `MPI_Isend` and `MPI_Irecv` calls.

Options:

- Iterations: `-b<iters>`, `--biters=<iters>` (Default: 16)
- Message Size: `-B<size>`, `--bsize=<size>` (Default: 32 MB)
- Unidirectional: `-U`, `--unidirectional` (send data in one direction only; default is bidirectional)
- Tolerance: `-u <tol>`, `--btol=<tol>` (specify tolerance - see ClusterKit Evaluation Logic for Pairwise Tests)

#### 8.4.1.2 Latency Test (-d lat)

The latency test is performed with a series of `MPI_Send` and `MPI_Recv` calls, where one partner sends a message to the other, which then sends a message back. This process is repeated `<iters>` times.

Options:

- Iterations: `-l<iters>`, `--liters=<iters>` (Default: 1024)
- Message Size: `-L<size>`, `--lsize=<size>` (Default: 0 Bytes)
- Tolerance: `-t <tol>`, `--ltol=<tol>` (specify tolerance - see ClusterKit Evaluation Logic for Pairwise Tests)

#### 8.4.1.3 GPU-GPU Latency Test (-d gpu_gpu_lat)

Measures latency of GPU-to-GPU communication with `MPI_ISend` and `MPI_IRecv`.

Options:

- Iterations: `-k`, `--gpulati=<iters>` (Default: 1024)
- Message Size: `-K`, `--gpulats=<size>` (Default: 0 Bytes)
- Tolerance: `-t <tol>`, `--ltol=<tol>` (specify tolerance - see ClusterKit Evaluation Logic for Pairwise Tests)
- Per-GPU test: `-z`, `--bygpu` (test corresponding GPU pairs: GPU0-to-GPU0, GPU1-to-GPU1, etc.)
- Use GPUDIRECT: `-G`, `--gpudirect` (use GPUDIRECT; default is to copy from GPU memory to host)

#### 8.4.1.4 GPU-GPU Bandwidth Test (-d gpu_gpu_bw)

Measures bandwidth of GPU-to-GPU communication with `MPI_ISend` and `MPI_IRecv`.

Options:

- Iterations: `-a`, `--gpubwi=<iters>` (Default: 64)
- Message Size: `-A`, `--gpubws=<size>` (Default: 1 MB)
- Tolerance: `-u <tol>`, `--btol=<tol>` (specify tolerance - see ClusterKit Evaluation Logic for Pairwise Tests)
- Per-GPU test: `-z`, `--bygpu` (test corresponding GPU pairs from different nodes: GPU0-to-GPU0, GPU1-to-GPU1, etc.)
- Use GPUDIRECT: `-G`, `--gpudirect` (use GPUDIRECT; default is to copy from GPU memory to host)

#### 8.4.1.5 NCCL GPU-GPU Bandwidth Test (-d nccl_bw)

Measures bandwidth of GPU-to-GPU communication with NCCL communications primitives.

Options:

- Iterations: `-a`, `--gpubwi=<iters>` (default: 64)
- Message Size: `-A`, `--gpubws=<size>` (default: 1 MB)
- Tolerance: `-u <tol>`, `--btol=<tol>` (specify tolerance - see ClusterKit Evaluation Logic for Pairwise Tests)

#### 8.4.1.6 NCCL GPU-GPU Latency Test (-d gpu_gpu_lat)

Measures latency of GPU-to-GPU communication with NCCL communications primitives.

Options:

- Iterations: `-k`, `--gpulati=<iters>` (default: 1024)
- Message Size: `-K`, `--gpulats=<size>` (default: 0 Bytes)

### 8.4.2 *SCOPED* Tests

#### 8.4.2.1 Collective Tests

Collective tests perform selected collective operations across all nodes in a defined scope.

Types of tests:

- barrier
- Allreduce
- bcast
- Alltoall (set as an argument to -d option)

Options:

- Iterations: `-n` , `--niter=<iters>` (default: 10000)

#### 8.4.2.2 NCCL Collective Tests

Performs NCCL collective operations among nodes in the same scope.

Types of Tests:

- `nccl_bcast`
- `nccl_allreduce`
- `nccl_reduce`
- `nccl_allgather`
- `nccl_reducescatter`

Options:

- Iterations: `-n` , `--niter=<iters>` (default: 10,000)

#### 8.4.2.3 Bisectional Bandwidth Test (-d bisect_bw)

Measures bisectional bandwidth by enabling communication between corresponding nodes in different scopes, assessing potential interference.

Options:

- Iterations: `-b<iters>` , `--biters=<iters>` (default: 16)
- Message Size: `-B<size>` , `--bsize=<size>` (default: 32 MB)
- Unidirectional: `-U` , `--unidirectional` (sends data in one direction only)
- Scope Order: `--scope_order=<scope_order>` (sets order of scopes for testing)

Scope Order File Format: The file consists of lines formatted as follows:

```
<pass_num>,<scope1>,<scope2>
```

Example:

```
1,scope01,scope02 1,scope03,scope04 2,scope02,scope03 3,scope01,scope04 3,scope02,scope03
```

This instructs ClusterKit to execute 3 passes, testing specified connections.

### 8.4.3 Other Tests

#### 8.4.3.1 Memory Bandwidth Test (-d mb)

The memory bandwidth test can be conducted with one of the following operations:

- ADD: `a[i] = b[i] + c[i]`
- COPY: `a[i] = b[i]`
- SCALE: `a[i] = D * b[i]`

- TRIAD: `a[i] = b[i] + D * c[i]`

Options:

- Iterations: `-I <iters>`, `--mbiters=<iters>` (default: 16)
- Array Size: `-I <size>`, `--mbsize=<size>` (default: 4 * L3 cache size)
- Test Type: `-m <type>`, `--memtest=add|copy|scale|triad` (default: TRIAD)

#### 8.4.3.2 Effective Bandwidth Ordered Test (-d beff_o)

Rings of doubling size are formed, starting at 2, and messages are passed in one direction based on rank ordering.

Options:

- Iterations: `-e`, `--beffi=<iters>` (default: 512)
- Message Size: `-E`, `--beffs=<size>` (default: 32 MB)
- Tolerance: `-u <tol>`, `--btol=<tol>` (specify tolerance. Nodes showing results worse than max * tolerance will be considered 'bad')

#### 8.4.3.3 Effective Bandwidth Random Test (-d beff_or)

Similar to the ordered test, but rings are created randomly.

Options:

- Iterations: `-e`, `--beffi=<iters>` (default: 512)
- Message Size: `-E`, `--beffs=<size>` (default: 32 MB)
- Tolerance: `-u <tol>`, `--btol=<tol>` (specify tolerance. Nodes showing results worse than max * tolerance will be considered 'bad')

#### 8.4.3.4 GPU Memory Bandwidth Test (-d gpumb)

Measures bandwidth for host-to-GPU and GPU-to-host memory transfers.

Options:

- Iterations: `-j`, `--gpumbi=<iters>` (default: 16)
- Message Size: `-J`, `--gpumbs=<size>` (default: 0 bytes)
- Tolerance: `-u <tol>`, `--btol=<tol>` (specify tolerance. Nodes showing results worse than max * tolerance will be considered 'bad')

#### 8.4.3.5 GPU Neighbor Latency Test (-d gpu_neighbor_lat)

A restricted variant of the GPU-GPU latency test that measures communication only between GPUs on neighboring nodes.

Options:

- Iterations: `-k`, `--gpulati=<iters>` (default: 1024)
- Message Size: `-K`, `--gpulats=<size>` (default: 0 bytes)

- Use GPUDIRECT: `-G`, `--gpudirect` (use GPUDIRECT - default is to copy from GPU memory to host)

#### 8.4.3.6 GPU Neighbor Bandwidth Test (-d gpu_neighbor_bw)

A restricted variant of the GPU-GPU bandwidth test that measures communication only between GPUs on neighboring nodes.

Options:

- Iterations: `-a`, `--gpubwi=<iters>` (default: 64)
- Message Size: `-A`, `--gpubws=<size>` (default: 1 MB)
- Use GPUDIRECT: `-G`, `--gpudirect` (use GPUDIRECT - default is to copy from GPU memory to host)

## 8.5 Automated Scope_Info File Creation

Bundled with ClusterKit is the script `$HPCX_CLUSTERKIT_DIR/bin/scopes/run_scopes.sh`.

This auxilliary script can create a scope_info file (see SCOPED Tests); create an order_info file for the Bisectional Bandwidth Test; and create a topology file (see Accounting for Oversubscription). It can process the output of `ibnetdiscover` utility or run the utility directly.

Syntax:

```
bin/scopes/run_scopes.sh [-C] [-f <scope_info>] [-d <distances_file>] [-o order_info] [-a] (-i <ibnetdiscover_cli> | <ibnetdiscover_output>)
```

where:

| | |
|---|---|
| `-C` | Do not remove the cache directory (Python virtual environment can cache required modules), which speeds up subsequent runs. |
| `-f <scope_info>` | Specify the output filename for the 'full' scope_info file. Use `-` for standard output. The 'full' scope_info file includes all nodes visible on the InfiniBand subnet. |
| `-d <distances>` | Calculate distances between all pairs of TORs and write them to the specified `<distances>` file. Useful for debugging purposes. |
| `-t <topofile>` | Extract topology information (such as switch oversubscription ratios) and write to the specified `<topofile>`. This file is used to make pairwise tests topology-aware (see Accounting for Oversubscription). |
| `-o <order_info>` | Create an `<order_info>` file for testing bisectional bandwidth based on distances between TORs. |
| `-a` | Write all possible pairs to the `<order_info>` file. Without this option, each scope is tested only once; with it, all possible combinations of scopes are tested. |

| | |
|---|---|
| `-i <ibnetdiscover_cli>` | Run `ibnetdiscover` with the specified `<ibnetdiscover_switches>`. Multiple switches can be used if quoted, for example: `bin/scopes/run_scopes.sh -i "-C mlx5_0 -P 1"`. Note that `ibnetdiscover` may require additional permissions to run. |
| `<ibnetdiscover_output>` | • Specify the filename for the `ibnetdiscover` output. Use `-` for standard input. This option is mutually exclusive with the `-i` switch. |

# 9 NCCL-RDMA-SHARP Plugins

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

## 9.1 Overview

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

The environment variable NCCL_IBEXT_DISABLE enables/disables the use of the plugin. When set to NCCL_IBEXT_DISABLE=1, it disables the plugin, causing a fallback to NCCL's native internal communication.

## 9.2 NCCL SHARP Plugin

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

```
NCCL_COLLNET_ENABLE=1
```

> ⚠ 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.

The following environment variable enables SHARP allgather overlap when using the plugin. This is useful when SHARP-based Reduce-Scatter is enabled, so the Reduce-Scatter ↔ Allgather phase can overlap.

```
SHARP_COLLNET_OVERLAP_AG=1
```

# 10 Multi-GPU Support

The feature enables parallelization techniques involving multiple CUDA GPUs within a single process.

## 10.1 Overview

The feature enables parallelization techniques involving multiple CUDA GPUs within a single process in the general case, and hybrid MPI techniques in particular: MPI + OpenACC/OpenMP/ stdpar; allowing a single MPI rank to manage more than one CUDA GPU.

The implementation is done in UCX library. Thus, the use of multiple CUDA GPUs is supported in applications that use UCX library directly, as well as in MPI applications.

Along with this functionality, the requirements for setting the CUDA device ( **cudaSetDevice** ) in each CPU thread of the application process using UCX library interfaces have also been relaxed. Previously, the user had to set the CUDA device in all threads, including the progress thread. Now, the user may set the CUDA device only once to utilize the CUDA features in UCX. Since CUDA devices in CUDA Runtime API and **CUcontext** s in CUDA Driver API are synonymous, this is also true for applications using CUDA Driver API.

## 10.2 Performance Tuning

By default, UCX library uses closest NICs to the CUDA GPU which is set before the initialization of the library. This is the optimal policy for applications using one CUDA GPU per process. The following combination of UCX parameters can be used to achieve the best performance for applications using multiple CUDA GPUs.

```
UCX_SELECT_DISTANCE_MD=
UCX_CONNECT_ALL_TO_ALL=y
```

## 10.3 Limitations

The feature is not supported for memory allocated VMM API if the device on which the memory is allocated and the device for which access rights are set do not match.

# 11 Spectrum-X NCCL Plugin

## 11.1 Overview

The Spectrum-X NCCL Plugin provides a set of plugins designed to optimize NVIDIA's NCCL library for Spectrum-X and Quantum platforms. It enables more resilient and higher-performance communication across these platforms. In HPC-X, the plugin is located at: **$HPCX_DIR/nccl_spectrum-x_plugin/lib** .

## 11.2 Loading the Plugin

For NCCL to detect the network plugin, add the plugin path to the library search path environment variable.

The plugin can be loaded by explicitly setting the library search path using **LD_LIBRARY_PATH** :

```
$ export LD_LIBRARY_PATH=$HPCX_DIR/nccl_spectrum-x_plugin/lib:$LD_LIBRARY_PATH$ <run command>
```

With HPCX, the plugin can also be loaded by NCCL's environment variable **NCCL_NET_PLUGIN=spcx**

## 11.3 Features

- Network Failure Recovery - Automatically detects and recovers from communication link failures, ensuring uninterrupted distributed training and system stability without user intervention.
- Dynamic Network Load Balance - Dynamically redistributes data across multiple network interfaces based on available throughput and communication patterns, preventing bottlenecks and improving overall transfer efficiency.
- Topology Awareness – Automatically detects long-haul connections using network topology information and applies optimized transport settings to enhance overall network performance.
- NCCL Profiler Plugin – Enables detailed monitoring of NCCL's internal network activity during communication.

### 11.3.1 Resiliency and Load Balancing

The Spectrum-X NCCL Plugin provides enhanced resiliency and load-balancing capabilities when multiple network devices (ports or NICs) are available per GPU in AI workloads. Starting from version 1.2, the plugin also considers the communication pattern being executed by NCCL, like number of peers and traffic per peer, along with device load characteristics to provide optimal distribution of traffic across multiple network devices.

### 11.3.2 Topology Injection and Awareness

Starting with version 1.2, the Spectrum-X NCCL Plugin introduces a mechanism to specify network topology and transport parameters. The plugin supports specifying topology through a file format similar to Slurm's topology format, where each entry defines connections between switches and hosts with relative latency or cable-length values.

Based on this topology, the plugin automatically identifies long-haul connections and takes advantage of NVIDIA XGS configuration for these. It also improves the NCCL algorithm selection model by taking the latencies across the long-haul connections into account.

For more information, refer to NVIDIA XGS.

### 11.3.3 Configurable Bandwidth Loss Limits

Starting with version 1.3, the Spectrum-X NCCL Plugin introduces configurable thresholds for acceptable bandwidth degradation during transparent failover. Applications can specify multi-tier limits where higher bandwidth loss triggers shorter tolerance windows before a fatal error is raised. Configuration is done through the **NCCL_IB_NIC_BW_LOSS_LIMITS** environment variable, which accepts comma-separated pairs of loss percentage and duration in minutes:

```
$ export NCCL_IB_NIC_BW_LOSS_LIMITS=25:2,50:1
```

This example configuration triggers a fatal error if bandwidth loss reaches 25% for 2+ minutes, and 50% for 1+ minute, allowing brief performance drops during recovery while preventing long-running jobs from continuing with severely degraded performance.

### 11.3.4 SHARP

Plugin supports Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) - an in-network computing technology on InfiniBand Quantum switches.

The following environment variable enables the SHARP aggregation with NCCL when using the plugin.

```
NCCL_COLLNET_ENABLE=1
```

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.

The following environment variable enables SHARP allgather overlap when using the plugin. This is useful when SHARP-based Reduce-Scatter is enabled, so the Reduce-Scatter ↔ Allgather phase can overlap.

```
SHARP_COLLNET_OVERLAP_AG=1
```

### 11.3.5 NetInspector Profiler

The Spectrum-X NCCL Plugin extends the NCCL Inspector profiler with NetInspector, which adds network-level instrumentation and telemetry.

#### 11.3.5.1 NetInspector provides:

- Transaction-level metrics – Per-transaction send/receive data including chunk sizes, resent bytes, div sizes (for app-aware load balancing), and peer addressing.
- Burst bandwidth telemetry – Real-time bandwidth measurements per InfiniBand device using exponential moving averages (EMA with configurable beta).
- Burst slowdown detection – Configurable threshold-based alerts when burst bandwidth falls below expected levels, with deficit tracking.
- Queue Pair Load Balancing – QP cumulative load-balancing weight metrics per device.
- Device resilience – Device failure and recovery events with total bandwidth, active device count, and bandwidth delta tracking.
- Link error monitoring – Link error events with IB work completion status codes and cumulative link-error counts per device.
- Async thread events – InfiniBand asynchronous event monitoring (port state changes, errors, etc.) captured from the IB async thread.
- Collective performance – Per-collective metrics including algorithm bandwidth, bus bandwidth, message sizes, execution time, and timing source (GPU/CPU). Supports optional verbose mode with detailed event sequence numbers and timestamps per kernel channel.

#### 11.3.5.2 Export Formats

NetInspector supports two export backends:

- JSON Lines (.jsonl) – Microsecond-level timestamps in JSON Lines format for performance analysis
- DTS (DOCA Telemetry Service) – OTLP-compatible metrics and logs export for integration with observability platforms (Prometheus, Grafana, etc.)

#### 11.3.5.3 Environment Variables

| Variable | Default | Description |
|---|---|---|
| NCCL_INSPECTOR_ENABLE | 0 | Enables the Inspector plugin. Set to 1 to enable profiling. |
| NCCL_INSPECTOR_TELEMETRY_LEVEL | 1 | Controls network-event verbosity:<br>0 = disabled;<br>1 = enabled (collective and kernel-channel events);<br>2 = enabled with per-transaction logging |
| NCCL_PROFILER_PLUGIN | – | Specifies the path to the profiler plugin library. Example: /path/to/libnccl-profiler-inspector.so,<br>or inspector to search in LD_LIBRARY_PATH. |

| Variable | Default | Description |
|---|---|---|
| NCCL_INSPECTOR_DUMP_THREAD_INTERVAL_MICROSECONDS | 0 | Interval (in microseconds) for the internal dump thread. 0 disables the dump thread. Lower values increase export frequency but may affect performance. |
| NCCL_INSPECTOR_DUMP_TYPE | 0 | Export backend type:<br>0 = disabled (none);<br>1 = JSON;<br>2 = DTS (DOCA Telemetry Service) |
| NCCL_INSPECTOR_DUMP_VERBOSE | 0 | Enables detailed event trace output (sequence numbers, timestamps per kernel channel). Set to 1 for verbose mode. |
| NCCL_INSPECTOR_DUMP_DIR | (auto-generated) | Sets output directory for profiler logs. Defaults to nccl-inspector-unknown-jobid or nccl-inspector-<SLURM_JOBID> when running under SLURM. |
| NCCL_INSPECTOR_BURST_SLOWDOWN_THRESHOLD_MBPS | 0 | Burst slowdown detection threshold in MB/s. When burst bandwidth falls below this threshold, a burst_slowdown event is generated. 0 disables detection. |

# 12 Common Abbreviations

## 12.1 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 |

# 13 User Manual Revision History

| Revision | Date | Section | Change |
|---|---|---|---|
| Rev 2.50 | June 04, 2026 | Installing and Loading HPC-X | Updated |
| | | Enabling HCOLL in Open MPI | Updated |
| | | Unified Collective Communication (UCC) | Updated |
| | | Spectrum-X NCCL Plugin | Updated |
| | | “Process and Memory Affinity with Open MPI v5.0” in Running, Configuring and Rebuilding HPC-X | Added |
| Rev 2.26 | February 23, 2026 | Installing and Loading HPC-X | Updated |
| | | Spectrum-X NCCL Plugin | Updated |
| Rev 2.25 | November 05, 2025 | Spectrum-X NCCL Plugin | Added |
| | | NCCL-RDMA-SHARP Plugins | Updated |
| Rev 2.24 | July 31, 2025 | No changes were made to this version. | N/A |
| Rev 2.23 | May 08, 2025 | Spectrum-X NCCL Plugin | Added |
| Rev 2.22.1 | January 31, 2025 | No changes were made to this version. | N/A |
| Rev 2.22.0 | January 08, 2025 | No changes were made to this version. | N/A |
| Rev 2.21.1 | December 24, 2024 | No changes were made to this version. | N/A |
| Rev 2.21.0 | November 07, 2024 | NCCL-RDMA-SHARP Plugins | Updated |
| | | Table under Adaptive Routing | Updated |
| | | Multi-Node NVLINK (MNNVL) | Added |
| | | **hw_dcs** and **dcs_hybrid** under Tuning UCX Settings | Added |
| | | DPU Transport (GGA) | Added |
| Rev 2.20.0 | August 14, 204 | ClusterKit | New section |
| Rev 2.19.0 | May 5, 2024 | Running, Configuring and Rebuilding HPC-X | Updated |
| | | HCOLL | Updated |
| | | NCCL-RDMA-SHARP Plugins | Updated overview |
| | | CPU/GPU Bcast | New section |
| | | On-demand Paging (ODP) | New section |
| | | Unified Communication - X Framework Library | Updated |
| | | Loading KNEM Module | Removed section |
| Rev 2.18.0 | February 9, 2024 | No changes were made to this version. | N/A |

| 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 |

# 14 Release Notes History



## 14.1 Release Notes Change Log History

### 14.1.1 HPC-X Toolkit Change Log History

| Category | Change |
|---|---|
| **Rev 2.26** | |
| **HPC-X Content** | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.14.5<br>• UCC v1.7.0 |
| **Open MPI Support** | Introduced an Open MPI v5.0.10 component in HPC-X v2.26. See Installing and Loading HPC-X page for instructions on using Open MPI v5.0.10. |
| **Spectrum-X NCCL Plugin Configurable Bandwidth Loss Limits** | Added support for configurable bandwidth-loss thresholds during transparent failover. |
| **Spectrum-X NCCL Plugin SHARP** | Added support for SHARP in-network aggregation on InfiniBand Quantum switches, including optional Reduce-Scatter ↔ Allgather overlap |
| **Rev 2.25.1** | |
| Bug Fixes | See Bug Fixes. |
| **Rev 2.25** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.13.0<br>• UCC v1.6.0 |
| Spectrum-X NCCL Plugin | Added support for optimized NCCL communication on Spectrum-X and Quantum platforms, providing enhanced resiliency, dynamic load balancing, topology awareness, and advanced profiling capabilities. |
| Bug Fixes | See Bug Fixes section. |
| **Rev 2.24.1** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release:<br>• UCX v1.20 |
| Bug Fixes | See Bug Fixes section. |
| **Rev 2.23** | |

| | |
|---|---|
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.11.0<br>• UCX v1.19.0<br>• UCC v1.4.4 |
| NCCLNet Plugin | Added support for NCCLNet plugin integration to enhance NCCL performance and resilience on Spectrum-X and Quantum platforms. For further information, see Spectrum-X NCCL Plugin section. |
| **Rev 2.22.1** | |
| ConnectX-8 and XDR Multi-Plane Networks Support | Introducing hardware-level support for Direct Data Placement (DDP), which maximizes the use of all network lanes to achieve the full bandwidth capacity of XDR networks. |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.10.3<br>• UCC v1.4.3 |
| **Rev 2.22.0** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.10.2-1<br>• ClusterKit 1.15 |
| **Rev 2.21.1** | |
| Support for Direct Data Placement in mlx5dv and DevX Interfaces | Added support for direct data placement (DDP) across both mlx5dv and DevX interfaces. This enhancement enables out-of-order receives and achieves full wire speed on ConnectX-8 adapters. |
| Adapter Cards | Added support for ConnectX-8 SuperNIC. |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.10.2 |
| **Rev 2.21.0** | |
| Hybrid Mode for Hardware DCS Offload | Added support for hardware DCS offload in hybrid mode, which combines existing software DCS with hardware DCS offload.<br>To enable this feature, set `UCX_DC_MLX5_TX_POLICY=dcs_hybrid` |
| GGA Transport for High-Speed Data Transfer | Added support for GGA transport, a high-speed DMA copy engine that enables efficient data transfer between host memory and the DPU's internal memory. |
| Multi-Node NVLink with Optimized Protocol Selection | Added support for multi-node NVLink, with automatic detection and selection of the most efficient data transfer protocols. |
| IP Address Filtering for RoCE Devices | Added support for filtering RoCE devices based on their IP address, allowing the selection of specific network subnets.<br>To configure the filter, set `UCX_IB_ROCE_SUBNETS` with a list of subnets. For example: UCX_IB_ROCE_SUBNETS=5.4.3.2/16,1.2.3.4/24. |

| | |
|---|---|
| Automatic Selection of GPU Bounce Buffers for Large Message Transfers | Added support for an optimization that automatically selects GPU bounce buffers for large message transfers when these buffers offer performance benefits over host memory buffers. |
| Single Memory Key Creation Using ODP for Enhanced Efficiency | Added support for a feature that leverages the ODP capability to create a single memory key for the entire process's virtual address space. This reduces the number of allocated memory keys, helping to bypass firmware limitations.<br>To enable this feature, set **UCX_GVA_ENABLE=y** |
| MLNX_OFED to DOCA-OFED Transition | Starting this version, the host driver is part of the NVIDIA DOCA package. DOCA-OFED is a DOCA-Host profile that includes the same components, drivers, and tools as MLNX_OFED. Installing DOCA-OFED will result with the same file system on the host as MOFED.<br>For further information, please see NVIDIA MLNX_OFED to DOCA-OFED Transition Guide. |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.9.0<br>• UCX v1.18.0 |
| **Rev 2.20.0** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v3.8.0<br>• ClusterKit v1.14<br>• OSU Micro-Benchmarks v7.4 |
| ClusterKit | Revised the ClusterKit section to include updated details and added subsections. |
| **Rev 2.19.0** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA SHARP v3.7.0<br>• UCC v13.0<br>• ClusterKit v1.13<br>• Added OSU Micro-Benchmarks v7.3<br>• nccl-rdma-sharp-plugin v2.6 |
| UCC Grace Support and Performance Profile | Added performance improvements of broadcast and reduce collectives on NVIDIA Grace CPU by optimizing parameters of shared memory transport (TL/SHM). |
| Unified Collective Communication (UCC) | Added the CPU/GPU Bcast feature which implements the MCAST Bcast algorithm in UCC. Refer to Unified Collective Communication (UCC). |
| Unified Communication - X Framework Library | Added the on-demand paging feature. Refer to On-demand Paging (ODP) |
| ClusterKit | Added the CPU/GPU stress testing feature. Refer to ClusterKit |
| Bug Fixes | See Bug Fixes. |
| **Rev 2.18.0** | |

| | |
|---|---|
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA SHARP v3.6.0<br>• UCC v13.0<br>• ClusterKit v1.12<br>• Added OSU Micro-Benchmarks v7.3<br>• nccl-rdma-sharp-plugin v2.6 |
| HPC-X and Singularity | Deprecated support for HPC-X singularity containerization. |
| OSU Micro-Benchmarks Path | Removed version from the OSU Micro-Benchmarks path.<br>The current one is ompi/tests/osu-micro-benchmarks and ompi/tests/osu-micro-benchmarks-cuda |
| Known Issues | See Known Issues. |
| **Rev 2.17.1** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA SHARP v3.5.1 |
| Known Issues | See Known Issues. |
| **Rev 2.17.0** | |
| HPC-X Content | Updated HPC-X Content section to reflect the communication libraries versions embedded in this HPC-X release.<br>• NVIDIA SHARP v3.5.0<br>• NCCL v2.x<br>• UCX v1.16<br>• ClusterKit v1.11<br>• nccl-rdma-sharp-plugin v2.5<br>Added the following Supported Platforms and OSs:<br>• Debian 10.x<br>• 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.<br>• NVIDIA SHARP v3.4.1<br>• UCC v1.3<br>• ClusterKit v1.10<br>• nccl-rdma-sharp-plugin v2.4<br>• NCCL v2.18<br>• 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.<br>• NVIDIA SHARP v3.4<br>• UCC v1.3<br>• ClusterKit v1.10<br>• nccl-rdma-sharp-plugin v2.4<br>• NCCL v2.18<br>• 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.<br>• NVIDIA SHARP v3.3<br>• UCX v1.15<br>• ClusterKit v1.9<br>• nccl-rdma-sharp-plugin v2.3<br>• GDRCopy v2.3<br>• NCCL v2.17.1-1<br>• 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.<br>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.<br>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.<br>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).<br>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.<br>• NVIDIA SHARP v3.1<br>• HCOLL v4.8<br>• UCC v1.2<br>• ClusterKit v1.8<br>• 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.<br>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.<br>• UCX version 1.13<br>• 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.<br>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.<br>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.<br>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:<br>• Open MPI version 4.1.x<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 2.4.x<br>• HCOLL version 4.7<br>• UCX version 1.10<br>• ClusterKit version 1.3<br>• 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:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 2.1.0<br>• HCOLL version 4.5<br>• 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.<br>• 8-16 KB message: Improved latency by up to 6.4%, bandwidth by up to 20%, and bidirectional bandwidth by up to 96%<br>• IMB/multiPingPong and osu_mbw_mr for messages up to 32B on full ppn on MLNX_OFED 5.0.<br>**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.<br>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:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 2.0<br>• HCOLL version 4.4<br>• UCX version 1.7 |
| | Removed CUDA init script (hpcx-init-cuda.sh), and environmental module (modules/hpcx-cuda) from HPC-X.<br>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.<br>**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:<br>• • NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.8<br>• HCOLL version 4.3<br>• 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.<br>**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:<br>• Open MPI version 4.0.x<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.7.2<br>• HCOLL version 4.2<br>• UCX version 1.5<br>• 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:<br>• `--with-libevent=internal`<br>• `--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:<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.7<br>• HCOLL version 4.1<br>• UCX version 1.4 |
| | Added support for Singularity containerization.<br>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:<br>• New API for establishing client-server connection.<br>• 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:<br>• Open MPI version 3.1.x<br>• NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.5<br>• HCOLL version 4.0<br>• MXM version 3.7<br>• UCX version 1.3<br>• 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.<br>• Added the following UCX features:<br>• Added support for GPU memory in UCX communication libraries<br>• 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<br>• Improved multi-threaded performance<br>• Improved shared memory performance<br>• Added support for NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v1.5<br>• Added support for NVIDIA SHARP software multi-channel/multi-rail capable algorithms<br>• Improved Allreduce large message algorithm<br>• 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:<br>• OpenMPI version 3.0.0<br>• Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.4<br>• HCOLL version 3.9<br>• UCX version 1.3 |
| UCX | • UCX is now at GA level.<br>• Added the following UCX features:<br>• **[ConnectX-5 only]** Added support for hardware Tag Matching with DC transport.<br>• **[ConnectX-5 only]** Added support for Out-of-order RDMA RC and DC to support adaptive routing with true RDMA.<br>• Added UCX datatypes - community approved datatype support.<br>• Added UCX support to Inbox RHEL.<br>• Added GPUDirect RDMA support.<br>• Hardware Tag Matching (See section *Hardware Tag Matching* in the User Manual)<br>• SR-IOV Support (See section *SR-IOV Support* in the User Manual)<br>• Adaptive Routing (AR) (See section *Adaptive Routing* in the User Manual)<br>• Error Handling (See section *Error Handling* in the User Manual) |

| | |
|---|---|
| HCOLL | • Added support for Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v1.4<br>• Added support for NCCL on-host GPU based collectives.<br>• Added support for Hierarchical GPU based allreduce using NCCL for scale-in and MXM/UCX for scale-out.<br>• Improved shared memory performance for allreduce, barrier, and broadcast. Targeting high thread count systems, e.g. Power9.<br>• Improved large message allreduce (multi-radix, zero-copy fragmentation, CPU vectorization.)<br>• Added new and improved AlltoAllv algorithm - hybrid logarithmic pair-wise exchange.<br>• Added support for on-demand HCOLL memory. Improves HCOLL's memory footprint on high thread count system e.g. Power9.<br>• Added a high performance multithreaded implementation to support MPI_THREAD_MULTIPLE applications. Designed specifically for high thread count systems, e.g. Power9.<br>• HCOLL startup improvements. |
| Open MPI / OpenSHMEM | • Added support for Open MPI 3.0.0.<br>• Added support for xpmem kernel module.<br>• Added a high performance implementation of shmem_ptr() with UCX SPML.<br>• 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.<br>• 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:<br>• OpenMPI version 2.1.2a1<br>• Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) version 1.3.1<br>• HCOLL version 3.8.1652<br>• MXM version 3.6.3103<br>• UCX version 1.2.2947 |
| UCX | Point-to-point communication API, with tag matching, remote memory access, and atomic operations.<br>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:<br>• RDMA<br>• DC<br>• Out-of-order<br>• HW tag matching offload<br>• Registration cache<br>• ODP |

| | |
|---|---|
| | Shared memory communications for optimal intra-node data transfer:<br>• SysV<br>• posix<br>• knem<br>• CMA<br>• 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:<br>• HCOLL updated to v3.7.<br>Open MPI updated to v2.10 |

| | |
|---|---|
| FCA | FCA 2.x is no longer the default FCA used in HPC-X.<br>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.<br>**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:<br>• Open MPI updated to v1.10<br>• UPC update to 2.22.0<br>• MXM updated to v3.4.369<br>• 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.<br>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. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

### 14.1.2 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 |

### 14.1.3 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 |

## 14.2 Bug Fixes History

| Internal Reference Number | Issue |
|---|---|
| 4722715/ 4727171 | **Description**: Fixed unnecessary CUDA context initialization in UCC when CUDA is not enabled. |
| | **Keywords**: CUDA; UCC |
| | **Discovered in Release**: 2.25 |
| | **Fixed in Release:** 2.25.1 |
| 4545852 | **Description**: Fixed an issue where CUDA executor kernels failed when requesting an excessive number of registers. |
| | **Keywords**: CUDA; Kernel |
| | **Discovered in Release**: 2.24 |
| | **Fixed in Release:** 2.25 |
| 4602658 | **Description**: Fixed an issue where applications could intermittently hang during `MPI_Finalize` or in the UCC tear-down flow. |
| | **Keywords:** Hang; UCX; MPI; UCC; shutdown |
| | **Discovered in Release:** 2.24 |
| | **Fixed in Release:** 2.24.1 |
| 4173667 | **Description**: Fixed discrepancy in UCX perftest results between client and server. |
| | **Keywords**: UCX; perftest |
| | **Discovered in Release:** 2.21 |
| | **Fixed in Release:** 2.22.1 |
| 4231305/ 4238964 | **Description**: Fixed a statistical issue causing a hang or time-out in ConnectX-8 when using DDP (Dynamic Device Pooling), which is enabled by default. |
| | **Keywords:** DDP; ConnectX-8; time-out |
| | **Discovered in Release:** 2.22.0 |
| | **Fixed in Release:** 2.22.1 |

| Internal Reference Number | Issue |
|---|---|
| 4088373 | **Description**: Fixed an issue where the application could fail with the error: `symbol lookup error: libuct_cuda_gdrcopy.so.0: undefined symbol: gdr_get_info_v2` , caused by a version mismatch in the `gdrcopy` library. |
| | **Keywords:** gdrcopy |
| | **Discovered in Release:** 2.20.0 |
| | **Fixed in Release:** 2.21.0 |
| 4025026 | **Description**: Fixed a FETCH_ADD remote access error for ODP regions. |
| | **Keywords:** Atomic operations; ODP; UCX |
| | **Discovered in Release:** 2.19.0 (UCX 1.17) |
| | **Fixed in Release:** 2.21.0 |
| 3955117 | **Description:** Fixed an issue where a segmentation fault could take place in applications using cuda_ipc transport across multiple UCX contexts, due to incorrect handling of the connectivity map data structure. |
| | **Keywords:** cuda_ipc; segfault |
| | **Discovered in Release:** 2.19.0 |
| | **Fixed in Release:** 2.21.0 |
| 3763160 | **Description:** Fixed an issue where MPI_Init experienced significant delays when there were many files in the /tmp directory. This occurred due to the use of the inotify mechanism for synchronization with a statistics monitoring tool. |
| | **Keywords:** VFS; MPI_Init |
| | **Discovered in Release:** 2.17.0 |
| | **Fixed in Release:** 2.21.0 |
| 3664432 | **Description:** Fixed an issue where a multi-threaded MPI application using its own lock to synchronize MPI calls could experience crashes or data corruption, even when calling MPI_Init_thread with MPI_THREAD_SERIALIZED mode. The problem was caused by incorrect synchronization of the BlueFlame register. |
| | **Keywords:** Data corruption; segfault; crash; multi-thread; MPI_THREAD_SERIALIZED |
| | **Discovered in Release:** Open MPI 4.1 |
| | **Fixed in Release:** 2.21.0 |
| 3819771 | **Description:** Fixed the issue where in certain scenarios, RDMA operations involving CUDA memory could encounter a failure, resulting in the following error: `UCX ERROR ibv_reg_dmabuf_mr(address=0xfff939e00000, length=16, access=0xf) failed: Invalid argument` . |
| | **Keywords:** DMA buffer, memory registration, `ibv_reg_dmabuf_mr` |
| | **Discovered in Release:** 2.19.0 |
| | **Fixed in Release:** 2.21.0 |

| Internal Reference Number | Issue |
|---|---|
| 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. |
| | **Keywords:** Multi-Threaded, Indirect, Key Registration |
| | **Discovered in Release:** 2.17.0 |
| | **Fixed in Release:** 2.18.0 |
| 3837556 | **Description**: Fixed UCX to not create an SRQ on RDMA network devices that do not support it. Before this fix, the application could fail with the error message "ibv_create_srq() failed: Operation not supported". |
| | **Keywords:** SRQ, UCX |
| | **Discovered in Release:** 2.17.0 |
| | **Fixed in Release:** 2.18.0 |
| 3774158 | **Description:** Fixed a failure with the message "Local length error". The issue is caused by some compilers replacing direct assignments with memmove() function, leading to corruption while writing to IO memory. |
| | **Keywords:** UCX, Local length error |
| | **Discovered in Release:** 2.17.0 |
| | **Fixed in Release:** 2.18.0 |
| 3774153 | **Description:** Fixed the issue where **i**n some cases, there could be a race condition between RDMA_WRITE and shared memory write, leading to MPI receiving invalid data with large messages or collective operations between ranks on the same node. |
| | **Keywords:** RDMA_WRITE |
| | **Discovered in Release:** 2.17.0 |
| | **Fixed in Release:** 2.18.0 |
| 3762227 | **Description:** Fixed the issue where the application may crash in UCX remote key packing procedure after failed memory registration. |
| | **Keywords:** UCX, assertion |
| | **Discovered in Release:** 2.17.1 |
| | **Fixed in Release:** 2.18.0 |
| 3748762 | **Description:** Fixed the issue where the application may crash in UCX remote key packing procedure after failed memory registration. |
| | **Keywords:** UCX, assertion |
| | **Discovered in Release:** 2.17.1 |
| | **Fixed in Release:** 2.18.0 |
| 3712109 | **Description**: Fixed UCC error in PyTorch 23.12 from HPC-X 2.17.0 upgrade |
| | **Keywords**: UCC error, PyTorch, Upgrade |

| Internal Reference Number | Issue |
|---|---|
| | **Discovered in Release**: 2.17.0 |
| | **Fixed in Release:** 2.18.0 |
| 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.<br>**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 |
| 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 |

| Internal Reference Number | Issue |
|---|---|
| 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) |
| | **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 |

| Internal Reference Number | Issue |
|---|---|
| | **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:<br>**"Fatal: endpoint reconfiguration not supported yet"**<br>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) |
| 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:<br>**-mca pmix_base_async_modex 1 -mca mpi_add_procs_cutoff 0 -mca pmix_base_collect_data 0**<br>Error messages:<br>• **PMIX ERROR: NOT-FOUND in file server/pmix_server_get.c at line 751**<br>• **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.<br>(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) |

| Internal Reference Number | Issue |
|---|---|
| 2096036 | **Description:** Fixed the issue where the verifier test may have failed with the following error when using the ud_x transport:<br>`ib_mlx5_log.c:139 Local QP operation on mlx5_0:1/IB (synd 0x2 vend 0x68 hw_synd 0/66)`<br>`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.<br>`sock.c:228 UCX ERROR recv(fd=1377) failed: 104`<br>(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) |
| 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.<br>(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) |

| Internal Reference Number | Issue |
|---|---|
| - | **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<br>(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. |
| | **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.<br><br>(Github issue: https://github.com/openucx/ucx/issues/2111) |
| | **Keywords**: UCX, multi-threaded |
| | **Discovered in Version**: 2.1 (UCX 1.3) |

| Internal Reference Number | Issue |
|---|---|
| | **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* ".<br><br>(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:<br>• *"rcache.c:481 UCX WARN failed to register region 0xdec25a0 [0x2b7139ae0020..0x2b7139ae2020]: Input/output error"*<br>• *"ucp_mm.c:105 UCX ERROR failed to register address 0x2b7139ae0020 length 8192 on md[1]=ib/mlx5_0: Input/output error"*<br>• *"ucp_request.c:259 UCX ERROR failed to register user buffer datatype 0x20 address 0x2b7139ae0020 len 8192: Input/output error"*<br><br>(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".<br>For example: osu-micro-benchmarks-5.3.2/osu_get_acc_latency (Latency Test for accumulate with Active/Passive Synchronization).<br><br>(Github issue: https://github.com/openucx/ucx/issues/2454) |
| | **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:<br>*"mpool.c:177 UCX ERROR Failed to allocate memory pool chunk: Out of memory."*<br><br>(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* ". |

| Internal Reference Number | Issue |
|---|---|
| | **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.<br><br>(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.<br>(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 |
| | **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 |

| Internal Reference Number | Issue |
|---|---|
| | **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. |
| | **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 |

| Internal Reference Number | Issue |
|---|---|
| 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 |

# 15 Legal Notices and 3rd Party Licenses

| Product | Version | Legal Notices and 3rd Party Licenses |
|---|---|---|
| HPC-X | 2.26 | • License<br>• 3rd Party Notice<br>• 3rd Party Unify Notice |

## Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. Neither NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: “NVIDIA”) make any representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

NVIDIA