We recommend using HPE's Message Passing Toolkit (MPT) unless there is a strong reason for using another Message Passing Interface (MPI) implementation. HPE MPT is generally more efficient on HPE systems than other MPI libraries.

Benefits of Using HPE MPT

- HPE MPT contains MPI enhancements that are specific to HPE systems. In particular, it offers multiple features for scaling applications to very large process counts.
- The use of MPT on NAS systems is supported by HPE. MPT bugs and issues are tracked closely to provide timely resolution.

Using the Default MPT Version

Typically, there is more than one version of MPT library installed on NAS systems. You can access the currently recommended version by loading mpi-hpe/mpt, which points to the recommended version and environmental settings:

```
% module load mpi-hpe/mpt
```

Note: NAS administrators change the MPT version and environmental settings used by mpi-hpe/mpt from time to time, as needed, without notification. No recompilation or relinking should be needed for your application.

Loading mpi-hpe/mpt instead of a specific MPT module version provides multiple benefits:

- You always get the HPE MPT library with the version and environmental settings recommended by NAS staff.
- You don't have to change your script to load a specific MPT module when a new, improved version becomes available.
- Your script will not fail if an old MPT version is removed from the NAS systems.

If you would like to keep track of which MPT version and settings are being used by your application, you can set the following environmental variables:

```
(for csh, tcsh)
setenv MPI_VERBOSE           (shows MPT version and non-default settings)
setenv MPI_DISPLAY_SETTINGS   (shows all MPT settings)

(for sh, bash)
export MPI_VERBOSE=1
export MPI_DISPLAY_SETTINGS=1
```

In addition, we strongly recommend using the mpiexec launcher, instead of other launchers (such as mpirun) for your MPI application. The mpiexec launcher allows logging of the MPI environment used in your PBS job, such as the MPT version and settings, to the NAS system logfiles. If you notice a change in behavior of your jobs as a result of possible changes in mpi-hpe/mpt, please provide the PBS job ID when requesting help for investigation.

See Porting with HPE MPT for more information.

HPE's Message Passing Interface (MPI) is a component of the HPE Message Passing Toolkit (MPT), a software package that supports parallel programming across a network of computer systems through a technique known as message passing. MPT requires the presence of an Array Services daemon (arrayd) on each host in order to run MPI processes.

HPE's MPT 1.*x* versions support the MPI 1.2 standard and certain features of MPI-2. The 2.*x* versions are fully MPI-2 compliant. Starting with version 2.10, MPT provides complete support for the new MPI 3.0 standard. Recent versions of MPT provide bug fixes and enhancements—such as an extension for the packet retry mechanism to cover more message types, and improvements to the congestion detection and control algorithm—to address InfiniBand stability and scalability issues.

There may be more than one version of MPT on NAS systems. You can access the recommended version by running:

```
module load mpi-hpe/mpt
```

Note that certain environment variables are set or modified when an MPT module is loaded. To see what variables are set when you load mpi-hpe/mpt, complete these steps:

1. Run module show mpi-hpe/mpt to find out which version of MPT the module points to. For example:

   ```
   module show mpi-hpe/mpt
   /nasa/modulefiles/sles12/mpi-hpe/mpt:

   system       logger -p local2.info -t envmodule zsmith display mpi-hpe/mpt
   module-whatis  Loaded recommended version of MPT.
   module       load mpi-hpe/mpt.2.23
   ```

2. Run module show *file_name* to see the environment variables that are set for that version. For example:

   ```
   module show mpi-hpe/mpt.2.23
   ```

You can use the following commands to build an MPI application using HPE MPT:

```
%ifort -o executable_name prog.f -lmpi
%icc -o executable_name prog.c -lmpi
%icpc -o executable_name prog.cxx -lmpi++ -lmpi
%gfortran -I/nasa/hpe/mpt/2.23/include -o executable_name prog.f -lmpi
%gcc  -o executable_name prog.c -lmpi
%g++ -o executable_name prog.cxx -lmpi++ -lmpi
```

TIP: Note that the Fortran 90 USE MPI feature is supported for the ifort command, but not gfortran. Replace USE MPI with include "mpif.h" if you want to use gfortran to compile your Fortran 90 code and link to an HPE MPT library.

HPC-X Message Passing Interface (MPI) is an implementation of Open MPI (based on version 4.x) that includes features and optimizations from Mellanox, the vendor of the InfiniBand networking hardware used in HECC systems. HPC-X MPI offers improved performance for some MPI routines, however, the main purpose we are supporting this implementation is to provide you with an alternative to using the HPE Message Passing Toolkit (MPT).

Some MPI constructs that had been removed from the MPI 3.0 standard—but had remained available for use in deprecated status—were formally removed from OpenMPI-4.x. Our HPC-X MPI installation was built with a flag that makes these deprecated features visible again. For a list of these features, see Removed MPI Constructs on the Open MPI website.

Recommendation: Keep your MPI applications up to date with the current MPI standard.

To use HPC-X MPI on HECC systems, simply load the HPC-X module and your preferred Intel compiler suite, then recompile your application and its dependencies. Launch the newly built application as usual using the mpiexec launcher.

You can safely use the HPC-X build with any of the Intel compiler versions listed using command module avail comp-intel. The following example uses a specific Intel compiler:

```
% module purge
% module load comp-intel/2018.3.222 mpi-hpcx/2.4.0
```

The default process placement strategy is to place ranks round-robin by socket. This approach balances the computational load on the available hardware, but it might not be appropriate for all applications. You can force placement of ranks in linear core order by using the mpiexec --map-by option. For example:

```
% mpiexec -np XX --map-by ppr:YY:socket --bind-to core
```

where *XX* is the total number of ranks, *YY* is the user-specified number of ranks per socket, and the bind-to option is used to pin the ranks to specific cores.

Some workflow changes will be required if you manipulate $PBS_NODEFILE in order to launch multiple MPI jobs within a single PBS job. In this case, do the following:

- In the PBS resource request, include the maximum number of MPI ranks you intend to launch on each node.
- Create a separate rankfile for each invocation of mpiexec, using the following rankfile format:

  ```
  % rank XX=machineName slot=cpuID
  ```

  where *XX* is the MPI rank within the MPI sub-job, *machineName* is one of the names taken from the set provided by PBS (run uniq $PBS_NODEFILE to obtain this set), and *cpuID* is a hardware core ID (for core numbering, see the diagrams for each processor type: Cascade Lake, Skylake, Broadwell, Haswell, Ivy Bridge, Sandy Bridge).

This sample rankfile runs 16 ranks (0-15) evenly distributed on four sockets on two Ivy Bridge nodes (r469i0n10 and r469i0n11):

```
rank 0=r469i0n10 slot=0
rank 1=r469i0n10 slot=1
rank 2=r469i0n10 slot=2
rank 3=r469i0n10 slot=3
rank 4=r469i0n10 slot=10
rank 5=r469i0n10 slot=11
rank 6=r469i0n10 slot=12
rank 7=r469i0n10 slot=13
rank 8=r469i0n11 slot=0
rank 9=r469i0n11 slot=1
rank 10=r469i0n11 slot=2
rank 11=r469i0n11 slot=3
rank 12=r469i0n11 slot=10
rank 13=r469i0n11 slot=11
rank 14=r469i0n11 slot=12
rank 15=r469i0n11 slot=13
```

This article provides some basic information about OpenMPI and some tips for using it on NAS systems.

Background

OpenMPI is an open source Message-Passing Interface (MPI) implementation developed and maintained by a consortium. Its usage has recently been on the rise for various reasons. For example:

- OpenMPI built with CUDA support is included in NVIDIA's software development kit (SDK) for running on the NVIDIA GPUs.

  Note: The HPE Message Passing Tookit (MPT) also provides support for running MPI + CUDA applications on NVIDIA GPUs.

- OpenMPI supports two different MPI IO implementations: ROMIO (developed by Argonne National Labs) and (OpenMPI's native IO implementation). OMPIO is the default for Lustre filesystems starting from OpenMPI versions 4.1.x. Some applications may see better IO performance using OMPIO than ROMIO.

  Note: HPE MPT currently only supports ROMIO.

- For portability across platforms, MPI applications in containers are often built with open source MPI (for example, OpenMPI, MVAPICH) instead of commercial MPI library (such as HPE MPT) for portability across platforms.
- Systems not supported by HPE for evaluating new architectures by NAS staff and pioneer users likely do not include HPE MPT or other commercial MPI libraries in their software stack.

On the other hand, the usability of OpenMPI at NAS is hindered by the following reasons:

- Being open source, there is no dedicated support for OpenMPI at NAS.
- Although OpenMPI's Modular Component Architecture (MCA) provides tremendous flexibility in building and using OpenMPI, it is not as straightforward as HPE MPT.
- When errors occur, sometimes the root causes and solutions are not obvious.

OpenMPI's Modular Component Architecture

The MCA hierarchy includes a series of projects at top, frameworks in the middle and components at the bottom.

Some examples of projects are:

- MPI
- SHMEM (Symmetric Hierarchical MEMory)
- OPAL (Open Portable Access Layer)
- PMIx (Process Management for Exascale Environment)
- PRRTE (PMIx Reference Run-Time Environment) to replace ORTE (Open Run-Time Environment) in OpenMPI 5.0.

Note: An MPI runtime takes care of tasks such as launching the MPI application's processes and helps the MPI library to establish connections between processes.

Under the MPI project, the frameworks that users mostly care about are:

- btl: Byte Transfer Layer
- pml: Point-to-point Messaging Layer
- coll: MPI collective algorithms
- mtl: MPI Matching Transport Layer
- io: MPI I/O

The available components for each framework depend on the OpenMPI version. For example, in OpenMPI 3.x, btl includes components such as:

- self: send-to-self MPI communication
- vader: a shared memory transport supporting local communication
- tcp: TCP/IP communication
- openib: OpenFabrics InfiniBand communication
- smcuda: CUDA-aware shared memory communication

However, note that in OpenMPI 5.x, vader is renamed to sm and openib is removed. For more information about MCA, read this OpenMPI page.

Communication Libraries

The most common communication libraries used by OpenMPI are Libfabric and Unified Communication X (UCX). To build OpenMPI with Libfabric or UCX, use the configure option --with-libfabric=*path-to-libfabric-installation-directory* or --with-ucx=*path-to-ucx-installation-directory*.

Libfabric is also known as Open Fabric Interfaces (OFI) and it supports all networks supported by the OpenMPI openib btl, including InfiniBand, RoCE (RDMA over Converged Ethernet, where RDMA stands for Remote Direct Memory Access), and iWARP (Internet Wide Area RDMA Protocol). Libfabric also supports many other network types, for example, Intel PSM/PSM2 (Performance Scaled

Messaging for using Intel OmniPath Interconnect), Cray uGNI (userspace Generic Network Interfaces), and AWS EFA (Elastic Fabric Adapter).

Note: The openib btl uses the OpenFabrics Alliance (OFA) verbs API stack to support InfiniBand, RoCE, and iWARP devices. The OFA's original name was "OpenIB", which is why the btl is named openib.

UCX is an open source library and it is integrated into OpenMPI as a pml layer. It provides higher-abstraction support for InfiniBand and RoCE and is designed to achieve high performance for HPC applications.

For the InfiniBand network type, the OpenMPI developers suggest using the UCX pml over the openib btl.

Note: On NAS systems, there is no Libfabric installed at system paths. The version of UCX under /usr/bin and /usr/lib64 is very old (1.8.0). If you plan to build your own OpenMPI with either Libfabric or UCX, you will need to build these libraries first. When building OpenMPI to support CUDA (--with-cuda --with-ucx), NVIDIA recommends building UCX with GDRcopy support (--with-gdrcopy).

WARNING: For OpenMPI modules built with --with-ucx, if the UCX was built with --with-xpmem, then during runtime, the XPMEM library used to build UCX must be compatible with the XPMEM library and kernel on the system. Unfortunately, this is not always the case. For example, NVIDIA's HPCX implementation of OpenMPI on NVIDIA systems and distributed to be used at non-NVIDIA sites may include --with-xpmem using an open source XPMEM library. This open source XPMEM is different from HPE's XPMEM on most NAS systems. This incompatibility would lead to some runtime errors similar to:
"....hpcx-2.16/ucx/mt/lib/ucx/libuct_xpmem.so.0 : undefined symbol: xpmem_make ."

Note: XPMEM is a Linux kernel module that enables a process to map the memory of another process into its virtual address space. UCT is a lower-level interface in UCX.

Tips on Using OpenMPI

- **Some OpenMPI modules installed by NAS staff are under /nasa/modulefiles/testing. Add this to your modulefiles search path in order to see all OpenMPI modules.**

  % module use -a /nasa/modulefiles/testing
  % module avail nvhpc
  % module avail openmpi

  Note: NVIDIA provides version 3, 4, and HPCX-enhanced OpenMPI under its HPC SDK distributions. At NAS, they are installed under /nasa/nvidia/hpc_sdk/.../*sdk_version_number*/comm_libs/., where *sdk_version_number* is a number such as 23.11. When you load a modulefile such as nvhpc/23.11, an OpenMPI library is included. To find out which version of OpenMPI is used after you load a nvhpc modulefile, do, for example:

- **module load nvhpc/23.11**
  which mpiexec
  mpiexec --version

  The HPCX-enhanced OpenMPI is usually not set as the default. However, you may find the HPCX related modulefiles under:
  /nasa/nvidia/hpc_sdk/.../*sdk_version_number*/comm_libs/
  ** *CUDA_version_number*/hpcx/latest/modulefiles,**

  where *sdk_version_number* is a number such as 23.11, and *CUDA_version_number* is a number such as 12.3.

- **Use the command ompi_info to learn how an OpenMPI module was built. For example, what version of OpenMPI (3.x, 4.x, 5.x), flavor of the architecture (x86_64, aarch64), what C, C++, Fortran compilers (GNU, Intel, NVIDIA, LLVM), and configure options were used to build it.**

  % module load an_openmpi_module
  % which ompi_info
  % ompi_info (or ompi_info -all)

  Note: In newer releases (23.5 or newer) of OpenMPI 4.x or 4.x + hpcx included in the NVIDIA HPC SDK, the utility ompi_info may not be included in the installation.

Compile Time Tips

- **OpenMPI provides the wrapper scripts** mpicc, mpic++ **(or** mpicxx, mpiCC**) and** mpifort **for compiling and linking MPI applications. The legacy wrappers** mpif77 **and** mpif90 **still exist but are symbolic links to** mpifort**. The compilers used by these wrappers default to the compilers used to build the OpenMPI library. Besides using the command** ompi_info**, one can also find the compilers used to build the MPI library by adding the option** --showme **when invoking the wrapper scripts. That is:**

  ```
  % mpicc --showme
  % mpic++ --showme
  % mpifort --showme
  ```

  Note: OpenMPI in NVIDIA HPC SDK was built by NVIDIA with their nvc, nvc++ **and** nvfortran **compilers.**

- **When building an MPI application written in C, most likely you can use an OpenMPI library built with any C compiler. For example, if the OpenMPI was built with gcc but you want to build your MPI application with clang, you can re-define the variable OMPI_CC and use the** mpicc --showme **command to check that the desired C compiler will be used as shown in the example below:**

  ```
  % module load comp-aocc/4.2.0
  % module load openmpi/4.1.6-toss4-gnu
  % export OMPI_CC=clang
  % mpicc --showme
  clang -I.....
  % mpicc [options] program.c
  ```

- **For C++ or Fortran MPI applications, the same compiler should be used for building both the MPI library and the application. This is discussed in [OpenMPI documentation, section 4.12.4. Installing Multiple Copies of OpenMPI].**

Runtime Tips

- **If the OpenMPI library used was not successfully built with** --with-tm **to support PBS, a cross-node MPI job will need the mpiexec flag** --hostfile $PBS_NODEFILE **or** --machinefile $PBS_NODEFILE **during runtime. That is,**

  ```
  mpiexec -np N --hostfile $PBS_NODEFILE ./a.out
  or
  mpiexec -np N --machinefile $PBS_NODEFILE ./a.out
  ```

  Another option is to use the --host **flag as explained [here].**

- **Before running your MPI application with an OpenMPI module, do a few basic tests as recommended by the OpenMPI developers in [Part 3 of the ABCs of OpenMPI].**
  - **Single-node tests**
    - **To test the basic OpenMPI runtime system:**

      ```
      mpirun -np 1 hostname
      ```

    - **To test the start-up and shut-down of the MPI layer:**

      ```
      mpirun -np 1 hello_c
      ```

    - **To test the MPI communication with more than 1 process:**

      ```
      mpirun -np 2 ring_c
      ```

  - **Cross-node tests**
    - **To test the basic OpenMPI runtime system:**

      ```
      mpirun -np 3 --host host1:1,host2:1,host3:1  hostname
      ```

    - **To test the start-up and shut-down of the MPI layer:**

      ```
      mpirun -np 3 --host host1:1,host2:1,host3:1 hello_c
      ```

    - **To test the MPI communication with more than 1 process:**

      ```
      mpirun -np 3 --host host1:1,host2:1,host3:1 ring_c
      ```

Note: hostname is a Linux command. You can find the source code of hello_c and ring_c in github. host1, host2, host3 refer to the actual hostnames, for example, r705c1t1n[1-3] assigned to a PBS job.

- The mpiexec flag -x PATH -x LD_LIBRARY_PATH may be needed in some situations. For example, when combining OpenMPI with other tools such as the NAS mpiprof profiling tool for running cross-node jobs, use:

```
module load scicon/mpiprof
mpiexec -np N --hostfile $PBS_NODEFILE -x PATH -x \
    LD_LIBRARY_PATH mpiprof [mpiprof-options] ./a.out
```

  Note: The mpiexec command line is shown on two lines due to formatting issues. It should be all on one line.

- OpenMPI provides some process/thread binding capabilities through the use of a three-phase procedure using the flags --map-by <arg>, --rank-by <arg> and --bind-to <arg>.

  - mapping: Assigns a default location to each process.

    The argument for --map-by can be slot, hwthread, core, socket (default), numa, board, node.

  - ranking: Assigns an MPI_COMM_WORKD rank value to each process

    The argument for --rank-by can be slot (default), hwthread, core, socket, numa, board, node.

  - binding: Constrains each process to run on specific processors.

    For --bind-to, the possible arguments are none, hwthread, core, l1cache, l2cache, l3cache, socket, numa, board, cpu-list. Among them, none is the default when oversubscribed, core is the default when np <= 2, and socket is the default when np > 2.

  In addition, OpenMPI provides the flag --report-bindings to visualize bindings.

  Experiment with the --map-by, --rank-by, --bind-to, and --report-bindings options to get a sense of how these options work.

  For instance, running a.out with N=4 on 1 sky_ele node (2 sockets, each with 20 physical cores, or 40 logical cores when Hyperthreading is turned on in BIOS),

  - --map-by socket --rank-by core --bind-to core --report-bindings yields:

    ```
    rank 0 bound to socket 0 [core 0[hwt 0-1]]
    rank 1 bound to socket 0 [core 1[hwt 0-1] ]
    rank 2 bound to socket 1 [core 20[hwt 0-1]]
    rank 3 bound to socket 1 [core 21[hwt 0-1]]
    ```

  - --map-by core --rank-by core --bind-to core --report-bindings yields:

    ```
    rank 0 bound to socket 0 [core 0[hwt 0-1]]
    rank 1 bound to socket 0 [core 1[hwt 0-1]]
    rank 2 bound to socket 0 [core 2[hwt 0-1]]
    rank 3 bound to socket 0 [core 3[hwt 0-1]]
    ```

  You can also use the NAS inhouse mbind.x tool for binding. For example, to use mbind.x to spread the MPI processes among the physical cores in each 2-socket node, do:

  ```
  mpiexec -np N [mpiexec-options] /u/scicon/tools/bin/mbind.x -cs ./a.out
  ```

  This yields the following binding scheme with N=4 on 1 sky_ele node:

  ```
  host: xxx, ncpus: 80, rank: 0 (r0), bound to cpu: 0
  host: xxx, ncpus: 80, rank: 1 (r1), bound to cpu: 10
  host: xxx, ncpus: 80, rank: 2 (r2), bound to cpu: 20
  host: xxx, ncpus: 80, rank: 3 (r3), bound to cpu: 30
  ```

  Note: The mbind.x tool developer recommends to use the OpenMPI's own binding capabilities instead of mbind.x in case of bugs or incompatibilities.

- For better control of the number of MPI ranks to run on each node, use one of the choices below. This is useful in situations where there are more slots availble in each node than what you want to use. For example, in a batch job with 2 nodes each with 40 cores, if you

want to run just 8 MPI ranks across 2 nodes each with 4 MPI ranks, do:

- Use the --host or -H option: (all one line)

```
mpirun -np 8 --host host1:4,host2:4 --hostfile $PBS_NODEFILE -x
  PATH -x LD_LIBRARY_PATH ./a.out
```

  where host1 and host2 are actual hostnames such as r101i5n0 and r101i5n1. You can find the hostnames of nodes assigned to your job by:

```
cat $PBS_NODEFILE | uniq
```

- Use the -npernode option (all one line):

```
mpirun -np 8 -npernode 4 --hostfile $PBS_NODEFILE -x
  PATH -x LD_LIBRARY_PATH ./a.out
```

- Use the -N option (all one line):

```
mpirun -np 8 -N 4 --hostfile $PBS_NODEFILE -x
  PATH -x LD_LIBRARY_PATH ./a.out
```

- Use the --map-by option (all one line):

```
mpirun -np 8 --map-by ppr:4:node --hostfile $PBS_NODEFILE -x
  PATH -x LD_LIBRARY_PATH ./a.out
```

- The inter-node network of NAS systems mainly use InfiniBand. The OpenMPI modules installed by NAS staff should have included support for InfiniBand. For example, if the OpenMPI was built with UCX, then UCX should automatically detect the presence of InfiniBand and will use it by default. If it was not built with UCX, then likely, it was built with support of openib in the btl framework. In this case, openib will also detect the presence of InfiniBand.

  Although not recommended, you can still control whether or how to use InfiniBand by adding --mca flags. For example,

```
mpiexec -np N --mca pml ucx --mca btl ^openib ./a.out
mpiexec -np N --mca btl self,vader,tcp ./a.out
```

  Note: ^openib means not to use openib for communication. TCP (Transmission Control Protocol) is a communication standard that allows devices to exchange messages over a network. Performance using TCP is slower than using InfiniBand RDMA (remote direct memory access).

- Starting in OpenMPI 5.0.0, you can view which networks (such as sm, uct) are used during runtime between different pairs of nodes by adding the flag --mca ompi_display_comm 1. That is,

```
mpiexec -np N --hostfile $PBS_NODEFILE --mca_display_comm 1 ./a.out
```

  This will generate a communication matrix to stdout.

- If your application is built with a recent OpenMPI version that uses UCX and experiences large run-to-run variations and/or degraded performance in communication time, compared to an older OpenMPI version or other MPI implementations (for example, HPE MPT), consider adding the UCX environment variables UCX_PROTO_ENABLE=n and UCX_GVA_ENABLE=n as follows:

```
mpiexec -np .. -x UCX_PROTO_ENABLE=n -x UCX_GVA_ENABLE=n ...
```

  Adding these two variables can help applications including LAMMPS, NPB FT, and IS benchmarks.

  For NPB FT, adding another variable UCX_IB_MLX5_DEVX=no also helps.

  For more information, read Known Issues: 3606445 (discovered in HPCX 2.17.0) and 3884209 (discovered in HPCX 2.19.0) on the use of UCX_PROTO_ENABLE=n; 4050321 (discovered in HPCX 2.21.0) on the use of UCX_GVA_ENABLE=n; and 4026461 (discovered in HPCX 2.20.0) on the use of UCX_IB_MLX5_DEVX=no.

- To explicitly choose which MPI IO implementation to use, add the flag --mca io .... For example,

```
mpiexec -np N --mca io ompio ./a.out
or
mpiexec -np N --mca io romio314 ./a.out
```

References

- **OpenMPI: Open Source High Performance Computing**
- **Slides and Videos of "What is MPI? What is OpenMP?" and "The ABCs of OpenMPI, parts 1 - 3"**
- **Open-MPI-SC23-BOF.pdf**

Cray Message Passing Toolkit (CMPT) is a collection of high-performance software libraries within the Cray Programming Environment (CPE) that enables parallel applications to communicate and exchange data efficiently across nodes in a Cray system with Slingshot interconnect, such as Athena which uses Slingshot 11. CMPT includes MPICH, PMI, OpenSHMEMX, DSMML, and MPIxlate. CPE version 25.09 includes MPICH 8.1.33 and 9.0.1, PMI 6.1.16, OpenSHEMEMX 11.8.0, DSMML 0.3.1, and MPIxlate 1.0.9.

Note: MPICH 8.1.33 implements the MPI-3.1 standard. MPICH 9.0.1 includes support for MPI 4.0 standard features. MPICH 8.1.33 is the current default on Athena.

As the name suggests, Cray MPICH is based on MPICH. If your existing executable was built with a dynamically linked MPI library that is Application Binary Interface (ABI)-compatible with the Cray MPICH library, it might be able run on Athena without recompiling and linking with the Cray MPICH library. For example, Intel MPI is also based on MPICH, and is ABI-compatible with the Cray MPICH library. On the other hand, OpenMPI and HPE/SGI MPT library are not ABI-compatible with MPICH.

This article provides brief information about using the CMPT modules cray-mpich, cray-mpich-abi and cray-mpixlate to build or prepare an MPI executable, together with the cray-pals module from the Cray Parallel Application Launch Service (PALS) to manage job startup and process launch on Athena.

Building Applications with the cray-mpich Module

There are three methods of using cray-mpich to build an application.

Method 1: Explicitly Invoking the Compiler and Linking to the Cray MPICH Library

To explicitly build an executable using the cray-mpich module and a compiler from either PrgEnv-cray, PrgEnv-intel, PrgEnv-aocc or PrgEnv-gnu:

```
module load PrgEnv-cray  (or PrgEnv-intel, PrgEnv-aocc, PrgEnv-gnu)
module load cray-mpich   (or cray-mpich/9.0.1)

name_of_compiler_driver <compiler flags> application_source_file.[f90,c,cpp]
-I/path-to-include-directory-of-other-library
-L/path-to-lib-directory-of-other-library -lname_of_other_library
-I${MPICH_DIR}/include
-L${MPICH_DIR}/lib -lmpi   (for C/C++ applications)
or
-L${MPICH_DIR}/lib -lmpi -lmpifort (for Fortran applications)
```

The name of the compiler driver varies as follows:

```
PrgEnv-cray : crayftn, craycc, crayCC
PrgEnv-intel: ifort, icc, icpc, ifx, icx, icpx
PrgEnv-aocc : flang, clang, clang++
PrgEnv-gnu  : gfortran, gcc, g++
```

The ${MPICH_DIR} can be replaced with ${CRAY_MPICH_PREFIX} or ${CRAY_MPICH_DIR}, since all three of them are defined in cray-mpich and they point to the same directory.

Depending on which PrgEnv is used {-cray, intel, aocc, gnu} and whether cray-mpich/8.1.33 or cray-mpich/9.0.1 are loaded, the actual path of ${MPICH_DIR} varies. For example, if using PrgEnv-cray and cray-mpich/8.1.33, then ${MPICH_DIR} is /opt/cray/pe/mpich/8.1.33/ofi/cray/20.0.

Note: Athena is an HPE Slingshot 11 system that only supports libfabric — a low-level communication library also known as the OpenFabrics Interfaces (OFI). The Unified Communication X (UCX) framework for communication is supported in Slignshot 10 systems, but not Slingshot 11 systems with Cray MPICH.

Method 2: Using the Cray MPICH Wrapper Scripts

Cray MPICH also provides standard MPI wrappers (mpicc, mpicxx, mpic++, mpifort, mpif77, mpif90) for invoking compiler and linking the proper Cray MPICH library in accordance with the compiler that is used (cray, intel, aocc, gnu).

Note: mpic++ is a soft link to mpicxx. mpif77 and mpif90 are soft links to mpifort.

Usage:

```
% module load PrgEnv-cray (or PrgEnv-intel, PrgEnv-aocc, PrgEnv-gnu)
% module load cray-mpich  (if not already loaded)
% mpifort <FFLAGS> source.f90
% mpicc <CFLAGS> source.c
% mpicxx <CXXFLAGS> source.cpp
```

If other non-MPI libraries are needed for building the application, they need to be explicitly added to the mpifort, mpicc, mpicxx command line as shown in Method 1.

When using the MPI wrappers, the variables MPICH_FC, MPICH_CC and MPICH_CXX can be used to change the choice of compilers.

For example, with PrgEnv-intel, mpifort, mpicc, and mpicxx default to using ifx, icx and icpx regardless of which of intel, intel-classic, intel-oneapi you have loaded. If you want mpifort, mpicc, and mpicxx to use ifort, icc, icpc, do the following before invoking mpifort, mpicc or mpicxx:

With csh:

```
% setenv MPICH_FC ifort
% setenv MPICH_CC icc
% setenv MPICH_CXX icpc
```

With bash:

```
% export MPICH_FC=ifort
% export MPICH_CC=icc
% export MPICH_CXX=icpc
```

Method 3: Using Cray's Compiler/Link Wrapper Scripts

When a cray-mpich module is loaded together with a compiler provided under either PrgEnv-cray, PrgEnv-intel, PrgEnv-aocc, or PrgEnv-gnu, the Cray wrappers ftn, cc, CC will automatically link the Cray MPICH library (-lmpich or simply -lmpi for C/C++ and -lmpi -lmpifort for Fortran) and other Cray libraries (for example, Cray Scientific Library) to ensure compatibility and optimized performance.

The actual -lmpi and -lmpifort linked in depend on the compiler used. For example, if a Cray compiler (crayftn, craycc, crayCC) is used, then -lmpi_cray and -lmpifort_cray are linked in.

Usage:

```
% module load PrgEnv-cray (or PrgEnv-intel, PrgEnv-aocc, PrgEnv-gnu)
% module load cray-mpich  (if not already loaded)
% ftn <FFLAGS> source.f90
% cc <CFLAGS> source.c
% CC <CXXFLAGS> source.cpp
```

If your application needs other libraries not included in the Cray Programming Environment, you must explicitly add them to the ftn, cc, CC command line as shown in Method 1.

Here are a few useful Cray MPICH and LIBFABRIC environment variables to consider:

- **MPICH_ENV_DISPLAY=1**: Causes rank 0 to display all MPICH environment variables and their current settings at MPI initialization time.
- **MPICH_VERSION_DISPLAY=1**: Causes MPICH to display the HPE Cray MPI version number as well as build date information; helps to track issues with different MPI versions.
- **MPICH_RANK_REORDER_DISPLAY=1**: Causes rank 0 to display which node each MPI rank resides in. The rank order can be manipulated via the MPICH_RANK_REORDER_METHOD environment variable.
- **MPICH_MEMORY_REPORT=*n***:
  - *n*=1: prints a summary of the min/max high water mark and associated rank to stderr.
  - *n*=2: outputs each rank's high water mark to a file as specified using MPICH_MEMORY_REPORT_FILE.
  - *n*=3: does both 1 and 2.
- **MPICH_OFI_VERBOSE=1**: Helps for debugging by displaying more verbose output during MPI_init to verify which libfabric provider has been selected, along with the name and address of the NIC being used.
- **MPICH_OFI_USE_PROVIDER**: On Slighshot 11 systems, the default is the cxi provider. Other libfabric providers (such as "sockets") can be requested for debugging purposes.

  Note: cxi stands for Cassini fabric provider in libfabric.
- **FI_PROVIDER**: Controls what providers to load.
- **FI_LOG_LEVEL=warn & FI_LOG_PROV=cxi**: Causes the Cassini provider to log messages which help to debug issues.
- **FI_CXI_RX_MATCH_MODE**: Specifies the receive message matching mode to be utilized; possible choices are hardware, software, hybrid.

  Set this variable to hybrid or software if your application fails to run and the error message includes a recommendation to set this variable.
- **MPICH_OFI_NIC_MAPPING**: Specifies the precise rank-to-NIC mapping to use on each node. This is evaluated only if the MPICH_OFI_NIC_POLICY variable is set to USER.
- **MPICH_OFI_NIC_POLICY**: Selects the rank-to-NIC assignment policy used by Cray MPI. Each MPI rank will be assigned to exactly one NIC. There are five available options: [BLOCK | ROUND-ROBIN | NUMA | GPU | USER].
- **MPICH_OFI_NIC_VERBOSE**: If set to 1, verbose information pertaining to NIC selection is printed at the start of the job.

- **MPICH_OFI_NUM_NICS**: Specifies the number of NICs the job can use on a per-node basis.

For more information, read **man intro_mpi** and the libfabric **man fabric** and **man fi_cxi**.

Running Applications with the cray-pals Module

Loading a cray-mpich module provides access to scripts such as mpifort, mpif77, mpif90, mpicc, mpicxx, mpic++ as well as various libmpi* for building MPI applications. However, cray-mpich does not provide mpiexec for use during runtime. A cray-pals module is needed during runtime to access the mpiexec executable in the parallel application launch service (PALS).

Note: Make sure cray-pals is loaded before launching your application with mpiexec.

Usage:

```
% module load cray-pals    (if not already loaded)
% mpiexec [options] your_executable
```

Here are some useful mpiexec options to consider:

- **-n, -np, --np:** Number of processes to start. Default is 1.
- **-ppn, --ppn:** Processes/ranks per compute node.
- **-hostfile, --hostfile:** File containing list of host names. Default for <*hostfile*.txt> is $PBS_NODEFILE. For example:

  ```
  mpiexec [-hostfile|--hostfile] <hostfile.txt> <application>
  ```

- **--cpu-bind [verbose],[core,depth,none,numa,socket,core,thread,list,mask]:** Bind ranks to CPUs. Default is thread. For example:

  ```
  mpiexec --cpu-bind core -n 8 <application>
  ```

- **--mem-bind [none, local, list, mask]:** Bind ranks to NUMA nodes.
- **-env** or **--env:** Set an environment variable. For example:

  ```
  mpiexec [-env|--env]  <ENV_VAR_NAME>="ENV_VAR_VALUE" <aplication>
  ```

- **-l, --label:** Add node and rank labels output.
- **--line-buffer** or **--no-line-buffer:** Line buffered mode for stdio, to avoid interleaving lines from multiple ranks. Default is --no-line-buffer to allow fully buffered stdio.
- **--restart:** mpiexec to restart application if it detects that an application aborted. Default value for --restart is 16.
- **-v, --verbose:** More verbose output.

Running Applications Built with Intel MPI Without Rebuilding the Executable

An existing executable built with the Intel MPI library might be able to run on Athena without having to rebuild it with the Cray MPICH library. To try this, use the cray-mpich-abi module. For example:

1. Load all modules and/or set any variables required to run the application, except the Intel MPI module.
2. Load the Cray MPICH module cray-mpich-abi instead of cray-mpich.
3. Load the cray-pals module.
4. Launch the application.

Note: If the existing executable was built with an Intel MPI version older than version 5.0 (released around 2014), use cray-mpich-abi-pre-intel-5.0 instead of cray-mpich-abi.

For example:

```
% module switch PrgEnv-cray PrgEnv-intel
   (if you executable needs some libraries from Intel compiler)

% setenv LD_LIBRARY_PATH \
    "/nasa/intel/oneapi/mkl/2023.2.0/lib/intel64:$LD_LIBRARY_PATH"
    (if MKL is needed. Note that the command line is too long to be formatted
as one line, so it is broken with a backslash [\].)
% module switch cray-mpich cray-mpich-abi
% module load cray-pals
% mpiexec [options] executable-built-with-intel-mpi
```

WARNING: Your existing executable may not be able to achieve the best performance that the Athena cluster can offer. For example, an executable built with Intel compiler on other NAS systems and/or linked with the Intel MKL library likely does not support the AVX512 capability of the AMD Turin processor. For best performance, consider rebuilding your application and linking with library

such as the Cray Scientific Library or the AMD Optimizing CPU Libraries (AOCL) instead of the Intel MKL library.

Translating Executables Built with OpenMPI or HPE/SGI MPT to Cray MPICH

MPIxlate, which stands for MPI Translate, enables C and C++ applications compiled using an MPI library that is not binary compatible with Cray MPICH to be run without recompilation on supported HPE Cray systems.

WARNING: MPIxlate does not provide support for Fortran applications.

MPIxlate transparently translates OpenMPI ABI and HPE/SGI MPT ABI to MPICH ABI at runtime.

Usage:

1. Load all modules and/or set any variables required to run the application, except the OpenMPI or HPE/SGI MPT module.
2. Load the Cray MPICH module `cray-mpich`.
3. Load the MPIxlate module `cray-mpixlate`.
4. Load the `cray-pals` module.
5. Launch the application using MPIxlate as a pass-through intermediary.

For example, to translate an application built with OpenMPI with an ABI major number 40 (`libmpi.so.40`) to Cray MPICH with an ABI major number 12 (`libmpi_cray.so.12`), do:

```
% mpiexec [mpiexec options] mpixlate -s ompi.40 -t cmpich.12 a.out ...
```

To translate an application built with HPE/SGI MPT 2.25 to Cray MPICH with an ABI major number 12, do:

```
% mpiexec [mpiexec options] mpixlate -s sgi.225 -t cmpich.12 a.out ...
```

Note: With MPIxlate 1.0.9, only `ompi.40` and `sgi.225` are supported. If your C/C++ executable was built with HPE/SGI MPT 2.30, try `-s sgi.225`.

For more information about MPIxlate and its limitations, see **man mpixlate**.