Among the Message Passing Interface (MPI) libraries installed on Aitken, Electra, and Endeavour, we recommend using HPE's Message Passing Toolkit (MPT) library. Use the command `module avail mpi-hpe` to determine which MPT versions are available. You can always load the default MPT version by running `module load mpi-hpe/mpt`.

Be sure to load the module in both of the following places:

- On the front-end node when you build your application.
- In the PBS script you use to run jobs on the compute nodes.

Network Considerations

NAS systems use an InfiniBand (IB) network for inter-process remote direct memory access (RDMA) communications. There are two InfiniBand fabrics, designated as ib0 and ib1. In order to maximize performance, we recommend using the ib0 fabric for all MPI traffic. The ib1 fabric is reserved for storage-related traffic. The default configuration for MPT on NAS systems is to use only the ib0 fabric, although the MPI_IB_RAILS environment variable is set to 1+ to allow failover to the ib1 fabric if there are errors using ib0.

TIP: Loading `mpi-hpe/mpt` instead of an explicit MPT modulefile allows you to always get the NAS-recommended MPT version and environmental variable settings that enable improvements in InfiniBand stability and scalability.

Environment Variables

When you load an MPT module, several paths (such as CPATH and LD_LIBRARY_PATH) and MPT-related environment variables are reset or modified to non-default values. You can also set other environment variables that may be useful for some applications or for debugging purposes.

Modified Environment Variables

The variables that are modified when you load a module might vary by module version. To find out what environmental variables are set by the modulefile, use the `module show` command as follows:

```
% module show mpt_modulefile_name
```

where *mpt_modulefile_name* refers to an explicit MPT modulefile that `mpi-hpe/mpt` points to. Some of the variables that are likely to be set to a non-HPE-default value include:

```
MPI_IB_TIMEOUT
MPI_IB_RAILS
MPI_IB_FAILOVER
MPI_IB_RECV_MSGS
MPI_IB_RNR_TIMER
MPI_SYSLOG_COPY
MPI_UD_RECV_MSGS
MPI_UD_TIMEOUT
MPI_WATCHDOG_TIMER
MPIO_LUSTRE_GCYC_MIN_ITER
```

The descriptions of these variables and their HPE default values are as follows:

MPI_IB_TIMEOUT
: When an IB card sends a packet, it waits some amount of time for an acknowledgement (ACK) packet to be returned by the receiving IB card. If it does not receive one, it sends the packet again. This variable controls the wait period. The time spent is equal to 4 * 2 ^ MPI_IB_TIMEOUT microseconds.
(HPE) Default: 18

MPI_IB_RAILS
: If the MPI library uses the IB driver as the inter-host interconnect, it will by default use a single IB fabric. If this variable is set to 2, the library will try to make use of multiple available separate IB fabrics (ib0 and ib1) and split its traffic across them. If the fabrics do not have unique subnet IDs, then the rail-config utility is required to have been run by the system administrator to enable the library to correctly use the separate fabrics. If this variable is set to 1+, then MPT will set up IB connection on two rails but only send traffic on one; the second rail will be reserved for failover in case of errors.
(HPE) Default: 1

MPI_IB_FAILOVER
: When the MPI library uses IB and a connection error is detected, the library will handle the error and restart the connection a number of times equal to the value of this variable. Once there are no more failover attempts left and a connection error occurs, the application will be aborted.
(HPE) Default: 0

MPI_IB_RECV_MSGS
: When the MPI library uses IB it must allocate some number of buffers to receive data headers and short messages into. If a rank has all its buffers filled before it can service them, it may go into an error mode. If this happens, try doubling the value of this variable.
(HPE) Default: 512

MPI_IB_RNR_TIMER
: When a packet arrives at an InfiniBand HCA and there are no remaining receive buffers for it, the receiving HCA sends a NAK to the requestor. The requesting HCA retries again after some period of time. This variable controls the delay time. Setting a

higher value may slow performance in some circumstances, but it will also likely significantly help fabric health during high congestion. See Table 45 of the official InfiniBand specification for precise translations of this value to delay times.
(HPE) Default: 14

MPI_SYSLOG_COPY
When this variable is set, messages about MPT internal errors and system failures are copied to the system log on the machines where they happened. If this variable is set to "2" then MPT internal warnings will also be copied to the syslog.
(HPE) Default: Not set

MPI_UD_RECV_MSGS
This controls the number of InfiniBand UD receive buffers.
(HPE) Default: 800

MPI_UD_TIMEOUT
The IB UD code uses this variable as a timeout to control when to resend UD packets if it has not received some form of ACK from the receiver. The variable is in units of milliseconds.
(HPE) Default: 20

MPI_WATCHDOG_TIMER
If an MPT process has been having continuous IB fabric problems and has not been able to contact another process for this many minutes, then it will abort the application. By default MPT keeps trying to make transfers even in the face of flaky fabrics. This variable may help abort the job sooner if a node has crashed. Set to "0" to disable.
(HPE) Default: 10

MPIO_LUSTRE_GCYC_MIN_ITER
Enables group-cyclic aggregator assignment. Group-cyclic allows multiple aggregators to write to each OST while minimizing lock contention. This value specifies the minimum number of stripes that each aggregator will write so as to ensure the added lock contention of more aggregators is offset by the added bandwidth of more data written per two-phase iteration. A value of "1" will allow assignment of the maximum possible number of aggregators.
(HPE) Default: 0 (not enabled)

See **man MPI** for more information.

Additional Environment Variables

For more MPT-related variables, read **man MPI** after you load an MPT module. Some variables may be useful for certain applications or for debugging purposes. Here are a few of them for you to consider:

MPI_DSM_DISTRIBUTE
Activates NUMA job placement mode. This mode ensures that each MPI process gets a unique CPU and physical memory on the node with which that CPU is associated. This feature can also be overridden by using dplace or omplace. This feature is most useful if running on a dedicated system or running within a cpuset.
Default: Enabled

MPI_MEMMAP_OFF
Turns off the memory mapping feature, which provides support for single-copy transfers and MPI-2 one-sided communication on Linux. These features are supported for single-host MPI jobs and MPI jobs that span partitions. At job startup, MPI uses the xpmem module to map memory from one MPI process onto another. The mapped areas include the static region, private heap, and stack. Memory mapping is enabled by default on Linux. To disable it, set the MPI_MEMMAP_OFF environment variable.
Default: Not enabled

MPI_BUFS_PER_PROC
Determines the number of private message buffers (16 KB each) that MPI is to allocate for each process. These buffers are used to send and receive long messages. Each process has this many buffers for sending with and this many for receiving with. The memory consumed on a shepherd is #ranks_in_the_shepherd * MPI_BUS_PER_PROC * 2 * 16k.
Default: 128 buffers (1 page = 16KB)

MPI_COREDUMP
Controls which ranks of an MPI job can dump core on receipt of a core-dumping signal. Valid values are NONE, FIRST, ALL, or INHIBIT.
NONE means that no rank should dump core.
FIRST means that the first rank on each host to receive a core-dumping signal should dump core.
ALL means that all ranks should dump core if they receive a core-dumping signal.
INHIBIT disables MPI signal-handler registration for core-dumping signals.
Default: FIRST

MPI_STATS (toggle)
Enables printing of MPI internal statistics. Each MPI process prints statistics about the amount of data sent with MPI calls during the MPI_Finalize process.
Default: Not enabled

MPI_DISPLAY_SETTINGS
If set, MPT will display the default and current settings of the environment variables controlling it.
Default: Not enabled

MPI_VERBOSE
Setting this variable causes MPT to display information such as what interconnect devices are being used and what environment variables have been set by the user to non-default values. Setting this variable is equivalent to passing mpirun the -v option.
Default: Not enabled

MPI_IB_XRC
When MPT is creating IB connections lazily and using IB host channel adapters (HCAs) that support extended reliable connection (XRC) queue pairs (QPs), MPT will use XRC QPs instead of regular RC QPs. XRC QPs are a more scalable version of the RC QPs currently found on ConnectX IB HCAs.
Default: Enabled

MPI_IB_NUM_QPS
When InfiniBand RC QPs are being allocated lazily, this limits the number of QPs that any one rank may use. This helps to

prevent the host from completely running out of QPs.
Default: (Maximum number of QPs the HCAs can support - 2k) / # of local ranks

MPI_IB_QTIME
MPT has the ability to use the controlled delay (CoDel) traffic throttling mechanism with Reliable Connection (RC) and User Datagram (UD) InfiniBand protocols. Cluster nodes have the ability to saturate InfiniBand fabrics to the state that they become backed up, start dropping packets, and break all sorts of services like TCP/IP control channels and Lustre. If enabled, MPT will monitor the approximate time that outgoing transfers are queued up at the HCA. If queue time exceeds the value of this variable, then MPT will take measures to significantly reduce the amount of traffic that it is putting onto the fabric for a short period of time. In many cases, this will eventually reduce fabric loading to a reasonable level. This variable is in units of microseconds (us); for example, a value of "200" will cause throttling to begin if minimum queue time exceeds 200us. Suggested values to try are in the range of 100us - 5000us.
Default: Disabled

Building Applications

To build MPI applications with the HPE MPT library, link with -lmpi and/or -lmpi++. See HPE MPT for some examples.

Running Applications

MPI executable files built with HPE MPT are not allowed to run on any front-end nodes such as the PFEs, AFEs, etc.

You can run your MPI job on the compute nodes in an interactive PBS session or through a PBS batch job. After loading an MPT module, use the mpiexec command to start your MPI processes (do not use mpirun).

For example:

```
#PBS -lselect=1:ncpus=128:mpiprocs=128:model=mil_ait
....
module load mpi-hpe/mpt
mpiexec -np N ./your_executable
```

The -np flag (with *N* MPI processes) can be omitted if the value of *N* is the same as the product of the value specified for select and the value specified for mpiprocs.

Performance Considerations

If your MPI job uses all of the cores in each node (28 processes/node for Broadwell, 40 processes/node for Skylake and Cascade Lake, and 128 processes/node for Rome and Milan), then pinning MPI processes greatly helps the performance of the code.

By default, the environment variable MPI_DSM_DISTRIBUTE is set to 1 (or true), which will pin processes consecutively on the cores.

If your MPI job does *not* use all the cores in each node, we recommend that you disable MPI_DSM_DISTRIBUTE as follows:

```
setenv MPI_DSM_DISTRIBUTE 0
```

Then, let the Linux kernel decide where to place your MPI processes. If you want to pin processes explicitly, you can try using one of the methods described in Process/Thread Pinning Overview.

The following recommended process pinning method uses the mbind.x tool to pin an eight-MPI-process job with every four processes on four processor cores of a node, using two nodes:

```
#PBS -lselect=2:ncpus=4:mpiprocs=4:model=xxx
mpiexec -np 8 mbind.x ./your_executable
```

Building Applications

To build an MPI/OpenMP hybrid executable using HPE's MPT and Intel's OpenMP libraries, you must compile your code with the -openmp flag and link it with the -lmpi flag. For example:

```
%module load comp-intel/ mpi-hpe/mpt
%ifort -o your_executable prog.f -openmp -lmpi
```

Running Applications

Here is a sample PBS script for running an MPI/OpenMP application using 3 Skylake nodes, with 20 MPI processes on each node and 2 OpenMP threads per MPI process.

```
#PBS -lselect=3:ncpus=40:mpiprocs=20:model=sky_ele
#PBS -lwalltime=1:00:00

module load comp-intel/ mpi-hpe/mpt
setenv OMP_NUM_THREADS 2

cd $PBS_O_WORKDIR

mpiexec ./your_executable
```

You can specify the number of threads on the PBS resource request line using ompthreads. This causes the PBS prologue to set the OMP_NUM_THREADS environment variable.

```
#PBS -lselect=3:ncpus=40:mpiprocs=20::ompthreads=2:model=sky_ele
#PBS -lwalltime=1:00:00

module load comp-intel/ mpi-hpe/mpt

cd $PBS_O_WORKDIR

mpiexec ./your_executable
```

Performance Issues

For pure MPI codes built with HPE's MPT library, you can improve performance on Broadwell, Skylake, Cascade Lake, Rome, and Milan nodes by pinning the processes through setting MPI_DSM_DISTRIBUTE environment variables to 1 (or true). However, for MPI/OpenMP codes, all the OpenMP threads for the same MPI process have the same process ID. In this case, setting this variable to 1 causes all OpenMP threads to be pinned on the same core and the performance suffers.

It is recommended that you set MPI_DSM_DISTRIBUTE to 0 and use omplace for pinning instead.

If you use Intel version 10.1.015 or later, you should also set KMP_AFFINITY to *disabled* or OMPLACE_AFFINITY_COMPAT to ON as Intel's thread affinity interface would interfere with dplace and omplace.

```
#PBS -lselect=3:ncpus=40:mpiprocs=20::ompthreads=2:model=sky_ele
#PBS -lwalltime=1:00:00

module load comp-intel/ mpi-hpe/mpt

setenv MPI_DSM_DISTRIBUTE 0
setenv KMP_AFFINITY disabled

cd $PBS_O_WORKDIR

mpiexec -np 60 omplace ./your_executable
```

You can use the Intel MPI library to build and run your MPI application. The Intel MPI modules available on NAS resources are:

- mpi-intel/2018.3.222
- mpi-intel/2019.5.281
- mpi-intel/2020.0.166

For more information, see Modules.

Compiling Your Applications

Before you compile an application, load an Intel compiler module and an Intel MPI module. Ensure that no other MPI module (MPT or MVAPICH2) is loaded. For example:

```
% module purge
% module load mpi-intel/2020.0.166
% module load comp-intel/2020.4.304
```

Compile your application by running either the mpiifort script to invoke the ifort compiler or the mpiicc script to invoke the icc compiler. For example:

```
% mpiifort -o your_executable program.f
```

To view more information about the scripts, use the help option: mpiifort -help or mpiicc -help.

Running Your Applications

The following script provides an example of using the Hydra process manager command. You can replacempiexec.hydra with mpiexec as the two binaries are identical.

```
## For applications that run the same number of
## MPI processes per node, use, for example
#PBS -lselect=4:ncpus=28:model=....

## If rank 0 needs all the memory in the
## first node, use, for example
#PBS -lselect=1:ncpus=1:model=...+3:ncpus=28...
..
module load mpi-intel/2020.0.166
module load comp-intel/2020.4.304    # or another Intel compiler module

...

# TOTAL_CPUS below represents value for the current job,
# it is to be replaced by an integer literal or
# by some variables that you set yourself

mpiexec.hydra -machinefile $PBS_NODEFILE -np TOTAL_CPUS your_executable
```

To view more mpiexec.hydra options, use the help option: mpiexec.hydra -help.

For more information about using Intel MPI libraries, see the Intel MPI Library Documentation website.

Building Applications

To build an OpenMP application, you need to use the -qopenmp Intel compiler flag:

%module load comp-intel
%ifort -o *your_executable* -qopenmp program.f

Running Applications

The maximum number of OpenMP threads that an application can use on a compute node depends on:

- The number of physical processor cores in the node
- Whether hyperthreading is available and enabled

With hyperthreading, the operating system views each physical core as two logical processors, and can therefore assign two threads per core. This is beneficial only when one thread does not keep the functional units in the core busy all the time and can share the resources in the core with another thread. Running in this mode may take less than twice the wall time compared to running only one thread on the core.

Hyperthreading technology is available and enabled on all Electra and Aitken processor types. See the following table for the maximum number of threads possible on each processor type:

Processor Type	Maximum Threads without Hyperthreading	Maximum Threads with Hyperthreading
Broadwell	28	56
Skylake	40	80
Cascade Lake	40	80
Rome	128	256
Milan	128	256

TIP: Hyperthreading does not benefit all applications. Also, some applications may show improvement with some process counts but not with others, and there may be other unforeseen issues. Therefore, before using this technology in your production run, you should test your applications with and without hyperthreading. If your application runs more than twice as slow with hyperthreading than without, do not use it.

Here is sample PBS script for running OpenMP applications on a NAS Skylake node without hyperthreading:

#PBS -lselect=1:ncpus=40:**ompthreads=40**:model=sky_ele, walltime=1:00:00

module load comp-intel

cd $PBS_O_WORKDIR

./your_executable

Here is sample PBS script with hyperthreading:

#PBS -lselect=1:ncpus=40:**ompthreads=80**:model=sky_ele, walltime=1:00:00

module load comp-intel

cd $PBS_O_WORKDIR

./your_executable

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**