

Table of Contents

Porting to Pleiades	1
<u>Recommended Compiler Options</u>	1
<u>Porting with SGI MPT</u>	3
<u>With MVAPICH</u>	8
<u>With Intel-MPI</u>	10
<u>With OpenMP</u>	12
<u>With SGI's MPI and Intel OpenMP</u>	14
<u>With MVAPICH and Intel OpenMP</u>	16

Porting to Pleiades

Recommended Compiler Options

Intel compiler versions 10.0, 10.1, 11.0, 11.1, and 12.0 are available on Pleiades as modules. Use the `module avail` command to find available versions. Since NAS does not set a default version for users on Pleiades, be sure to use the `module load ...` command to load the version you want to use.

In addition to the few flags mentioned in the article Recommended Intel Compiler Debugging Options, here are a few more to keep in mind:

Turn On Optimization: `-O3`

If you do not specify an optimization level (`-On`, $n=0, 1, 2, 3$), the default is `-O2`. If you want more aggressive optimizations, you can use `-O3`. Note that using `-O3` may not improve performance for some programs.

Generate Optimized Code for a Processor Type: `-xS`, `-xSSE4.1` or `-xSSE4.2`

Intel version 10.x, 11.x and 12.x compilers provide flags for generating optimized codes specialized for various instruction sets used in specific processors or microarchitectures.

Processor Type	Intel V10.x	Intel V11.x and above
Harpertown	-xS	-xSSE4.1
Nehalem-EP	N/A	-xSSE4.2
Westmere	N/A	-xSSE4.2
Sandy Bridge	N/A	-axAVX

Since the instruction set is upward compatible, an application which is compiled with `-xSSE4.1` can run on Harpertown, Nehalem-EP, Westmere, or Sandy Bridge processors. An application that is compiled with `-xSSE4.2` can only run on Nehalem-EP or Westmere processors. An application that is compiled with `-axAVX` can run *only* on Sandy Bridge processors.

If your goal is to get the best performance out of the Nehalem-EP/Westmere processors, it is recommended that you do the following:

- Use either Intel 11.x or 12.x compilers as they are designed for Nehalem-EP/Westmere-EP micro-architecture optimizations
- Use the Nehalem-EP/Westmere-EP processor specific optimization flag `-xSSE4.2`

WARNING: Running an executable built with the `-xSSE4.2` flag on the Harpertown

processors will result in the following error:

```
Fatal Error: This program was not built to run on the processor in
your system. The allowed processors are: Intel(R) processors with
SSE4.2 and POPCNT instructions support.
```

If your goal is to have a portable executable that can run on Harpertown, Nehalem-EP, Westmere, or Sandy Bridge you can choose one of the following approaches:

- Use none of the above flags
- Use `-xSSE4.1` (with 12.x compiler)
- Use `-O3 -ipo -axSSE4.1,axAVX` (with version 12.x compiler)

This allows a single executable that will run on any of the four Pleiades processor types with suitable optimization to be determined at run time.

Turn Inlining On: `-ip` or `-ipo`

Use of `-ip` enables additional interprocedural optimizations for single file compilation. One of these optimizations enables the compiler to perform inline function expansion for calls to functions defined within the current source file.

Use of `-ipo` enables multfile interprocedural (IP) optimizations (between files). When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

Use a Specific Memory Model: `-mmodel=medium` and `-shared-intel`

Should you get a link time error relating to `R_X86_64_PC32`, add in the compiler option of `-mmodel=medium` and the link option of `-shared-intel`. This happens if a common block is > 2gb in size.

Turn Off All Warning Messages: `-w -vec-report0 -opt-report0`

Use of `-w` disables all warnings; `-vec-report0` disables printing of vectorizer diagnostic information; and `-opt-report0` disables printing of optimization reports.

Parallelize Your Code: `-openmp` or `-parallel`

`-openmp` handles OMP directives and `-parallel` looks for loops to parallelize.

For more compiler/linker options, read **man ifort**, **man icc**.

Porting with SGI MPT

Among the many MPI libraries installed on Pleiades, it is recommended that you start with SGI's MPT library.

The available SGI MPT modules are:

- mpi/mpt.1.25
- mpi-sgi/mpt.1.26
- mpi-sgi/mpt.2.04.10789

There is no default MPT version set, but you are recommended to start with the MPT 2.04.10789 version by loading the mpi-sgi/mpt.2.04.10789 module. You should load the same module when you build your application on the front-end node and also inside your PBS script for running on the back-end nodes.

Note:Pleiades uses an InfiniBand (IB) network for interprocess RDMA (remote direct memory access) communications and there are two InfiniBand fabrics, designated as ib0 and ib1. In order to maximize performance, SGI advises that the ib0 fabric be used for all MPI traffic. The ib1 fabric is reserved for storage related traffic. The default configuration for MPI is to use only the ib0 fabric.

Environment Variables

When you load an MPT module, several paths (such as **C_PATH**, **C_INCLUDE_PATH**, **LD_LIBRARY_PATH**, etc.) and MPT or ARRAYD related variables are set or modified. For example, with the mpi-sgi/mpt.2.04.10789 module, the following MPT and ARRAYD related variables are reset to some non-default values:

```
setenv      MPI_BUFS_PER_HOST 256
setenv      MPI_IB_TIMEOUT 20
setenv      MPI_IB_RAILS 2
setenv      MPI_DSM_DISTRIBUTE 0 (for Harpertown processors)
setenv      MPI_DSM_DISTRIBUTE 1 (for Nehalem-EP, Westmere, and Sandy Bridge processors)
setenv      ARRAYD_CONNECTTO 15
setenv      ARRAYD_TIMEOUT 180
```

The meanings of these variables and their default values are:

MPI_BUFS_PER_HOST

Determines the number of shared message buffers (16 KB each) that MPI is to allocate for each host (that is, the Pleiades node used in the run). These buffers are used to send and receive long inter-host messages.

Default: 96 pages (1 page = 16KB)

MPI_IB_TIMEOUT

When an IB card sends a packet it waits some amount of time for an ACK packet to be returned by the receiving IB card. If it does not receive one it sends the packet again. This variable controls that wait period. The time spent is equal to $4 * 2^{\wedge}$

MPI_IB_TIMEOUT microseconds.

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 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.

Default: 1

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 for MPT.1.26; Not Enabled for MPT.1.25

ARRAYD_CONNECTTO

Tuning this variable is useful when you want to run jobs through **arrayd** across a large cluster, and there is network congestion. Setting this variable to a higher value might slow down some array commands when a host is unavailable but it will help to prevent MPI start up problems due to connection time-out.

Default: 5 seconds

ARRAYD_TIMEOUT

Tuning this variable is useful when you want to run jobs through **arrayd** across a large cluster, and there is network congestion. Setting this variable to a higher value might slow down some **array** commands when a host is unavailable but it will help to prevent MPI start up problems due to connection time-out.

Default: 45 seconds

For more MPT related variables, read **man mpi** after loading an MPT module. Some of them may be useful for some applications or for debugging purposes on Pleiades. Here are a few of them for you to consider:

MPI_BUFS_PER_PROC

Determines the number of private message buffers (16 KB each) that MPI is to allocate for each process (that is, the MPI rank). These buffers are used to send long messages and intrahost messages.

Default: 32 pages (1 page = 16KB)
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.

Default: 4

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 environmental 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 environmental 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

Building Applications

Building MPI applications with SGI's MPT library simply requires linking with **-lmpi** and/or **-lmpi++**. See the article [SGI MPT](#) for some examples.

Running Applications

MPI executables built with SGI's MPT are not allowed to run on the Pleiades front-end nodes.

You can run your MPI job on the back-end nodes in an interactive PBS session or through a PBS batch job. After loading an MPT module, use `mpiexec`, not `mpirun`, to start your MPI processes. For example:

```
#PBS -lselect=2:ncpus=8:mpiprocs=4:model=har
....
module load mpi-sgi/mpt.2.04.10789
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 that specified for `mpiprocs`.

Performance Issues

On Nehalem-EP, Westmere, and Sandy Bridge nodes, if your MPI job uses all the processors in each node (8 MPI processes/node for Nehalem-EP, 12 MPI processes/node for Westmere, and 16 MPI processes/node for Sandy Bridge), pinning MPI processes greatly helps the performance of the code. SGI's mpi-sgi/mpt.2.06r6 will pin processes by default by setting the environment variable `MPI_DSM_DISTRIBUTE` to `1` (or `true`) when jobs are run on the Nehalem-EP, Westmere, and Sandy Bridge nodes. On Harpertown nodes, setting `MPI_DSM_DISTRIBUTE` to `1` is not recommended due to a processor labeling issue.

If your MPI job do not use all the processors in each node, it is recommended that you disable `MPI_DSM_DISTRIBUTE` by:

```
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 use `dplace`. Beware that with SGI's MPT, only one shepherd process is created for the entire pool of MPI processes and the proper way of pinning using `dplace` is to skip the shepherd process. In addition, knowledge of the processor labeling in each processor type is essential when you use `dplace`. Below are the recommended ways of pinning an 8 MPI process job with every 4 processes on 4 processor cores of a node, using two nodes:

- Harpertown
`mpiexec -np 8 dplace -s1 -c2,3,6,7 ./your_executable`
- Nehalem-EP
`mpiexec -np 8 dplace -s1 -c2,3,4,5 ./your_executable`
- Westmere
`mpiexec -np 8 dplace -s1 -c4,5,6,7 ./your_executable`

- **Sandy Bridge**

```
mpiexec -np 8 dplace -sl -c6,7,8,9 ./your_executable
```

With MVAPICH

On Pleiades, there are multiple modules of MVAPICH2 built with either `gcc` or Intel compilers.

- `mpi-mvapich2/1.2p1/gcc`
- `mpi-mvapich2/1.2p1/intel`
- `mpi-mvapich2/1.2p1/intel-PIC`
- `mpi-mvapich2/1.4.1/gcc`
- `mpi-mvapich2/1.4.1/intel`

You can get more information of what options were used to build each module as follows:

- Load the desired MVAPICH2 module:

```
%module load mpi-mvapich2/1.4.1/intel
```

- Use the `mpiname` utility provided with the module:

```
%mpiname -a  
MVAPICH2 1.4.1 2010-03-12 ch3:mrail
```

```
Compilation
```

```
CC: icc -fpic -DNDEBUG -O2
```

```
CXX: icpc -DNDEBUG -O2
```

```
F77: ifort -fpic -DNDEBUG -O2
```

```
F90: ifort -DNDEBUG -O2
```

```
Configuration
```

```
--prefix=/nasa/mvapich2/1.4.1/intel.sles11 --enable-f77 --enable-f90 --enable-cxx
```

Because of a formatting issue, the "Configuration" above may appear as several lines. It should only be one line.

Building Applications

Here is an example of how to build an MPI application with MVAPICH2:

```
%module load mpi-mvapich2/1.4.1/intel  
%module load comp-intel/11.1.072  
%mpif90 program.f90
```

Running Applications

To run your job, submit your job through PBS. Within the PBS script, there are two ways to run MPI applications built with MVAPICH2.

```
1. #PBS ..
```

```
...
module load mpi-mvapich2/1.4.1/intel
module load comp-intel/11.1.072

mpiexec -np TOTAL_CPUS your_executable
```

2. #PBS ..

```
...
module load mpi-mvapich2/1.4.1/intel
module load comp-intel/11.1.072

mpirun_rsh -np TOTAL_CPUS -hostfile $PBS_NODEFILE your_executable
```

Performance Issues

To pin processes, the MVAPICH library uses the environment variable **VIADEV_USE_AFFINITY**, which does something similar to SGI's **MPI_DSM_DISTRIBUTE**. By default, **VIADEV_USE_AFFINITY** is set to 1.

If you wish to pin processes explicitly, beware that with MVAPICH, one shepherd process is created for each MPI process. You can use the command to see these processes of your running job:

```
/u/scicon/tools/bin/qsh.pl jobid
```

```
'ps -C executable -L -opsr,pid,ppid,lwp,time,comm'
```

To properly pin MPI processes using **dplace**, one cannot skip the shepherd processes. In addition, knowledge of the processor labeling in each processor type is essential when you use **dplace**. Below are the recommended ways of pinning an 8 MPI process job with every 4 processes on 4 processors of a node, using two nodes:

- Harpertown
`mpiexec -np 8 dplace -c2,3,6,7 ./your_executable`
- Nehalem-EP
`mpiexec -np 8 dplace -c2,3,4,5 ./your_executable`
- Westmere
`mpiexec -np 8 dplace -c4,5,6,7 ./your_executable`
- Sandy Bridge
`mpiexec -np 8 dplace -c6,7,8,9 ./your_executable`

Further information about pinning can be found [here](#).

For more descriptions including the MVAPICH User Guide and other MVAPICH publications, see <http://mvapich.cse.ohio-state.edu>.

With Intel-MPI

Intel's MPI library is another alternative for building and running your MPI application. The available Intel MPI modules are:

- mpi-intel/3.1.038
- mpi-intel/3.1b
- mpi-intel/3.2.011
- mpi-intel/4.0.028
- mpi-intel/4.0.2.003

To use Intel MPI, first create a file `$HOME/.mpd.conf` that has the single line:

```
MPD_SECRETWORD=sometext
```

'*sometext*' should be unique for each user. Change the permission of the file to read/write by you only.

```
%chmod 600 $HOME/.mpd.conf
```

Building Applications

To compile, load an Intel compiler module and an Intel MPI module. Make sure that no other MPI module is loaded (that is, MPT, MVAPICH or MVAPICH2):

```
%module load mpi-intel/4.0.2.003  
%module load comp-intel/11.1.072
```

Use the `mpiifort/mpiicc` scripts which invoke the Intel `ifort/icc` compilers.

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

Running Applications

To run it, in your PBS script make sure the Intel MPI modules are loaded as above, start the MPD daemon, use `mpiexec`, and terminate the daemon at the end. For example:

```
#PBS ..  
..  
module load mpi-intel/4.0.2.003  
module load comp/intel/11.1.072  
  
# Note: The following three lines should really be in one line  
  
mpdboot --file=$PBS_NODEFILE --ncpus=1 --totalnum=`cat $PBS_NODEFILE |  
sort -u | wc -l` --ifhn=`head -1 $PBS_NODEFILE`  
--rsh=ssh --mpd=`which mpd` --ordered  
  
# CPUS_PER_NODE and TOTAL_CPUS below represent numerical numbers
```

```
# for the job at hand  
mpiexec -ppn CPUS_PER_NODE -np TOTAL_CPUS ./your_executable  
  
# terminate the MPD daemon  
mpdallexit
```

With OpenMP

Building Applications

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

```
%module load comp-intel/11.1.072
%ifort -o your_executable -openmp program.f
```

Running Applications

The maximum number of OpenMP threads an application can use on a Pleiades node depends on (i) the number of physical processor cores in the node and (ii) if hyperthreading is available and enabled. Hyperthreading technology is not available for the Harpertown processor type. It is available and enabled at NAS for the Nehalem-EP, Westmere, and Sandy Bridge processor types. With hyperthreading, the OS views each physical core as two logical processors and can assign two threads to it. 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 2 times the walltime compared to running only one thread on the core.

Tip: Before running with hyperthreading for your production runs, it is recommended that you experiment with it to see if it is beneficial for your application.

Maximum Threads		
Processor Type	Maximum Threads without Hyperthreading	Maximum Threads with Hyperthreading
Harpertown	8	N/A
Nehalem-EP	8	16
Westmere-EP	12	24
Sandy Bridge	16	32

Here is sample PBS script for running OpenMP applications on a Pleiades Nehalem-EP node without hyperthreading:

```
#PBS -lselect=1:ncpus=8:ompthreads=8:model=neh,walltime=1:00:00

module load comp-intel/11.1.072

cd $PBS_O_WORKDIR

./your_executable
```

Here is sample PBS script with hyperthreading:

```
#PBS -lselect=1:ncpus=8:ompthreads=16:model=neh,walltime=1:00:00  
module load comp-intel/11.1.072  
cd $PBS_O_WORKDIR  
./your_executable
```

With SGI's MPI and Intel OpenMP

Building Applications

To build an MPI/OpenMP hybrid executable using SGI's MPT and Intel's OpenMP libraries, your code needs to be compiled with the `-openmp` flag and linked with the `-mpi` flag.

```
%module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789
%ifort -o your_executable prog.f -openmp -lmpi
```

Running Applications

Here is a sample PBS script for running MPI/OpenMP application on Pleiades using three nodes and on each node, four MPI processes with two OpenMP threads per MPI process.

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:model=neh
#PBS -lwalltime=1:00:00

module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789
setenv OMP_NUM_THREADS 2

cd $PBS_O_WORKDIR

mpiexec ./your_executable
```

You can specify the number of threads, `ompthreads`, on the PBS resource request line, which will cause the PBS prologue to set the `OMP_NUM_THREADS` environment variable.

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:ompthreads=2:model=neh
#PBS -lwalltime=1:00:00

module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789

cd $PBS_O_WORKDIR

mpiexec ./your_executable
```

Performance Issues

For pure MPI codes built with SGI's MPT library, performance on Nehalem-EP and Westmere nodes improves 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 and setting this variable to 1 causes all OpenMP threads to be pinned on the same core and the performance suffers.

It is recommended that **MPI_DSM_DISTRIBUTE** is set to 0 and **omplace** is to be used 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=8:mpiprocs=4:ompthreads=2:model=neh
#PBS -lwalltime=1:00:00
```

```
module load comp-intel/11.1.072 mpi-sgi/mpt.2.04.10789
```

```
setenv MPI_DSM_DISTRIBUTE 0
setenv KMP_AFFINITY disabled
```

```
cd $PBS_O_WORKDIR
```

```
mpiexec -np 4 omplace ./your_executable
```

With MVAPICH and Intel OpenMP

Building Applications

To build an MPI/OpenMP hybrid executable using MVAPICH and Intel's OpenMP libraries, use `mpif90`, `mpicc`, and `mpicxx` with the `-openmp` flag.

```
%module load comp-intel/11.1.072 mpi-mvapich2/1.4.1/intel
%mpif90 -o your_executable prog.f90 -openmp
```

Running Applications

With MVAPICH, a user's environment variables (such as `VIADEV_USE_AFFINITY` and `OMP_NUM_THREADS`) are not passed in to `mpiexec`, thus they need to be passed in explicitly, such as with `/usr/bin/env`.

Here is an example on how to run a MVAPICH/OpenMP hybrid code with a total of 12 MPI processes and 2 OpenMP threads per MPI process:

```
#PBS -lselect=3:ncpus=8:mpiprocs=4:model=neh

module load comp-intel/11.1.072 mpi-mvapich2/1.4.1/intel

mpiexec /usr/bin/env VIADEV_USE_AFFINITY=0 OMP_NUM_THREADS=2 ./your_executable
```

Performance Issues

Setting the environment variable `VIADEV_USE_AFFINITY` to 0 disables CPU affinity because MVAPICH does its own pinning. Setting it to 1 actually causes multiple OpenMP threads to be placed on a single processor.