How to Use the Condo, CyEnce and CyStorm Clusters

Glenn R. Luecke
July 15, 2016

Online Information and Help

If you experience problems and would like help, send an email to **hpc-help@iastate.edu** with detailed information about the problem. Online documentation for Condo, CyEnce and CyStorm can be found at:

- http://www.hpc.iastate.edu/guides/condo
- http://www.hpc.iastate.edu/guides/cyence
- http://www.hpc.iastate.edu/guides/cystorm

Configuration of the Condo Cluster (installed spring 2015)

1. **168 compute nodes** each with two 8-core, 2.6 GHz Intel Haswell E5-2640 v3 processors (16 cores/node), **128 GB of memory** and 2.5 TB of available local disk ($TMPDIR). The peak theoretical performance of this processor is 144/8 = **18.0 Gflops/core** (with turbo boost). Turbo boost is on for all nodes. Use the "cpuinfo" command to find additional information. The following gives the cache structure for each node in the cluster:
   - L1 32 KB one for each core (no sharing among cores)
   - L2 256 KB one for each core (no sharing among cores)
   - L3 20 MB one for each processor (shared among the 8 cores on the processor)
2. **One "large" memory node** with four 8-core, 2.6 GHz Intel Ivy Bridge E5-4620 v2 processors (32 cores/node), 1 TB of memory and 1.8 TB of available local disk ($TMPDIR). The peak theoretical performance of this processor is 173/8 = **21.6 Gflops/core** (with turbo boost).
3. **One "huge" memory node** with four, 10-core, 2.2 GHz Intel Ivy Bridge E7-4830 v2 processors (40 cores/node), 2 TB of memory and 1.3 TB of available local disk ($TMPDIR). The peak theoretical performance of this processor is 77/10 = 7.7 **Gflops/core** (with turbo boost).
4. All nodes and storage are connected via Intel/Qlogic's QDR **40 Gb/s InfiniBand switch**.
5. 256 TB shared **Lustre** scratch diskspace named /ptmp.
6. 756 TB of RAID-6 long-term **NFS** disk space under quota
7. Operating System: Red Hat Enterprise Linux with Intel's Hyper-Threading turned OFF
8. TORQUE (PBS) for resource and job management
9. **Head Node** (condo.its.iastate.edu), **Data Transfer Node** (condodtn)

Configuration of the CyEnce cluster (installed summer 2013 along with hpc-class)

1. **240 compute nodes** each with two 8-core, 2.0 GHz Intel Sandy Bridge E5-2650 processors (16 cores/node), **128 GB of memory** and 2.5 TB of available local disk ($TMPDIR). The peak theoretical performance of this processor is 179/8 = **22.4 Gflops/core** (with turbo boost). Turbo boost is on for all nodes. Use the "cpuinfo" command to find additional information. The cache structure for all nodes is the same as for the Condo cluster.
2. **One large memory ("fat") node** with four, 8-core, 2.0GHz Intel Sandy Bridge processors (32 cores/node, 1 TB of memory and 2.5 TB of available local disk ($TMPDIR).
3. **16 GPU nodes** configured as a compute node plus 2 NVIDIA K20s.
4. **16 Phi nodes** configured as a compute node plus 2 Intel Phi co-processors.
5. All nodes and storage are connected via Intel/Qlogic's QDR **40 Gb/s InfiniBand switch**.
6. **288TB** shared **Lustre** scratch diskspace named /lustre.
7. **768TB** of RAID-6 long-term **NFS** disk space under quota
8. Operating System: Red Hat Enterprise Linux with Intel's Hyper-Threading turned OFF

9. TORQUE (PBS) for resource and job management
10. **Gateway Node** (discovery.its.iastate.edu), **Head Node** (share), **Data Transfer Node** (stream

Configuration of the CyStorm cluster (installed summer 2009)

1. **Variety of compute nodes. Most compute nodes** have two 4-core AMD Opteron 2345 processors *8 cores/node), **8 GB of memory** and 1.5 GB of available local disk ($TMPDIR). Use the “cpuinfo” command to find additional information.
2. 60 nodes are being used for Hadoop. For Hadoop information see: http://www.hpc.iastate.edu/guides/cystorm/hadoop
3. Nodes connected via a **20 Gb/s InfiniBand switch**.
4. There is no Lustre storage.
5. **23TB** of RAID-6 long-term **NFS** disk space under quota
6. TORQUE (PBS) for resource and job management.
7. **Head Node** (cystorm.its.iastate.edu), **Data Transfer Node** (cystormdtn)
8. The operating system is Red Hat Enterprise Linux.

Obtaining an account & logging onto the Condo cluster

Those who have purchased both nodes and storage can use the Condo cluster. Nodes and storage can be purchased by going to https://hpc-ga.its.iastate.edu/condo/access.html and clicking on Condo Cluster Purchase Form. To obtain an account send an email to hpc-help@iastate.edu . LAS, Mathematics and Statistics have purchased nodes and storage for their researchers. To obtain an account, contact

- For LAS send an email to researchit@iastate.edu
- For mathematics send an email to Cliff Bergman cbergman@iastate.edu
- For Statistics send an email to Michael Brekke mbrekke@iastate.edu

For security **Google Authenticator** is required to logon to the Condo cluster. Instructions on how to install and use Google Authenticator are on the Condo cluster web page: http://www.hpc.iastate.edu/guides/condo/access-and-login

After your account has been set up, you will receive an email with information on how to logon to Condo (condo.its.iastate.edu). Condo is directly accessible only from on-campus machines, i.e. machines whose address ends in “.iastate.edu”. To access Condo from off-campus machines, use the ISU VPN, see http://www.it.iastate.edu/howtos/vpn.

To logon issue: ssh username@condo.its.iastate.edu, where “username@” can be omitted if the username on the machine from which you are issuing the ssh is the same as that on Condo. If you need an X-session, change “ssh” to “ssh -X”. The vi (vim), emacs and xemacs editors are available.

Obtaining an account and logging onto the CyEnce cluster

PIs on the NSF MRI grant and their students and collaborators can use the CyEnce cluster. To obtain an account and SecurID, PIs are to send an email to hpc-help@iastate.edu.

You must first logon to the gateway node (discovery.its.iastate.edu) either using your SecurID or Google Authenticator, and then from the gateway node logon to the head node (share). Discovery is directly accessible only from on-campus machines, i.e. machines whose address ends in “.iastate.edu”. To access discovery from off-campus machines, use the ISU VPN, see http://www.it.iastate.edu/howtos/vpn.

After your account has been set up, you will receive an email with information on how to logon. When you logon the first time, you will create a PIN for discovery and a password for share. The PIN plus the 6 digits from your SecurID is your PASSCODE. The PASSCODE is needed to logon to discovery. You will need to issue the following:

```
ssh username@discovery.its.iastate.edu    [return]
ssh share   [return]
```

("username@" can be omitted if the username on the machine from which you are issuing the ssh is the same as that on discovery.) If you need an X-session, change "ssh" to "ssh -X" when logging onto discover and share. The vi (vim), emacs and xemacs editors are available.

Obtaining an account and logging onto the CyStorm cluster

ISU researchers may apply for an account on CyStorm at ISU's High Performance Computing website, http://www.hpc.iastate.edu/access, by logging with their ISU NetID and password, and submitting the form.

For security **Google Authenticator** is required to logon to CyStorm. Instructions on how to install and use Google Authenticator are on the CyStorm cluster web page: http://www.hpc.iastate.edu/guides/cystorm/access-and-login

After your account has been set up, you will receive an email with information on how to logon to CyStorm (cystorm.its.iastate.edu). CyStorm is directly accessible only from on-campus machines, i.e. machines whose address ends in ".iastate.edu". To access CyStorm from off-campus machines, use the ISU VPN, see http://www.it.iastate.edu/howtos/vpn.

To logon issue: **ssh username@cystorm.its.iastate.edu**, where "username@" can be omitted if the username on the machine from which you are issuing the ssh is the same as that on CyStorm. If you need an X-session, change "ssh" to "ssh -X". The vi (vim), emacs and xemacs editors are available.

Creating the Work and Lustre/ptmp Directories

For both Condo and CyEnce, data is stored on two separate sets of storage servers: NFS storage for permanent data storage (/work) and Lustre storage (/lustre on CyEnce and /ptmp on Condo) for temporary data storage and for parallel I/O. CyStorm has NFS storage and no Lustre storage. Along with the home directories, the /work and /ptmp storage are accessible from all nodes.

Currently each user has 1 GB of home directory space, /home/<username>. In addition, each group has a directory shared by all members of the group. Files should be stored in the work directory and not in the home directory. The group working directories are located in /work on Condo and CyStorm and/or in /work1, /work2 on CyEnce. To find your group issue "groups" on the head node. To find the location of your work directory on CyEnce issue

```
ls -d /work*/<group>
```

Next create your own directory in your group working directory:

```
mkdir /work*/<group>/<username>   where * is either a 1 or 2 on CyEnce
mkdir /work/<group>/<username>     on Condo and CyStorm
```

For CyEnce and Condo, create your own lustre/ptmp directory by issuing:

```
mkdir /lustre/<group>/<username>  on CyEnce
mkdir /ptmp/<group>/<username>   on Condo
```

Lustre is a parallel file system and allows one to take advantage of the parallel MPI routines. Currently, the CyEnce cluster allows for a maximum speedup of 8 for CyEnce and 6 for Condo when using Lustre. Your lustre/ptmp directory is temporary storage where old files will be automatically deleted. Files you want to keep should be copied from your lustre/ptmp directory to your work directory. Please delete your lustre/ptmp files when you have finished with them so that others can utilize this storage.

Normally, all jobs should be submitted from the work directory. However, since CyEnce and Condo have Lustre storage, jobs on these clusters using large files and/or large number of files > 1 MB should be submitted from the lustre/ptmp directory:

```
/lustre/<group>/<username>   on CyEnce
/ptmp/<group>/<username>    on Condo
```

ITS has created the “cdw” command to allow users to easily change to their working directory and the “cds” command to easily change to the Lustre/ptmp directory.

Using the Data Transfer Node (DTN)

On CyEnce the DTN is named “stream”, on Condo it is named “condodtn” and on CyStorm it is named cystormdtn. The DTNs on Condo and CyEnce have 10 Gbit/second connections to the campus network and to the internet allowing for fast transfer of data to on-campus and off-campus machines. The DTN on CyStorm has a 1 Gbit/second connection to the campus network. The DTN should be used to transfer data to and from the cluster. The DTN can also be used to compile large programs when compilation on the head node fails with a segmentation fault.

To use the DTN ssh from the head node. All files and directories on the DTN are identical to those on the head node. There are several ways of transferring data to and from the DTN. If you have files on MyFiles (https://www.it.iastate.edu/services/storage/myfiles), you can copy files to/from MyFiles and DTN. You will need to issue 'kinit' and enter your university password. After entering your password, you will be able to cd to /myfiles/<deptshare>, where <deptshare> is your departmental share name; this is usually your department or college’s shortname, such as engr or las.

You can also use scp and rsync to transfer files between computers. If you want to copy many files and/or directories and/or large files, rsync is recommended over scp. An advantage of using rsync instead of scp is that rsync will only copy files if they have been changed. The following are typical examples using scp and rsync. One can replace “scp” or “scp -r” with “rsync -ae ssh” in the following examples.

Example 1: To copy ~username/mydir/prog.c from hpc-class to ./mydir/prog.c in your current directory on the DTN, issue the following on the DTN:

```
scp  username@hpc-class.its.iastate.edu:mydir/prog.c   ./mydir/prog.c   OR
rsync -ae ssh  username@hpc-class.its.iastate.edu:mydir/prog.c   ./mydir/prog.c
```

Example 2: To copy the entire “mydir” directory in example 1 to the current directory on the DTN, issue the following on the DTN: (The “/ “ is not needed, but without it the “.” is easy to miss.)

```
scp -r  username@hpc-class.its.iastate.edu:mydir   ./  OR
rsync -ae ssh  username@hpc-class.its.iastate.edu:mydir   ./
```

Example 3: To copy all the files in “mydir” that end in .c from hpc-class to DTN, issue the following on the DTN:

```
scp  username@hpc-class.its.iastate.edu:mydir/*.c   ./newdir   OR
rsync -ae ssh  username@hpc-class.its.iastate.edu:mydir/*.c   ./newdir
```

Example 4: To copy prog1.f90 in your current directory on the DTN to prog2.f90 in your home directory on hpc-class issue the following on the DTN:

```
scp  prog1.f90  username@hpc-class.its.iastate.edu: prog2.f90   OR
rsync -ae ssh  prog1.f90  username@hpc-class.its.iastate.edu: prog2.f90
```

If you’re using Windows computer, you probably don’t have scp available. In this case you can download free SFTP/SCP client WinSCP.

Globus Online is the recommended when transferring large amounts of data. Globus Online is available only on CyEnce and Condo. For information on how to use Globus Online, see http://www.hpc.iastate.edu/guides/gridftp.

Software

The web site for each machine contains a listing of the software available for each machine. Groups are responsible for installing and paying for their own group specific software. Some groups are using the modules facility to manage installed software with several versions, e.g. R and OpenFoam. Alternative compilers and MPI implementations can also be loaded via modules. To see which modules are available, issue

```
module avail
```

To load a specific module, issue "module load NAME" command, e.g.

```
module load intel          - loads default Intel software
module load intel/16.0   - loads version 16.0 of the Intel software
```

The "module list" command shows the currently loaded modules. The "module purge" command unloads all loaded modules. To ensure routines from the desired module are used, we recommend setting the module environment within the job script by first purging all modules and loading only those modules that are needed for the job

Compilers

The Intel Fortran (ifort) and C/C++ (icc) compilers are the recommended compilers. To use Intel compilers load the intel module. The following command will load the default intel module:

```
module load intel
```

One can also load specific version, such as intel/16.0:

```
module load intel/16.0
```

The GNU (gfortran, gcc) compilers are also available. In addition on CyEnce the PGI (pgfortran, pgCC) compilers are also available. Compiler options can be found by issuing.

```
man <compiler_name>
```

Free format Fortran files must be named something.f90, C files must be named something.c and C++ files must be named something.cc. When one successfully compiles a program, the executable will be placed in the file named “a.out” in your current directory. Use the ***-o*** option if you want the executable to go to a different file.

Compiles are usually done on the head node. To prevent users from running computations on the head node, memory limits on the head node are set low. This may result in segmentation fault when compiling large programs. In this case you can compile your program on the DTN.

Debugging Options for Intel Compilers

The ***-g*** option produces symbolic debug information that allows one to use a debugger. The ***-debug*** option turns on many of the debugging capabilities of the compiler. The ***-qopenmp*** option allows the recognition of OpenMP directives/pragmas and should not be used if it is not needed. The debuggers available are Allinea’s ddt, Intel’s Inspector XE (inspxe-cl and inspxe-gui), GNU’s gdb and PGI’s pghpf. The recommended debugger is ddt. The following shows how to perform compiling for debugging with automatic linking of Intel’s MPI libraries and producing an executable named “prog”. Note, if you do not need the MPI library, change “mpiifort” to “ifort”, “mpiicc” to “icc” and “mpiicpc” to “icpc”. Intel’s MPI Checker provides special checking for MPI programs. To use **MPI Checker** add the ***-check-mpi*** to the mpirun command.

```
mpiifort  -g  -debug  -check all  -traceback   -o prog prog.f90
mpiicc    -g  -debug  -check-pointers=rw  -check-uninit  -traceback  -o prog  prog.c
mpiicpc   -g  -debug  -check-pointers=rw  -check-uninit  -traceback  -o prog  prog.cc
```

Remarks:

- The ***-check all*** option turns on all run-time error checking for Fortran. (There appears to be no equivalent option for Intel’s C/C++ compiler.)
- The ***-check-pointers=rw*** option enables checking of all indirect accesses through pointers and all array accesses for Intel’s C/C++ compiler but not for Fortran.
- The ***-check-uninit*** option enables uninitialized variable checking for Intel’s C/C++ compiler. This functionality is included in Intel’s Fortran ***-check all*** option.
- The ***-traceback*** option causes the compiler to generate extra information in the object file to provide source code traceback information when a severe error occurs at run-time.

High Performance Options for Intel Compilers

The Intel compiler will generate code for the host processor being used with the **-xHost** or equivalently **-march=native** option. Intel recommends the ***-O2*** compiler option as the optimization option that will usually give the best performance and this option is on by default. Intel also provides the **-O3, -ipo, *-fast*** and **-Ofast** options that turn on many optimizations, but they sometimes slow down program execution. The ***-ipo*** option turns on interprocedural analysis between procedures. For best performance, experiment with the ***-02, -03***, ***-ipo***, **-Ofast** and ***-fast*** compiler options and find out which gives the best performance for your application. Choosing good compiler options can significantly reduce execution time. If the application uses both MPI and OpenMP, then on must add the ***-qopenmp*** option to the following:

```
mpiifort  -xHost   -o prog  prog.f90
mpiicc    -xHost   -o prog  prog.c
mpiicpc  -xHost   -o prog  prog.cc
```

On the Condo cluster the fat and huge nodes have older processors than the head node's processors. Thus compiles with the –xHost compiler option should be done on the fat/huge node when compiling programs that will run on those nodes. For the fat/huge nodes compile commands should be added to the job script so that the compiles will be performed on the fat/huge nodes.

How to run MPI and OpenMP jobs

Condo, CyEnce, CyStorm and the student cluster use the TORQUE (PBS) job scheduler using the routing queue feature. Jobs are submitted to "routing_queue". The routing_queue submits jobs to a destination queue based on job attributes and queue constraints. This means that one need not specify a specific job queue. Jobs are submitted using the qsub command. For example, if the script file is named "prog.script", then the job is submitted for execution by issuing

```
qsub            prog.script  -  for the regular compute nodes
qsub -q fat     prog.script  -  for the 1 TB node
qsub -q huge prog.script  -  for the 2 TB node
```

The program will be compiled on the first node assigned to this job.

The compute nodes for Condo and CyEnce have 128 GB of memory and 2 processors with each processor having 8 cores. Thus each node has 16 cores and 128 GB of memory (128/16 = 8 GB per core). On CyStorm regular compute nodes have 8 GB of memory and 8 cores (1GB per core). For help creating scripts on CyEnce see: http://hpcgroup.public.iastate.edu/HPC/CyEnce/CyEnce_script_writer.html. For scripts on Condo, see: http://hpcgroup.public.iastate.edu/HPC/Condo/Condo_script_writer.html. For CyStorm see: http://hpcgroup.public.iastate.edu/HPC/CyStorm/CyStorm_script_writer.html
What follows below provides background information to help you run your MPI and MPI/OpenMP programs using scripts on Condo and CyEnce. To run on CyStorm, these scripts will need to be modified.

The -***perhost n*** option on the mpirun command means that **n** MPI processes are assigned per node, cycling through nodes in a round-robin fashion. If one wants 16 consecutive MPI processes assigned to each node, one should use ***-perhost 16***. If one wants 1 MPI process for each of the two processors/sockets on a node, then use ***-perhost 2***. With the current version of Intel's MPI this option takes effect only when combined with the '-f ${PBS_NODEFILE}' option for mpirun.

Pinning MPI processes to cores: During program execution, the operating system may move MPI processes to different cores on the same node. Remember that each processor/socket on a node has its own memory and that memory accesses from one socket to the memory of the other socket takes longer that memory accesses to the socket's memory. Memory is allocated using "first touch" and memory allocations remain in the same physical memory throughout the execution of the MPI job. By turning on MPI process pinning, MPI processes cannot be moved to other cores during program execution. MPI process pinning is on by default. If Intel changes this default, memory pinning can be enabled by using the ***-genv*** option with the mpirun command. The following are sometimes useful when using the ***-genv*** option with the mpirun command.

- **-genv I_MPI_PIN on** Turns on processor pinning. It is on by default.

- **-genv I_MPI_PIN_PROCESSOR_LIST 0-15** Pins MPI process 0 to core 0, process 1 to core 1, . . . , process 15 to core 15. This is the default assignment.

- **-genv I_MPI_PIN_PROCESSOR_LIST 0,8** Pins MPI processes to cores 0 and 8. This is useful when running programs using both MPI and OpenMP.

- **-genv I_MPI_PIN_DOMAIN socket** Pins MPI processes to processors/sockets and is useful for MPI/OpenMP programs since this option insures that the execution of OpenMP threads cannot be moved to different sockets.

- **-genv I_MPI_DEBUG 4** - prints process pinning information.

Adding **"command time -v"** before **mpirun** when in the bash shell provides an easy way to time your program and to estimate its parallel efficiency by looking at the "Elapsed (wall clock) time". Replacing **mpirun** with **command time –v mpirun** in the following example scripts will cause the timing information to be placed in the prog.errors file.

If one runs out of stack memory, then a segmentation fault error will be produced and the program will not run. It is recommended to always remove this **stacksize** limit by issuing:

```
ulimit -s unlimited
```

Script 1: MPI with 16 MPI processes/node
Suppose one wants to run an MPI program, prog.f90, with 64 MPI processes using all 64 cores on 4 nodes with a 3 hour and 20 minutes time limit with the output written to ./prog.out and error messages written to ./prog.errors. Setting ppn=16 in the following means that all 16 cores on each node are reserved for your job. ("ppn" stands for "processors per node".) Setting nodes=4 means 4 nodes are reserved for your job. Since pinning MPI processes to cores is the default, the **-genv I_MPI_PIN on** option need not be specified. (I suggest you save this script as prog.script64 so you know that the script is for prog.f90 with 64 MPI processes.)

```
#!/bin/bash
#PBS  -o ./prog.out
#PBS  -e ./prog.errors
#PBS -lnodes=4:ppn=16:compute,walltime=3:20:00
cd $PBS_O_WORKDIR
ulimit -s unlimited
module purge
module load intel
mpiifort  -xHost  -o prog  prog.f90
mpirun   -perhost 16  -np 64  ./prog
```

Script 2: MPI with 8 MPI processes/node (double the memory/MPI process)
If the above does not provide enough memory per MPI process, then the ***perhost 8*** option on the mpirun command can be used so that there will be 8 (instead of 16) MPI processes per node. To keep the same number of MPI processes, the script specifies 8 instead of 4 nodes. One needs to keep "ppn=16" so entire nodes are reserved for the job. In the following script MPI processes are pinned to the first 4 cores on each of the two processors/sockets on each node. Note: the perhost option is currently not working and below shows the work-around needed to make it work. Currently, all scripts using the perhost option need to use this work-around.

```
#!/bin/bash
#PBS  -o ./prog.out
#PBS  -e ./prog.errors
#PBS -lnodes=8:ppn=16:compute,walltime=3:20:00
cd $PBS_O_WORKDIR
ulimit -s unlimited
module purge
module load intel
mpiifort  -xHost  -o prog  prog.f90
mpirun  -perhost 8 -f ${PBS_NODEFILE} -genv I_MPI_PIN_PROCESSOR_LIST 0-3,8-11 -np
64  ./prog
```

Script 3: MPI with 2 MPI processes/node+OpenMP
OpenMP is used for parallelization on shared memory computers and MPI for parallelization on distributed (and shared) memory computers. Since memory is shared on a node, one can parallelize with OpenMP within nodes and MPI between nodes. One could specify 1 MPI process per node and use 16 OpenMP threads to parallelize within a node.   However, since there are two processors/sockets per node and since each processor has memory physically close to it, it is generally recommended to use 2 MPI processes per node and use 8 OpenMP threads for the 8 cores on each processor/socket. One can set the number of threads by either setting the OMP_NUM_THREADS environment variable or by calling omp_set_num_threads in the program. OpenMP parallelization requires the insertion of directives/pragmas into a program and then compiled using the ***-qopenmp*** option. To configure script 3 to use the same number of cores as scripts 1 means that we want to use all 64 cores in 4 nodes and 2 MPI processes per node for a total of 8 MPI processes.

To keep OpenMP threads active, set OMP_WAIT_POLICY=ACTIVE. To keep threads executing on the same processor/socket set OMP_PLACES=sockets and OMP_PROC_BIND= close. There is a default OpenMP stacksize and jobs that require more memory will produce a segmentation fault error message and will not run. The OpenMP stacksize can be increased using the OMP_STACKSIZE environment variable. For the student cluster, I recommend setting the OpenMP stacksize=1G and =2G for the CyEnce and Condo clusters. The following is the recommended script when using both MPI and OpenMP:

```
#!/bin/bash
#PBS  -o ./prog.out
#PBS  -e ./prog.errors
#PBS -lnodes=4:ppn=16:compute,walltime=3:20:00
cd $PBS_O_WORKDIR
ulimit -s unlimited
module purge
module load intel
export OMP_WAIT_POLICY=ACTIVE
export OMP_PROC_BIND=close
export OMP_PLACES=sockets
export OMP_STACKSIZE=2G
mpiifort  -xHost -qopenmp  -o prog  prog.f90
mpirun  -perhost 2  -genv I_MPI_PIN_DOMAIN socket  -np 8  ./prog
```

It may be that better performance will be achieved by pinning threads to cores and not allowing threads to move between cores. To do this, replace

```
export OMP_PROC_BIND=close
export OMP_PLACES=sockets
```

with

```
export OMP_PROC_BIND=true
```

Calling MPI routines outside of OpenMP parallel regions does not require the replacement of mpi_init with mpi_init_thread. However, when placing MPI calls inside parallel regions, then one must replace the call to mpi_init(ierror) with mpi_init_thread(*required*, *provided*, ierror) where *required* is the desired level of thread support. In the *provided* the level of provided thread support will be returned. The following are the values *required* and *provided* can have:

- MPI_THREAD_SINGLE: only one thread will execute.
- MPI_THREAD_FUNNELED: The process may be multi-threaded, but only the main thread will make MPI calls (all MPI calls are funneled to the main thread).
- MPI_THREAD_SERIALIZED: The process may be multi-threaded, and multiple threads may make MPI calls, but only one at a time: MPI calls are not made concurrently from two distinct threads (all MPI calls are serialized).
- MPI_THREAD_MULTIPLE: Multiple threads may call MPI, with no restrictions.

For best performance, use the least general option possible. Notice that one can overlap communication and computation by having one thread execute MPI routines and the other threads do computations. There is also a possibility of reducing memory requirements over a pure MPI implementation since processor cores share a common memory.

Script 4: How to run OpenMP on Large Memory Nodes

On CyEnce, there is one large memory node, called “fat” that has 1 TB of memory and 4 processor/sockets (32 total cores). Condo has two large memory nodes: one is called “large” and has 1 TB of memory and 32 cores and the other is called “huge” and has 2 TB of memory and 40 cores. The recommended way to use the large memory nodes is to use 32 or 40 OpenMP threads by setting omp_set_num_threads(32 or 40) in the program. In the example script below, one must change “ppn=32:fat” to “ppn=40:huge” for the huge memory node and replace “fat” with “huge”. One must use ***qsub -q fat*** to submit jobs for the 1 TB memory node and ***qsub -q huge*** for the 2 TB node. Notice the compiler command inside the script so that the program will be compiled on the large memory node where it will execute. This is important since these nodes use different processors from the regular compute nodes.

```
#!/bin/bash
#PBS  -o ./prog.out
#PBS  -e ./prog.errors
#PBS -lnodes=1:ppn=32:fat,walltime=3:20:00
cd $PBS_O_WORKDIR
ulimit –s unlimited
module purge
module load intel
export   OMP_WAIT_POLICY=ACTIVE
export   OMP_PROC_BIND=close
export   OMP_PLACES=sockets
export   OMP_STACKSIZE=4G
ifort      -xHost  -qopenmp  -o prog  prog.f90
./prog
```

Job Queue Commands

1. **qstat -a** - status of all jobs executing and queued
2. **qstat –u <userid>** - status of only your jobs – useful when there are many jobs
3. **qstat -q** - status of all queues and the current queue structure
4. **qtop #** - allows one to look at current memory and cpu usage for each node being used by job # and look at the %MEM and %CPU columns for each node being used for your job
5. **qstat -f #** - gives information about the job for job number #
6. **qdel #** - deletes the job with number #. One can only delete one’s jobs

How many nodes should one use?

The answer depends on the problem size and how well your application has been parallelized. Often applications will require a minimum number of nodes due to large memory requirements. Once this minimum number of nodes has been established, one must decide how many nodes to use for running the application. For example, let's take an MPI parallelization of the Jacobi iteration with N = 4*1024 and N = 64*1024 using differing numbers of nodes. Both of these problem sizes can be run on a single node, so the question is how many nodes should one use. The following numbers were obtained running on ISU CyEnce cluster in the fall of 2013.

For N = 4*1024, it is best to use 1 node for this program if one wants to make best use of the allocation and use 8 nodes to minimize the execution time, see Table 1. For N = 64*1024, using 64 nodes gives the shortest time and the cost for the allocation is not much different from the best value using only 1 node, see Table 2.

Number of Nodes	Seconds for N = 4*1024	Node-Seconds for N = 4*1024
1	3.3	3.3
2	2.3	4.6
4	1.3	5.2
8	0.8	6.4
16	1.8	28.8

Table 1: Jacobi iteration with N = 4*1024 using different numbers of nodes.

Number of Nodes	Seconds for N = 64*1024	Node-Seconds for N = 64*1024
1	875.6	875.6
2	442.4	884.8
4	224.7	898.8
8	113.8	910.4
16	59.4	950.4
32	32.6	1,043.2
64	17.2	1,100.8

Table 2: Jacobi iteration with N = 64*1024 using different numbers of nodes.

The Student Cluster (hpc-class)

The purpose of the student cluster is to support HPC instruction at ISU. Compute nodes are identical to the compute nodes on CyEnce but with half the memory. There are 4 GPU nodes each with a single K20. There are 4 Phi nodes each with a single Phi co-processor. For information see: http://www.hpc.iastate.edu/education/

Using Intel's Math Kernel Library

Intel provides highly optimized scientific library routines for Fortran and some C versions in their Math Kernel Library (MKL). You should call these routines from your program whenever possible to increase the performance of your application. For documentation see http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation . Intel provides a tool to help one select the correct link –L and –l options to add to a Fortran or C/C++ compile/link command and to correctly set environmental variables. This tool is available at: http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor . To link the MKL serial optimized libraries compile your program with

```
ifort -mkl=sequential.
```

How to use the Accelerator Nodes

Both the student cluster and CyEnce clusters have accelerator nodes but the Condo cluster does not. Information on how to use the accelerator nodes can be found on the web sites of these clusters.

ISU HPC Courses

Parallel computing techniques taught in the courses listed below will allow students to take discipline specific courses that teach parallel algorithms relevant to the discipline.

Math/ComS/CprE 424: Introduction to Scientific Computing (3 credits)

- For students with no unix experience and no experience in writing scientific programs in C or Fortran
- Course topics: Unix, high performance serial and shared memory parallel programming with OpenMP
- Offered fall semesters

Math/ComS/CprE 525: High Performance Computing (3 credits)

- For students with unix experience and experience in writing scientific programs in C or Fortran
- Course topics: HPC concepts and terminology, HPC machine architecture, how to use debugging and performance tools, advanced parallel programming with MPI, OpenMP and possibly OpenACC and/or CUDA.
- Offered spring semesters