How to use the Student Cluster with Slurm

Professor Glenn Luecke

January 25 2017

The Configuration of the Student Cluster (installed July, 2013 by Atipa)

1. Online information: http://hpcgroup.public.iastate.edu/HPC/hpc-class/homepage.html
   problems via email to hpc-help@iastate.edu.
2. Report problems via email to hpc-help@iastate.edu.
3. 40 **compute nodes** each with 2 Intel Sandy Bridge processors each with 8 cores (16 cores/node), 64 GB of memory and 2.5 TB of available local disk ($TMPDIR). The peak theoretical performance of this processor is **22.4 Gflops/core** with turbo boost (16 Glops without turbo boost). The student cluster and CyEnce clusters are run with turbo boost. Two of these nodes are reserved for interactive use. Use the "cpuinfo" command to find additional information. The cache structure is: (a cache line = 64 bytes)
   - 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)
4. 4 **GPU nodes** configured as a compute node plus 1 NVIDIA K20 GPU.
5. 4 **Phi nodes** with each configured as a compute node plus 1 Intel Phi co-processor.
6. **Communication Network**: All nodes and storage are connected via Intel/Qlogic's QDR InfiniBand (40 Mb/s) switch.
7. **Operating System**: Red Hat Enterprise Linux with Intel's Hyper-Threading turned OFF
8. **Job-Scheduler**: Slurm for resource and job management
9. **Head Node** (hpc-class.its.iastate.edu)

Obtaining an Account on the Student Cluster

The purpose of the student cluster is to support HPC for instruction. Students are welcome to use the student cluster, but not for funded research projects. Students cannot apply for an account directly but can obtain an account via an instructor/faculty member. Instructors can obtain accounts for themselves and the students in their classes and students signed up for research credits by filling out the online form at:

http://hpcgroup.public.iastate.edu/HPC/hpc-class/instructor_application_interactive.html

Logging Onto the Student Cluster

To logon to hpc-class, issue ssh to username@hpc-class.its.iastate.edu The "username@" can be omitted if the username on the machine from which you are issuing the ssh is the same as that on hpc-class. If you need an X-session, change "ssh" to "ssh -X" when logging on. If your PC/laptop is a Windows machine, you can download free ssh client PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.htm . The vi (vim), nano, emacs and xemacs editors are available. The password for your account on hpc-class is the same as that for CyMail or http://asw.iastate.edu The first time you issue ssh to logon to hpc-class, the message "host key not found, generate hostkey?(yes/no)" will appear. Answer yes.

Copying Data to/from the Student Cluster

If your PC/laptop is a Windows machine, you can install the SSH Secure Shell File Transfer Client to move files to and from hpc-class. Use "scp" or "rsync -ae ssh" to copy files to and from hpc-class. “rsync” should be used for large files and has the advantage that it will only copy updated files. In the following examples one can replace “scp” or “scp -r” by “rsync -ae ssh”.

Example 1: Suppose you are logged on a machine that is not hpc-class and would like to copy the file mydir/prog.f90 from hpc-class to ./mydir/prog.f90 in your current working directory. This can be done by issuing the following in your current working directory on the non-hpc-class machine:

```
scp username@hpc-class.its.iastate.edu:mydir/prog.f90  ./mydir/prog.f90
```

Example 2: To copy the entire “mydir” directory in example 1, issue the following from your current working directory:

```
scp -r username@hpc-class.its.iastate.edu:mydir  ./mydir
```

Backing up files: Files are not backed up, so users are encouraged to back up their files to the automatically backed up Myfiles. To back up your files to Myfiles first make sure that your Kerberos ticket is valid by **issuing: kinit** <return> when you are on he head node (i.e. you are not in an interactive session). Next issue the following from your home directory:

```
rsync -pvr  /home/$USER/  /myfiles/Users/$USER/hpc-class
```

The backed up files can be viewed on hpc-class by going to /myfiles/Users/$USER/hpc-class.

Software:

Since various users on the cluster need a different selection of software packages, often with different versions and configurations, the Environment Modules are used to manage multiple versions and configurations of software packages. The following are commonly used module commands.

```
module avail  -  displays all available modules
module load <module_name>  - loads specified module
module unload <module_name>  -  unloads specified module
module list  -  lists all loaded modules
module purge  -  unloads all loaded modules
```

We recommend using the Intel module by adding the following two lines to all scripts and to your .bashrc file: (The sample scripts below illustrate this.)

```
module purge
module load intel
module load allinea
```

where the “module load allinea” is needed to use Allinea’s DDT, MAP and Performance Reports tools.

Compilers:

The Intel Fortran (ifort) and C/C++ (icc) compilers are the recommended compilers. The GNU (gfortran, gcc) and 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, e.g ifort ***-o prog*** program.f90 will produce the executable “prog” when the compile is successful.

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 functions and directives/pragmas. The debuggers available are Allinea’s ddt, Intel’s idb, GNU’s gdb and PGI’s pghpf. The recommended debugger is ddt. The following shows how to compile 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 “miicpe” to “icpc”.

```
    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 file traceback information when a severe error occurs at run-time.

Checking MPI programs using Intel’s Message Checker

Intel’s Message Checker tool checks for MPI errors at run-time. To use Message Checker, compile with the ***-g*** option and then add the ***-check-mpi*** option on the ***mpirun*** command.

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,** ***-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
```

The Slurm Workload Manager

On HPC clusters computations should be performed on the compute nodes and not on the head node. Special programs called resource managers, workload managers or job schedulers are used to allocate compute nodes to users' jobs. The Slurm Workload Manager is used for this purpose. Jobs can be run in interactive or batch modes.

Interactive Job Execution

When executing/debugging short-running MPI and OpenMP jobs, interactive execution instead of batch execution may speed up program development. Two compute nodes on the student cluster have been reserved for interactive usage. To start an interactive session, issue

```
salloc -N <number of compute nodes> -n <number_of_cores> -t <hh:mm:ss>
```

where the -N option is the number of compute nodes (1 or 2 on the student cluster). If the -N option is omitted, then it defaults to 1. Since each compute node on the student cluster has 16 cores, 32 is the maximum <number_of_cores> when 2 compute nodes are used and 16 is the maximum number_of_cores when 1 compute node is used. If the -t option is omitted, then it defaults to 1 hour. Issuing "salloc" without any arguments will allocate 1 compute node and 4 cores for a period of 1 hour. If one wanted to use 2 compute nodes and 16 cores for 3 hours, 15 minutes and 0 seconds, one would issue the following:

```
salloc -N 2 -n 16 -t 3:15:00
```

When running interactively, our job may be sharing processors with jobs from other users so your job may run slowly due to contention with other interactive jobs, so timings may not be accurate.

Batch Job Execution

When running longer jobs, batch and not interactive execution should be used. To run your job using batch execution a script should be created and submitted to the job queue by issuing

```
sbatch <job_script>
```

The job script will contain Slurm settings, such as time and the number of nodes/cores requested, and the commands that should be executed during the batch session. The Slurm Script Writer at http://www.hpc.iastate.edu/guides/classroom-hpc-cluster/slurm-script-writer-for-hpc-class can be used to generate slurm job scripts. Existing PBS job scripts can be translated to Slurm using pbs2sbatch command. Example scripts are presented later in this document.

How to run MPI and OpenMP jobs

As described above, for interactive execution first issue

```
salloc -N <number of nodes> -n <number_of_cores>  –t <hh:mm:ss>
```

Assume your program has been successively compiled and an executable prog file has been formed. Then issuing:

```
mpirun –np <number_of_cores>  ./prog
```

will execute your job interactively using <number_of_cores> MPI processes with output directed to the screen of your workstation. To terminate interactive sessions issue: exit

Use the sbatch Slurm command for batch execution of jobs via scripts. For example, if the script file is named "prog.script", then the job is submitted for execution by issuing

```
sbatch prog.script
```

There is an automatic scriptwriter at http://www.hpc.iastate.edu/guides/classroom-hpc-cluster/slurm-script-writer-for-hpc-class that one can use to write scripts. What follows provides background information to help you run your MPI and MPI/OpenMP programs using scripts.

Recall that each compute node has 64 GB of memory and 2 processors/sockets with each processor having 8 cores. Thus, MPI programs that do not use OpenMP are normally run with 16 MPI processors per node. If there is not enough memory on a node, then one can have twice the available memory by running 8 MPI processes per node and using twice as many nodes.

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*** (the default value is 16). If one wants 1 MPI process for each of the two processors/sockets on a node, then use ***-perhost 2***. In order for this option to work, first you need to execute the following command:

```
export setenv I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=disable
```

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, then 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 sockets and is useful for MPI/OpenMP programs since option 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"** to mpirun when in the bash, dash and ksh shells provides an easy way to time your program and to estimate its parallel efficiency by looking at the "Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.73". Replacing "mpirun" with "command time –v mpirun" in the following example scripts will cause the timing information to be placed in 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. One can remove this stacksize limit by issuing:

```
ulimit -s unlimited
```

Script 1: MPI with 4 nodes each with16 MPI processes

Suppose one wants to run a 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 N to 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
# SBATCH  -o ./prog.out
# SBATCH  -e ./prog.errors
# SBATCH -N 4
# SBATCH -n 64
# SBATCH -t 3:20:00
module purge
module load intel
module load allinea
ulimit -s unlimited
mpiifort  -xHost  -o prog  prog.f90
export setenv I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=disable
mpirun   -perhost 16  -np 64  ./prog
```

Script 2: MPI with 8 nodes and 8 MPI processes/node

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 and 128 cores instead of 64 cores. In the following script MPI processes are pinned to the first 4 cores on each of the two processors/sockets on each node.

```
#!/bin/bash
# SBATCH  -o ./prog.out
# SBATCH  -e ./prog.errors
# SBATCH -N 8
# SBATCH -n 128
# SBATCH -t 3:20:00
module purge
module load intel
module load allinea
ulimit -s unlimited
mpiifort  -xHost  -o prog  prog.f90
export setenv I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=disable
mpirun  -perhost 8 -genv I_MPI_PIN_PROCESSOR_LIST 0-3,8-11 -np 64  ./prog
```

Script 3: MPI with 4 nodes and 2 MPI processes/node+OpenMP

OpenMP is used for parallelization of shared memory computers and MPI for parallelization of 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 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 Condo and CyEnce clusters. The following is the recommended script when using both MPI and OpenMP:

```
#!/bin/bash
# SBATCH  -o ./prog.out
# SBATCH  -e ./prog.errors
# SBATCH –N 4
# SBATCH –n 64
# SBATCH –t 3:20:00
module purge
module load intel
module load allinea
ulimit -s unlimited
export setenv I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=disable
export OMP_WAIT_POLICY=ACTIVE
export OMP_PROC_BIND=close
export OMP_PLACES=sockets
export OMP_STACKSIZE=1G
mpiifort  -xHost -qopenmp  -o prog  prog.f90
mpirun  -perhost 2  -genv I_MPI_PIN_DOMAIN socket  -np 8  ./prog
```

Note: If one wants to pin threads to cores and not allow threads to move between cores, then 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 and *provided* is the thread support provided. The following are the values *required* 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 are free to 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: OpenMP with 1 Node and 16 Threads

A recommended script for running an OpenMP program using 16 threads and without MPI would look like the following. (Notice that one is limited to execution on a single node.)

```
#!/bin/bash
# SBATCH  -o ./prog.out
# SBATCH  -e ./prog.errors
# SBATCH -N 1
# SBATCH -n 16
# SBATCH -t 3:20:00
module purge
module load intel
module load allinea
ulimit -s unlimited
export  OMP_WAIT_POLICY=ACTIVE
export  OMP_PROC_BIND=close
export  OMP_PLACES=sockets
export  OMP_STACKSIZE=1G
ifort  -xHost -qopenmp  -o prog  prog.f90
./prog
```

Job Queue Commands

1. **squeue** - status of all jobs executing and queued
2. **squeue –u <your_userid>** status of only your jobs – useful when there are many jobs
3. **sinfo** - status of all queues and the current queue structure
4. **scancel #** - 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, lets 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.

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 . One can link in the MKL serial optimized libraries compile your program with ***ifort -mkl=sequential***.

How to use the Accelerator Nodes

Information on how to use the accelerator nodes can be found on the student cluster's web site.