The Information Science & Technology Center

ISTeC.colostate.edu

ISTeC Cray High Performance Computing System

User's Guide

Version 5.1

Updated 06/17/15

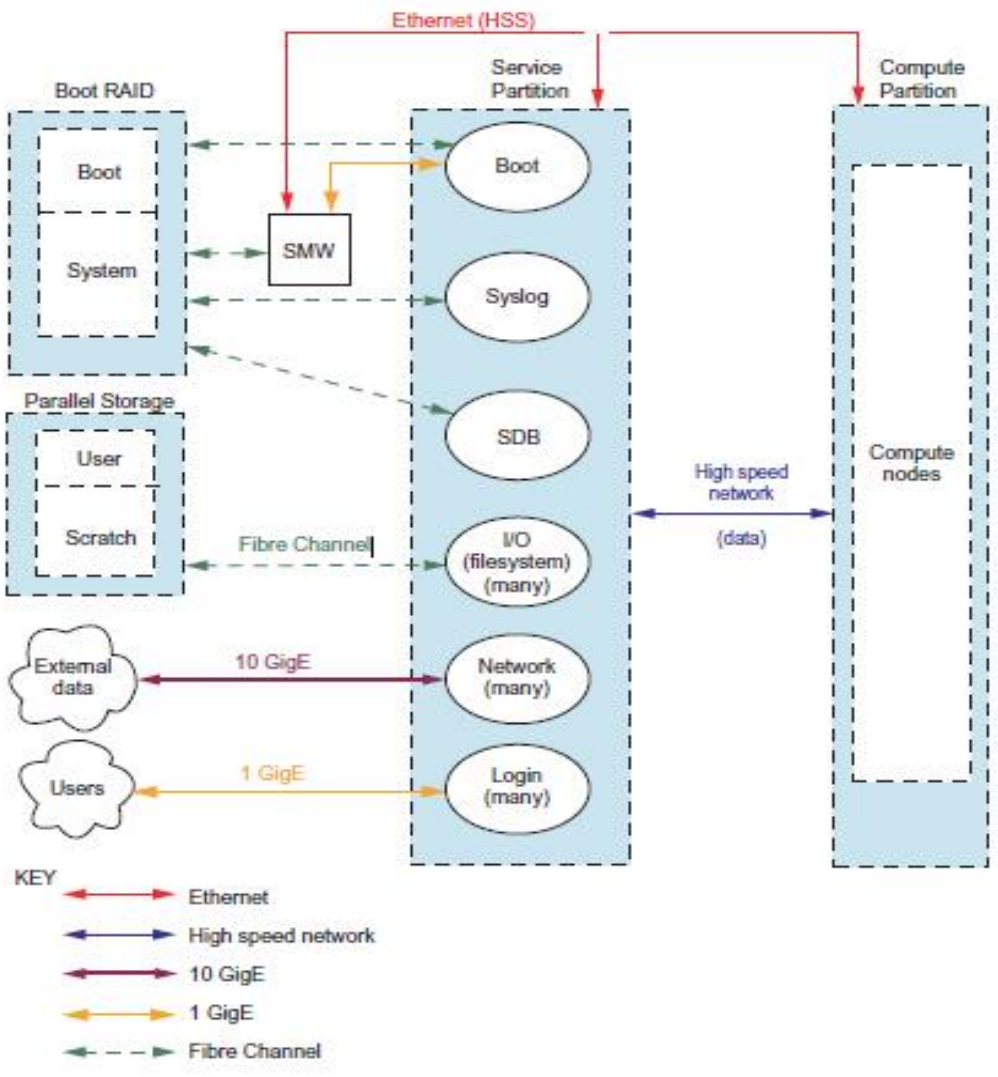
Contents

Purpose and Audience for this Document

This document is intended for users of ISTeC's Cray High Performance Computer at Colorado State University. The emphasis is for new users, and some of the information presented is particular to CSU's environment and system. As such, this document is designed to supplement, and not displace, Cray's excellent and extensive documentation available at http://docs.cray.com/ (Platforms, Cray XE).

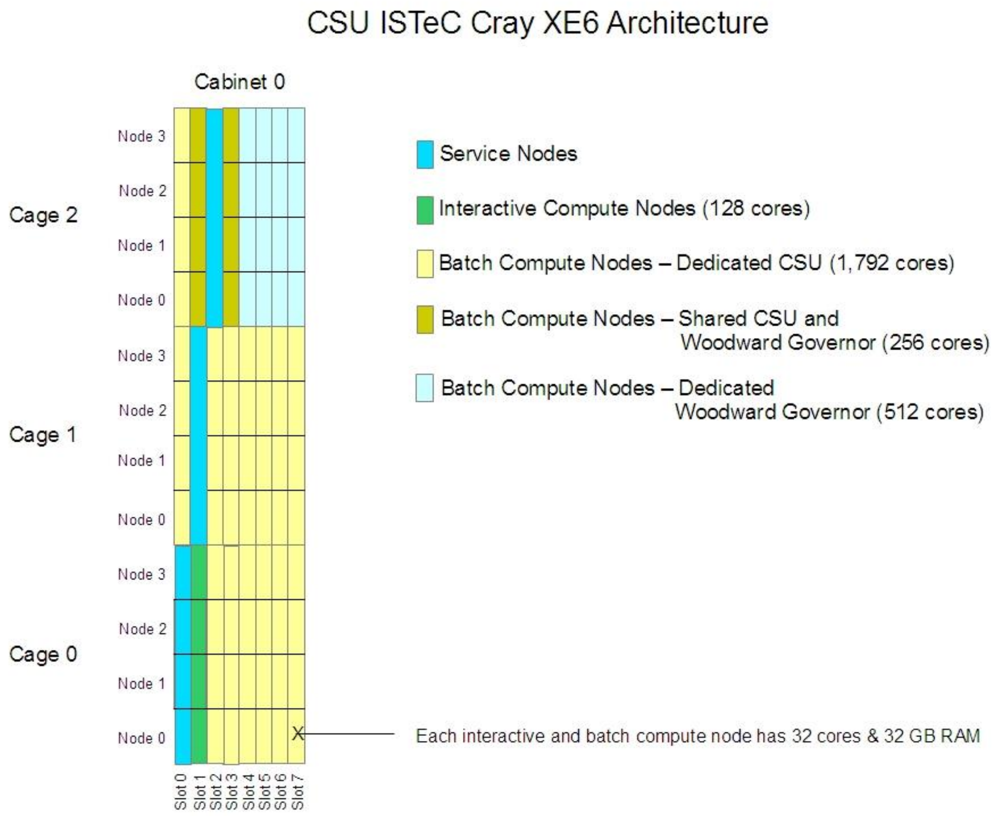
System Architecture

The ISTeC Cray HPC System is a model XE6 supercomputer. An overview of the Cray XE6 system architecture is shown below.



The Cray CPU's are 16-core AMD Interlagos processors, two of which comprise a single compute node; thus, a compute node contains 32 CPU cores. There are two primary partitions – a service partition and a compute partition. The service partition includes a login node and several system administration nodes. The compute partition includes compute nodes and a high-speed interconnect network. The compute nodes and service node access 32 TB of RAID disk space through a high-speed Fibre Channel connection. The RAID array is managed by a Lustre2 high-speed parallel file system. The remaining elements in the architecture diagram are primarily used for system administration. There are 84 total nodes (2,688 cores) on the Cray. These are split into 4 interactive compute nodes (128 cores) and 80 batch compute nodes (2,560 cores). Small, interactive jobs can be run on the interactive compute nodes, but large, long-running jobs should be run on the batch compute nodes. Users login directly to the service node, which in turn accesses the compute nodes through a high-speed Gemini network. Users cannot login to the compute nodes, but must submit jobs to them from a login node using the *aprun* command. When users access the login node, they use a full version of SUSE Enterprise Linux, but a stripped-down, more efficient Cray Linux Environment (CLE) operating system based on SUSE Enterprise Linux version 11.0 x86_64 is used on the compute nodes.

The allocation of cores is shown below:


CSU ISTeC Cray XE6 Architecture
Cabinet 0
Cage 2
Cage 1
Cage 0
Node 3
Node 2
Node 1
Node 0
Slot 0
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Service Nodes
Interactive Compute Nodes (128 cores)
Batch Compute Nodes – Dedicated CSU (1,792 cores)
Batch Compute Nodes – Shared CSU and Woodward Governor (256 cores)
Batch Compute Nodes – Dedicated Woodward Governor (512 cores)
Each interactive and batch compute node has 32 cores & 32 GB RAM

There are two primary partitions – service partition and compute partition. The service partition includes a login node and several system administration nodes. The compute partition includes compute nodes and a high-speed interconnect network. 56 batch nodes (1,792 cores) are dedicated to CSU. 16 batch node (512 cores) are dedicated to Woodward Governor Inc., a Fort Collins Engineering firm. 8

batch nodes (256 cores) are shared between CSU and Woodward Governor. Small, interactive jobs can be run on the interactive compute nodes, but large, long-running jobs should be run on the batch compute nodes. Users login directly to the service node, which in turn accesses the compute nodes through a high-speed Gemini network. Users cannot login to the compute nodes, but must submit jobs to them from a login node.

The Cray XE6 is designed to run large, parallel jobs efficiently. One nuance of this is that an entire node, consisting of 32 cores, is the smallest resource that can be used on any node in the compute partition. Stated differently, only a single job can be run at a time on any node, consisting of 32 cores. Ergo, running a scalar job on a single core may result in 'wasting' the other 31 cores on the node. Users are thus instructed to parallelize their jobs across at least 32 cores. However, there are times when the memory architecture represents a bottleneck in performance, and this represents an exception to the adage to use all of the cores on a node. In these cases, it may be advantageous to use multiple nodes, each with fewer than 32 cores in use, so as to make effective use of the memory bandwidth by scaling your job up across multiple, partially used nodes. But still the code should be run in parallel.

Website

The ISTeC Cray website is: http://istec.colostate.edu/istec_cray.
The website is updated periodically with new information about the Cray.

Accounts

You can request an account on the Cray by submitting the CSU ISTeC Cray Account Request Form provided in Appendix A.

Access

There are several methods by which you can access the Cray depending on your workstation operating system, whether you are on-campus or remote, and whether you want to login directly to the Cray or transfer files to/from the Cray.

The hostname of the Cray is: **cray2.colostate.edu**

The IP address of the Cray is: **129.82.175.4**

The Cray is accessed interactively using SSH (secure shell). File transfers to/from the Cray are handled by SFTP (secure FTP) or SCP (secure copy).

Windows OS

If you are running Windows OS (Windows XP, Windows 7), you may use Secure Shell Client and Secure File Transfer Client to login or transfer files to/from the Cray. Here are instructions to download, install, and configure this software.

1) Go to www.acns.colostate.edu
2) Under “Resources” choose “Software Downloads”
3) Choose “Site Licensed Software (Free)”
4) You will be prompted to login. Enter your CSU eID information - eName and ePassword. (If you do not have an eID, go to https://eid.colostate.edu/)
5) Under “Other Utilities,” under “Terminal Clients,” choose “Windows Secure Shell (SSH)”
6) Save the file somewhere on your workstation/laptop. It will appear as an icon “SSHClient-3.2.0.exe”.
7) Launch (double-click) the icon “SSHClient-3.2.0.exe”. Go through all of the installation steps; choosing all the default settings should work. The icons “SSH Secure Shell Client” and “SSH Secure File Transfer Client” should appear on your desktop.

Configuring SSH

1) Launch (double-click) “SSH Secure Shell Client”. Click “Quick Connect”. Enter the following information in the four data fields, and save the configuration:

    i. Host Name: cray2.colostate.edu
    ii. User Name: your Cray account username
    iii. Port Number: 22
    iv. Authentication Method: Keyboard Interactive

2) Click “Connect”. The first time you login the dialog box “Host Identification” will appear; click “Yes”. (In subsequent logins you should not see this dialog box again). In the next dialog box enter your Cray password and click OK. In the next dialog box click OK.
3) Congratulations! You should now be logged into the Cray interactively.
4) You can also easily transfer files to and from the Cray using *ssh*. In fact, the file transfer window opens in *ssh* by default. When transferring files, make sure you transfer them in ASCII mode (under the ‘Operation’ menu, go to ‘File Transfer Mode’ and select ‘ASCII’).
5) Interactive, terminal windows are opened as ‘New Terminal’ under the ‘Window’ menu. It is often useful to open up two interactive, terminal windows in addition to the file transfer window.
6) Under the ‘File’ menu, ‘Profiles,’ go to ‘Add Profile’ and add a profile for the Cray. Then, when you open *ssh*, you can access the Cray directly from the ‘Profiles’ menu.

Mac / Linux OS

If you are running Mac OS or any variant of Linux, you can open a terminal window session and enter ssh or sftp commands directly. The method by which you open a terminal window varies by OS so check your OS documentation.

To login to the Cray with ssh, in a terminal window enter:

```
ssh account@cray2.colostate.edu
```

where “account” is your Cray account name.

To transfer files to/from the Cray with sftp enter:

```
sftp account@cray2.colostate.edu
```

where “account” is your Cray account name.

Client Software

In addition to these methods, there are several popular client applications you may use to access the Cray, including PuTTy and FileZilla.

PuTTy is available at:
http://www.chiark.greenend.org.uk/~sgtatham/putty/

FileZilla is available at:
http://filezilla-project.org/

File Storage

After logging into the Cray, you are placed in your home directory. The Cray is configured to have very little space in users’ home directories, but the files in users’ home directories are backed up (indeed, these are the only files on the Cray that are backed up). Typically, users store their source code and script files in their home directories. Large files, and indeed numerous small files that are large in aggregate, should be stored in the lustrefs directory, which is NOT backed up. Because lustrefs is not backed up, files stored there may be lost if there is any problem with the Lustre file system or the disks which are used for it.

Modules

Cray uses the Modules Environment Management package to support dynamic modification of the user environment via module files. Each module file contains all the information needed to configure the shell for a particular application, making the Cray very easy to use. The advantage of using Modules is that you are not required to specify explicit paths for different executable versions or to set the $MANPATH and other environment variables manually. Instead, all the information required in order to use a given piece of software is embedded in the module file and set automatically when you load the module file. In general, the simplest way to make certain that the elements of your application development environment function correctly together is by using the Modules software and trusting it to keep track of paths and environment variables, and avoid embedding specific directory paths into your startup files, make files, and scripts. To make major changes in your user environment, such as switching to a different compiler or a different version of a library, use the appropriate Modules commands to select the desired module files.

After logging into your account, you can view the module files that are currently loaded in your shell environment by entering:

module list

To see all available module files on the Cray enter:

module avail

To load a module enter:

module load modulefile

where “modulefile” is one of the files shown in the “module avail” list.

To unload a module enter:

module unload modulefile

where “modulefile” is one of the files shown in the “module list” list.

To swap new module2 for current module1 enter:

module swap modulefile1 modeulefile2

where “modulefile1” is a file shown in the “module list” list and “modulefile2” is a file shown in the “module avail” list. For example, to switch from the Cray compiler environment to the GNU compiler environment, enter:

module swap PrgEnv-cray PrgEnv-gnu

To see the contents of a particular module enter:

module show modulefile

To see a description of a particular module enter:

module help modulefile

Compilers

The ISTeC Cray HPC System supports four compiler suites including:

- Cray Compiling Environment (Fortran, C, C++)

- The Portland Group Compilers (Parallel Fortran, C, C++)
- GNU Compilers (Fortran, C, C++)

The various compilers have different strengths and weaknesses, and their relative strengths and weaknesses are constantly changing as new revisions and updates are released. No one compiler may be best for all applications under all circumstances, nor does any one compiler always produce faster executable code than another.

By default, your account is set up to use the Cray compilers. However, you can ensure that your environment is set up for Cray compilers by entering:

```
module load PrgEnv-cray
```

To load the Portland Group compilers enter:

```
module load PrgEnv-pgi
```

To load the GNU compilers enter:

```
module load PrgEnv-gnu
```

When loading modules, if you get an error message stating that the module you are trying to load conflicts with a currently loaded module, then you should use the "module swap" command to swap out the currently loaded module and replace it with your chosen module (see previous section). For example, if the Cray compiler module is currently loaded and you wish to replace it with the Portland Group compiler module, issue the command:

```
module swap PrgEnv-cray PrgEnv-pgi
```

Because of the multiplicity of possible compilers, Cray supplies compiler drivers, wrapper scripts, and disambiguation of man pages. No matter which vendor's compiler module is loaded, always use one of the following commands to invoke the compiler:

ftn - Invokes the Fortran compiler, regardless of which compiler module is currently loaded. This command links in the fundamental libraries required in order to produce code that can be executed on the Cray compute nodes. For more information, see the ftn(1) man page.

cc - Invokes the C compiler, regardless of which compiler module is currently loaded. This command links in the fundamental header files and libraries required in order to produce code that can be executed on the Cray compute nodes. For more information, see the cc(1) man page.

CC - Invokes the C++ compiler, regardless of which compiler module is currently loaded. This command links in the fundamental header files and libraries required in order to produce code that can be executed on the Cray compute nodes. For more information, see the CC(1) man page.

C Compiler

To compile C code:

```
cc hello.c
```

C++ Compiler

To compile C++ code:

```
CC hello.C
```

Fortran Compiler

To compile Fortran code:

```
ftn hello.f90
```

Note. The Cray compiler has 5-user licenses, the Pathscale compiler has a 2-user license, and the Portland Group compiler has a 2-user license. This causes problems in situations where more than nine users who try to compile simultaneously. If the number of users attempting to compile code simultaneously exceeds the license limit, the following error message is produced:

“Unable to obtain a Cray Compiling Environment License.”

Generally, if you wait a few minutes you’ll be able to compile code again. However, if you chronically get this message please contact technical support. Or, you can use the gnu environment, which has an unlimited number of licenses. Often, extensive development is done in the gnu environment, and then preferred compiler is used to produce a production code.

Interactive Jobs

When you login to your account, in your home directory you will see a “**lustrefs**” directory and perhaps other directories and files in addition to the “lustrefs” directory. This directory is linked to the Lustre parallel file system. Serial, single-CPU jobs can be run inside or outside the “lustrefs” directory. However, parallel multiple-CPU jobs *must* be run from within the “lustrefs” directory. Due to the file storage limitations, it is recommended that users work in the lustrefs directory whenever possible.

Serial Jobs

If you want to run serial single-CPU jobs only, you can do so either inside or outside the “lustrefs” directory. Simply type the name of an executable file. It will run on a single CPU on the login node. Only small test jobs are allowed to be run in serial mode. The login node is a shared resource and long-running or resource intensive jobs will impact other users on the login node, and is therefore prohibited.

Parallel Jobs

On the Cray there are two types of compute nodes for executing parallel jobs: interactive compute nodes and batch compute nodes. Interactive compute nodes are used for relatively small jobs that are not too resource intensive. The interactive nodes can be used to test parallel code, develop code, debug code, get baseline performance metrics, and for similar activities. Batch compute nodes are used for large jobs that require substantial resources. The batch nodes are typically used for long-running production jobs.

If you want to run parallel jobs on the interactive or batch compute nodes, you must first go into the “lustrefs” directory and move all of your application code and files inside this directory. You should compile, load, and execute all parallel code from within the “lustrefs” directory. In addition, parallel jobs require at least a minimal set of OpenMP or MPI2 commands (see the Parallelization section) to execute properly on the compute nodes.

You can run parallel interactive jobs with the “**aprun**” command. The “aprun” command places and launches applications on the compute nodes. “aprun” submits jobs to the Application Level Placement Scheduler (ALPS) for placement and execution, forwards your login node environment to the assigned compute nodes, forwards signals, and manages the stdin, stdout, and stderr streams. There are numerous options for the “aprun” command but only a few will be considered here. For a complete list, see the “aprun” man page.

The “-n” option is used to specify the number of independent, distributed memory cores to allocate to a job. To run an executable file on one core enter:

```
aprun –n 1 executable
```

where “executable” is the name of an executable file (*a.out* by default).

To run an executable file independently on 32 cores enter:

```
aprun –n 32 executable
```

You may specify “-n” up to 128 cores (the total number of cores available on the interactive compute nodes). If “-n” exceeds 128, you will get an error message “apsched: request exceeds max alloc”.

To show all currently running applications (you will need to open up another window using ssh to see the jobs which are running, including your own):

```
apstat
```

To kill a running interactive job:

```
apkill apid
```

where "apid" is the application ID number as displayed by the "apstat" command. You must be the owner of a running application in order to kill it. Or, you can simply type <CTRL-C> from within the window used to launch aprun.

Batch Jobs

The ISTeC Cray HPC System includes the Torque/Moab/PBS batch queuing system for managing batch jobs. You can and should run large parallel jobs in the batch queues. The Torque scheduler load balances multiple jobs on the compute nodes to provide the most efficient utilization of the system.

Batch Queues

The Cray batch queue is currently structured as follows:

Queue Name	Priority	Max. Runtime	Max. Number Jobs per User
small	high	1 hour	20
medium	medium	24 hours	2
large	low	168 hours	1
ccm_queue	---	---	---
priority_queue	high	24 hours	1
woodward	---	---	---
woodward_ccm	---	---	---

The "small" queue is generally used for jobs that require minimal resources and run to completion in less than one hour. Jobs in this queue have high priority and go "in-and-out" of the system quickly. The "medium" queue is for jobs with modest resource requirements and that run to completion in less than one day. The "large" queue is for jobs that require significant resources and that can run for up to one week. The "ccm_queue" queue is a special queue for applications that must be run in a traditional cluster environment; see the section "Cluster Compatibility Mode (CCM)" for more information about using this queue. The "priority_queue" is used for high-priority jobs; this queue is rarely used and is used only with explicit permission from the system administrators. The "woodward" and "woodward_ccm" queues are dedicated to Woodward Governor users and are assigned only to the Woodward cores.. Use the commands below to determine which queues are available.

List all available queues (brief):

```
qstat –Q
```

List all available queues (full):

```
qstat –Qf
```

Show the status of jobs in all queues:

```
qstat
```

(Note: if there are no jobs running in any of the batch queues, this command will show nothing and just return the Linux prompt).

Show only the status of jobs that belong to your account, where “username” is your account name:

```
qstat –u username
```

Submit a job to the default batch queue:

```
qsub filename
```

where “filename” is the name of a file that contains batch queue commands.

Delete a job from the batch queues:

```
qdel jobid
```

where “jobid” is the job ID number as displayed by the “qstat” command. You must be the owner of the job in order to delete it.

Sample Batch Jobs

To use the batch queues, you must create a text file (batch script) that contains Torque/PBS commands. You submit this file with the “qsub” command:

```
qsub filename
```

where “filename” is the name of the batch text file.

Here is a sample batch script that you can use to run jobs on the Cray, or use as a template for more complex scripts.

```
#!/bin/bash
#PBS –N jobname
#PBS –j oe
#PBS –l mppwidth=32
```

```
#PBS –l walltime=1:00:00
#PBS –q small
cd $PBS_O_WORKDIR
aprun –n 32 executable
```

The first line specifies the shell environment to use for the batch job. This line is not required but it's often a good idea to explicitly name the shell. The "#PBS" mnemonic represents a PBS directive. The "-N" directive renames the output files to whatever name you specify. The "-j oe" option indicates that standard output and standard error information should be combined in a single file. The "-l mppwidth" option specifies the number of cores to allocate to your job. The number of cores should not exceed 1,792, which is the maximum number of cores available in the largest batch queue. The "-l walltime" option specifies the maximum amount of time in hours, minutes and seconds that the job may run before exiting. The "-q small" option specifies which queue the job should be run in. There are four options: "small", "medium", "large", "ccm_queue". If no queue is specified in the batch script, the job will be routed to the "small" queue by default. The PBS_O_WORKDIR environment variable is generated by Torque/PBS and contains the absolute path to the directory from which you submitted your job. This is required for Torque/PBS to find your executable files.

Both options, "-l mppwidth" and "-n", should be specified and the number of cores listed should be the same. If you specify different values for "-l mppwidth" and "-n", it will generate an error and will not run the job.

Suppose the batch commands above were contained in a file called "hello.job". When the batch script is finished, a single file "hello.job.o1234" would be created, where "1234" is a variable that represents the job ID. The file would contain both standard output and standard error in a single file. The "aprun" command submits the executable file "executable" to the Cray batch compute nodes. The "executable" is an executable file already compiled in some programming language (i.e. Fortran, C, C++).

Parallelization

The ISTeC Cray HPC System supports two forms of parallelism – OpenMP and MPI. You may choose either method, or both methods, to parallelize your code.

OpenMP

Here is an example of a basic set of OpenMP commands to run a parallel job on the Cray (assume the code is in a file “ompc.c”):

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
void main (int argc, char *argv[])
{
int nthreads, tid;
#pragma omp parallel private(nthreads, tid)
  {
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);
    if (tid == 0)
    {
      nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
    }
  }
}
```

To compile and run the job in parallel:

```
cc -o ompc ompc.c
aprun -d 32 ompc
```

or, using the default executable *a.out*:

```
cc ompc.c
aprun –d 32 a.out
```

Message Passing Interface (MPI)

Here is an example of a basic set of MPI commands to run a parallel job on the Cray (assume the code is in a file “mpic.c”):

```
#include "mpi.h"
#include "stdio.h"
void main(int argc, char *argv[])
{
   int rank
   int numprocs;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   printf("Hello from pe %d of %d\n",rank,numprocs);
   MPI_Finalize(); /* terminate the MPI environment */
}
```

To compile and run the job in parallel:

```
cc -o mpic mpic.c
aprun -n 32 mpic
```

Performance Analysis

CrayPat is a performance analysis tool that can be used to generate performance metrics for your code and applications. Here are the steps required to use CrayPat with your code.

1) After logging into your account, load the Cray “**perftools**” module:

   module load perftools

2) Compile your code with options to create an object file, i.e.:

   cc –c hello.c

   This step creates the object file “hello.o”. Regardless of the programming language used, CrayPat requires an object file in order to instrument your code.

3) Link your object files to create an executable file, i.e.:

   cc –o hello hello.o

4) Instrument your executable file with the “**pat_build**” command, i.e.:

   pat_build hello

   This step creates a file with “+pat” appended to the executable filename, i.e. “hello+pat”.

5) Run the instrumented executable file, i.e.:

   aprun –n 32 hello+pat

   This step creates a file with “.xf” appended to the instrumented executable filename, i.e. “hello+pat+PID.xf”. The “PID” portion of the filename is variable and depends on the environment in which the code is executed.

6) Generate a performance analysis report using the “**pat_report**” command, i.e.:

```
pat_report hello+pat+PID.xf
```

This step prints a text report to stdout, which includes a variety of performance metrics. The performance metrics can help you identify code "hot spots" that may be parallelized to speed up the code. However, the user is cautioned that a massive amount of information is produced as text output, and it can be very cumbersome and labor intensive to distill meaningful information in this manner.

Generally, a very large amount of information is produced by CrayPat, large enough so that a graphics program is needed to interpret it. However, that is beyond the scope of this user's manual, and the reader is referred to the Cray documentation.

Timing Jobs

The intrinsic function omp_get_wtime() returns 'wall clock' time, and is the best timing routine for high-resolution time, accurate to approximately a microsecond. It can be used for timing any job, a serial job, or a parallel job using MPI, OpenMP, or a combination of the two. It returns elapsed time since some arbitrary initial value, so elapsed wall clock time must be computed from differences in the time obtained from successive calls e.g.

```
t0 = omp_get_wtime();
… work
t1 = omp_get_wtime();
elapsed_time = t1 – t0;
```

Also, the time required to make the call to the omp_get_wtime() function is included in the timing, but may be (approximately) removed by subtraction to yield more accurate timing, as follows:

```
t0 = omp_get_wtime();
t1 = omp_get_wtime();
… work
t2 = omp_get_wtime();
elapsed_time = t2 + t0 – 2.*t1;
```

Compute Node Status

Before running jobs on the Cray, it's often useful to check the status of the interactive and batch compute nodes before actually submitting jobs. There are two useful commands to check the status of compute nodes – "**xtprocadmin**" and "**xtnodestat**".

The "xtprocadmin" command shows whether interactive and batch compute nodes are up or down. Here is sample output from the command:

```
   NID    (HEX)    NODENAME     TYPE    STATUS        MODE
    12      0xc  c0-0c0s3n0  compute        up interactive
    13      0xd  c0-0c0s3n1  compute        up interactive
    14      0xe  c0-0c0s3n2  compute        up interactive
    15      0xf  c0-0c0s3n3  compute        up interactive
    16     0x10  c0-0c0s4n0  compute        up interactive
    17     0x11  c0-0c0s4n1  compute        up interactive
    18     0x12  c0-0c0s4n2  compute        up interactive
    42     0x2a  c0-0c1s2n2  compute        up       batch
    43     0x2b  c0-0c1s2n3  compute        up       batch
    44     0x2c  c0-0c1s3n0  compute        up       batch
    45     0x2d  c0-0c1s3n1  compute        up       batch
    61     0x3d  c0-0c1s7n1  compute        up       batch
    62     0x3e  c0-0c1s7n2  compute        up       batch
    63     0x3f  c0-0c1s7n3  compute        up       batch
```

(Numerous lines have been deleted to shorten the output). You'll want to make sure that the "STATUS" column indicates the nodes are "up" before submitting jobs.

The "xtnodestat" command shows the state of interactive and batch compute nodes and whether they are already allocated to other user's jobs. Here is sample output from the command:

```
Current Allocation Status at Mon Feb 21 12:36:45 2011

     C0-0
  n3 -----dfj
  n2 -----c-i
  n1 -----b-h
c1n0 -----aeg
  n3 SSS;;;;;
  n2    ;;;;;
  n1    ;;;;;
c0n0 SSS;;;;;
    s01234567

Legend:
   nonexistent node                 S  service node
;  free interactive compute node    -  free batch compute node
A  allocated, but idle compute node ?  suspect compute node
X  down compute node                Y  down or admindown service node
Z  admindown compute node

Available compute nodes:         20 interactive,         22 batch
```

Check the man page for "xtnodestat" for a more detailed description of the output.

Running Multiple Sequential Applications

Occasionally, users want to run applications sequentially on the Cray compute nodes, often supplying different parameters or data sets for each application. This can be accomplished on the Cray as follows.

Suppose we have two applications – hello1.c and hello2.c – as follows:

```
hello1.c

#include <stdio.h>
int main(int argc, char *argv[])
{ printf("Hello world 1\n"); }
```

```
hello2.c

#include <stdio.h>
int main(int argc, char *argv[])
{ printf("Hello world 2\n"); }
```

The following batch script will run these applications sequentially on a single compute node:

```
#!/bin/bash
#PBS -N multiseq
#PBS -j oe
#PBS -l mppwidth=6
cd $PBS_O_WORKDIR
aprun -n 1 ./hello1
echo "Finished 1"
aprun -n 6 ./hello2
echo "Finished 2"
exit 0
```

In this scenario, one copy of “hello1.c” and 6 copies of “hello2.c” will be executed on the compute nodes to yield the following output:

```
Hello world 1
Application 37239 resources: utime ~0s, stime ~0s
Finished 1
Hello world 2
Hello world 2
Hello world 2
Hello world 2
Hello world 2
Hello world 2
Application 37240 resources: utime ~0s, stime ~0s
Finished 2
```

Note: To run multiple sequential applications, the number of processors you specify as an argument to qsub must be equal to or greater than the **largest number** of processors required by a single invocation of aprun in your script. For example, in the sample above, the “-l mppwidth” value is 6 because the largest aprun “-n” value is 6.

Running Multiple Parallel Applications

Occasionally, users want to run multiple applications in parallel on the Cray compute nodes, often supplying different parameters or data sets for each application. This can be accomplished on the Cray as follows.

Suppose we have two applications – hello1.c and hello2.c – as follows:

```
hello1.c

#include <stdio.h>
int main(int argc, char *argv[])
{ printf("Hello world 1\n"); }
```

```
hello2.c

#include <stdio.h>
int main(int argc, char *argv[])
{ printf("Hello world 2\n"); }
```

The following batch script will run these applications in parallel on the compute nodes:

```
#!/bin/bash
#PBS -N multipar
#PBS -j oe
#PBS -l mppwidth=6
cd $PBS_O_WORKDIR
aprun -n 1 ./hello1 &
echo "Finished 1"
aprun -n 6 ./hello2 &
echo "Finished 2"
wait
exit 0
```

In this scenario, one copy of “hello1.c” and six copies of “hello2.c” will be executed to yield the following output:

```
Finished 1
Finished 2
Hello world 2
Hello world 2
Hello world 2
Hello world 2
Hello world 2
Hello world 2
Application 37245 resources: utime ~0s, stime ~0s
Hello world 1
Application 37246 resources: utime ~0s, stime ~0s
```

Note how the output of the two applications is interspersed randomly in the output file. The application threads run in parallel on separate compute nodes and the order in which they run and finish will vary from one run to the next.

Note: If you are running multiple parallel applications, the number of processors must be equal to or greater than the total number of processors specified by calls to aprun. For example, the “-l mppwidth” value is 6 because the largest aprun “-n” value is 6.

Cluster Compatibility Mode (CCM)

The Cray system architecture is non-traditional in the sense that compute nodes are served through a shared-root file system, proprietary SeaStar interconnect, and Compute Node Linux (CNL), which is a lightweight SUSE Linux kernel. Some applications require more traditional Linux cluster services (ssh, rsh, ypbind, nscd, ldap, etc.) and full SUSE Linux on the compute nodes to run correctly without a significant effort in porting code to the CNL environment. Cluster Compatibility Mode (CCM) is a Cray solution for providing this Linux cluster environment.

To use CCM you must submit a batch script to the “ccm_queue” queue. Here is a sample script:

```
#!/bin/bash
#PBS -N ccm
#PBS -l mppwidth=32
#PBS -l mppnppn=12
#PBS -j oe
#PBS -q ccm_queue
# set mppwidth to the number of cores allocated to the job
# set mppnppn to the number of cores per compute node (<=32)
cd $PBS_O_WORKDIR
source /opt/modules/default/init/bash
module load ccm
ccmrun appname -n32
```

The option “#PBS –q ccm_queue” routes the job to the “ccm_queue” queue. The “source” statement is required and loads the correct bash shell environment. The “module” statement is required and loads the CCM module environment. The “ccmrun” statement runs your application in CCM. Most applications include a parameter that specifies the number of cores to allocate to the job; this value should be set equal to “-l mppwidth”. There are several ways to invoke CCM and run jobs in the CCM environment. If you have issues with CCM please contact technical support for help.

OpenMPI

OpenMPI (www.open-mpi.org) is an opensource high-performance computing version of MPI (www.mcs.anl.gov/research/projects/mpi). Some applications require OpenMPI libraries for parallel execution in lieu of the Cray’s default implementation of MPICH2. To use OpenMPI on the Cray:

1) Add the line

   ```
   export PATH=/apps/openmpi-1.8.4/bin:$PATH
   ```

   in the file “.bash_profile”. “.bash_profile” is a hidden file located in your home directory. Use “ls –a” to view hidden files. After updating “.bash_profile”, source the file (“source .bash_profile”) or logoff and log back in again.

2) Include the header file “/apps/openmpi-1.8.4/include/mpi.h” in your C/C++ source code.

3) On the Cray, the directory “/apps/craytools/openmpi” includes sample template code that you can modify for OpenMPI. The file “buildhost_openmpi” is a script that creates a host file, “hostlist”, which is used by OpenMPI to determine a list of compute nodes where MPI threads will run. Do not modify “buildhost_openmpi”. Simply copy it to a directory where you intend to run OpenMPI jobs. The file “openmpi.c” is a simple example of C code that uses OpenMPI. The file “openmpi.job” is a sample batch script that shows how to compile and run “openmpi.c”. Note the use of the “mpicc” compiler and “mpirun” command. You should submit the batch script with “qsub openmpi.job”.

Appendix A

CSU ISTeC Cray HPC System
Account Request Form

Name:	____
Department/Affiliation:	____
Phone:	____
Email:	____
Preferred login name:	____

If a CSU employee, your CSU ID: ____________________________________
(a 9-digit number beginning with an ‘8’, found on your employee ID card)

Account requirements:

1) I agree to acknowledge “This research utilized the CSU ISTeC Cray HPC System supported by NSF Grant CNS-0923386” in any publications that result from research using the CSU ISTeC Cray.

2) I will supply complete citations and electronic reprints of above defined publications to the Chair of the ISTeC HPC System Management and Allocations Committee, H.J. Siegel, at HJ@ColoState.edu (this is a requirement of the ISTeC Cray NSF funding).

3) Once per year, no later than June 1, I agree to send a descriptive paragraph about each research project that has utilized the ISTeC Cray HPC System to HJ@ColoState.edu. (this is a requirement of the ISTeC Cray NSF funding).

Send form via email (see below) or via surface mail to the Cray System Administrator, Richard Casey, at ISTeC, Campus Delivery 1873, or hand deliver to the Computer Science building room 366. Please allow 3 days for implementation of a new account.

If you need assistance with this form please contact:
Richard Casey, Phone: 970-492-4127, richard.casey@colostate.edu

Appendix B

Technical Support

For technical support with the Cray please contact:

Wimroy D’Souza (software applications)
Phone: 970-491-4941
Email: wimroy.dsouza@colostate.edu

Dan Hamp (system administration)
Phone: 970-491-3263
Email: daniel.hamp@colostate.edu

Richard Casey
Phone: 970-492-4127
Cell: 970-980-5975
Email: richard.casey@colostate.edu