Cray Programming Environment User's Guide

S–2529–116

RECORD OF REVISION

S–2529–116 Published June 2014 Supports Cray XC series systems running Cray Linux Environment (CLE) release 5.1 or later and Cray XE and Cray XK systems running Cray Linux Environment (CLE) release 4.2 or later. Supports Intel® Xeon® Phi™ in autonomous and offload modes.

S–2529–114 Published March 2014 Supports Cray XC30 and Cray XC30-AC systems running Cray Linux Environment (CLE) release 5.0 or later and Cray XE and Cray XK systems running Cray Linux Environment (CLE) release 3.1 or later. Supports Intel® Xeon® Phi™ in autonomous mode only.

S–2529–111 Published December 2013 Supports Cray XC30 and Cray XC30-AC systems running Cray Linux Environment (CLE) release 5.0 or later and Cray XE and Cray XK systems running Cray Linux Environment (CLE) release 3.1 or later.

S–2529–107 Published July 2013 Supports Cray XC30 and Cray XC30-AC systems running Cray Linux Environment (CLE) release 5.0 or later and Cray XE and Cray XK systems running Cray Linux Environment (CLE) release 3.1 or later.

S–2529–103 Published March 2013 Supports Cray XC30 and Cray XC30-AC systems running Cray Linux Environment (CLE) release 5.0 or later and Cray XE and Cray XK systems running Cray Linux Environment (CLE) release 3.1 or later.

Changes to this Document

Cray Programming Environment User's Guide S–2529–116

This guide replaces the *Cray Application Developer's Environment User's Guide* and supports version 6.32 (and later) of the Cray Application Development Environment and version 1.16 (and later) of the Cray Developer Toolkit.

Revised information

- The Intel Xeon Phi targeting instructions have been revised and expanded. See Targeting for Intel Xeon Phi on page 31.

Contents

Page

Page

Page

Figures

Tables

Page

Introduction [1]

This guide describes the software environment and tools used to develop, debug, and run applications on Cray XT, Cray XE, Cray XK, and Cray XC series systems. It is intended as a general overview and introduction to the Cray system for new users and application programmers.

This guide is intended to be used in conjunction with *Workload Management and Application Placement for the Cray Linux Environment* (S–2496), which describes the Application Level Placement Scheduler (ALPS) and `aprun` command in greater detail.

The information contained in this guide is of necessity fairly high-level and generalized, as the Cray platform supports a wide variety of hardware nodes as well as many different compilers, debuggers, and other software tools. Therefore, system hardware and software configurations vary considerably from site to site. For specific information about your site and its installed hardware, software, and usage policies, contact your site administrator.

1.1 What You Must Know About Your System

Because of processor and network interface differences, you can invoke different options when compiling and executing your programs. This guide focuses on compilation differences. Execution differences are discussed in *Workload Management and Application Placement for the Cray Linux Environment* (S–2496).

1.1.1 Processor Type

The Cray XT, Cray XE, and Cray XK systems use 64-bit AMD Opteron processors as the basic computational engines. Cray XC series systems use Intel Xeon processors. The number of computational units per node varies from system to system and sometimes from cabinet to cabinet within a system.

- Cray XT6 and Cray XE6 systems use two AMD Magny-Cours, two AMD Interlagos, or two AMD Abu Dhabi Opteron processors per compute node.
- Cray XK systems combine one AMD Interlagos or Abu Dhabi Opteron processor and one NVIDIA Tesla or Kepler GPU per compute node.

- Cray XC series systems use two Intel Xeon processors per compute node. Hybrid systems may combine Intel Xeon CPUs with NVIDIA GPUs or Intel® Xeon® Phi™ coprocessors on compute nodes.

1.1.2 Compute Units and CPUs

At a high level, AMD and Intel microprocessors differ from each other in the degree and type of resource sharing used in their diverse architectures. *Compute unit affinity* gives Cray users more control over job scheduling and placement to either eliminate or take advantage of the shared resources in the designs of these microprocessors.

AMD Interlagos and AMD Abu Dhabi processors consist of up to 8 Bulldozer modules. Each Bulldozer module consists of 2 integer cores and a shared floating point unit (FPU) and shared L2 cache. Certain applications may see performance benefit by using only one integer core per compute unit, as opposed to two, thereby not sharing the FPU or L2 cache located on the same Bulldozer module with other threads or PEs.

Intel processors contain Hyper-Threading Technology (HTT). Using Intel terms, each processor consists of multiple cores, each of which contains multiple threads. Each thread contains a unique set of registers but shares execution resources with one or more other threads within the same core. A set of threads sharing the same execution resource are called a core. Again, the sharing of execution resources has performance implications and some applications may see performance benefit by not sharing execution resources.

Because AMD and Intel use overlapping terminology to describe distinct entities, Cray uses a terminology mapping that unifies the common concepts for scheduling and placement purposes as shown below.

Table 1. AMD, Intel, Cray, and BASIL Terminology

AMD	Intel	Cray
Bulldozer module	core	compute unit
core	thread	CPU

Cray Inc. will use the term *CPU* to refer to both an AMD core and an Intel thread. Cray Inc. will use the term *Compute Unit* (CU) to indicate a grouping of one or more CPUs that share execution resources, thus CU refers to the AMD Bulldozer module in the Interlagos/Abu Dhabi context and the Intel core in the Sandy Bridge context.

Current HTT-enabled Intel microprocessors, such as Sandy Bridge, contain 2 CPUs per CU. Current AMD Interlagos and Abu Dhabi microprocessors contain 2 CPUs per CU. Earlier architectures (Magny-Cours and prior) contain 1 CPU per CU.

Please see *Using Compute Unit Affinity on Cray Systems* for more information.

1.1.3 CPU Numbering

Though the Intel Sandy Bridge and the AMD Abu Dhabi microprocessors both contain 2 CPUs per CU, Intel and AMD number the CPUs differently. Users should be aware that this difference in the CPU numbering scheme affects the order in which CPUs are reserved and the order in which software threads/PEs are assigned to CPUs.

AMD numbers the CPUs starting with the first CPU on the first CU, then the second CPU on the first CU, then moves to the first CPU on the second CU, and so on, incrementally numbering all the CPUs first on one socket, then the other socket on the node.

Intel first numbers the first CPU in each compute unit, across CUs in all sockets on the node, then continues with the second CPU in each compute unit, across all sockets in the node. For example, the first CPU in the first CU is CPU0, then the first CPU in the second CU is CPU1, and so on though all the CUs on all sockets on the node. Then the numbering wraps back to the second CPU on the first CU, then the second CPU on the second CU, and finishing with the second CPU on the last CU on the last socket of the node.

1.1.4 Which Network ASIC?

The Cray network application-specific integrated circuit (ASIC) provides an interface between the processors and the interconnection network with support for message passing, one-sided operations, and global address space programming models.

- Cray XT systems use SeaStar™ or SeaStar2+™ ASICs to manage inter-processor communications
- Cray XE and Cray XK systems use Gemini™ ASICs to manage inter-processor communications
- Cray XC30 systems use Aries™ ASICs for inter-processor communications

Because of the differences in the network ASICs and accompanying network APIs, applications that use inter-process communication, use different versions of the libraries which implement inter-process communication.

Specifically, SeaStar (Cray XT) systems, Gemini (Cray XE and Cray XK) and Aries (Cray XC30) systems use different versions of the MPI and SHMEM libraries. Also, the compilers' inter-process communication functionality depends on network specific versions of the network APIs.

The differences between the versions of MPI and SHMEM are discussed in more detail in MPT on page 65.

For more information about the Generic Network Interface (GNI) and Distributed Shared Memory Application (DMAPP) APIs, see *Using the GNI and DMAPP APIs*.

1.1.5 Which GPU or Coprocessor?

Systems equipped with hybrid CPU/GPU nodes require different libraries, depending on which GPU accelerator or coprocessor is installed. At this time Cray systems support NVIDIA Fermi (K20), Kepler (GK110), and Tesla (K40) GPUs and the Intel® Xeon® Phi™ coprocessors codenamed Knights Corner. If necessary, use the `cnselect -L subtype` command to determine which GPUs or coprocessors are installed on your system. For example:

```
$ cnselect -L subtype
nVidia_Kepler
```

NVIDIA Fermi (K20) GPUs are supported by the `craype_accel_nvidia20` module, while NVIDIA Kepler (GK110) and Tesla (K40) GPUs are supported by the `craype_accel_nvidia35` module.

Intel Xeon Phi coprocessors are not accelerators, and therefore do not use a `craype_accel` module.

If your system has mixed nodes, you can use the `cnselect` command to identify which nodes have which accelerators or coprocessors. For example, to find just the nodes with Intel Xeon Phi coprocessors, enter this command:

```
> cnselect -e subtype.eq.Intel_KNC
36-43
```

This information can be used later to specify placement of applications on nodes having the desired hardware.

1.1.6 Which Operating System?

All current Cray systems run the Cray Linux Environment (CLE) operating system on the login nodes and a lightweight kernel, Compute Node Linux (CNL), on the compute nodes. Some of the options available to application developers vary depending on which version of CLE is currently running on the system.

- Cray XC30 systems run CLE release 5.0 or later.
- Cray XK systems run CLE release 4.0 or later.
- Cray XE5 and Cray XE6 systems run CLE release 3.1 or later.
- Cray XT6 and Cray XT6m systems run CLE release 3.0 or later.

If you are not certain which release your site is using, check the MOTD (message of the day) when you log in. If the information is not there, there are several other ways to determine the CLE release number.

- On CLE 3.0 and later systems, `cat` the contents of the `/etc/opt/cray/release/clerelease` file. This returns the CLE release and update number.
- Cray Development and Login (CDL) nodes do not run CLE and do not have this file. On those machines, you will need to be on the actual compute node to check the `/etc/opt/cray/release/clerelease` file. For example:

```
qsub -I -lmppwidth=0
cat /etc/opt/cray/release/clerelease
```

1.1.7 What Is a Compute Node?

From the application developer's point of view, a Cray system is a tightly integrated network of thousands of nodes. Some are dedicated to administrative or networking functions and therefore off-limits to application programmers. Programmers typically use the following node types:

- *login nodes* — The node you access when you first log in to the system. Login nodes offer the full Cray Linux Environment (CLE) operating system, are used for basic development tasks such as editing files and compiling code, generally have access to the network file system, and are shared resources that may be used concurrently by multiple users.

  Login nodes are also sometimes called *service nodes*.
- *Cray Development and Login (CDL) nodes* — External Services system, either managed or unmanaged. (Formerly *esLogin* nodes.)
- *compute nodes* — The nodes on which production jobs are executed. Compute nodes run CNL, can be accessed only by submitting jobs through a batch management system (e.g., PBS Professional, Moab HPC Suite, TORQUE Resource Manager, or Platform LSF), generally have access only to the high-performance parallel file system and are dedicated resources, exclusively yours for the duration of the batch reservation.

When new users first begin working on the Cray system, this difference between login/CDL and compute nodes can be confusing. Remember, when you first log in to the system, you are placed on a login node. You cannot execute parallel programs on the login node, nor can you directly access files stored on the high-performance parallel file system.

Instead, use your site's batch system to place parallel programs on the compute nodes, either from the login node or from a mount-point on the parallel file system.

Note: You can execute *serial* (single-process) programs on login nodes, but executing large or long-running serial programs on login nodes is discouraged, as login nodes are shared resources.

1.1.8 Which File System?

All Cray systems require the use of a high-performance parallel file system. Most sites currently use the Lustre File System, although others are also supported. All examples shown in this guide were developed on a Lustre file system using Lustre commands. Before copying any examples from this guide verbatim, verify which file system your site uses and what your site's policies are regarding home directories, scratch space, disk quotas, backup policies, and so on. If required, adjust the instructions accordingly.

1.1.9 Which Batch System?

Cray systems typically operate under the control of a batch system such as PBS Professional, OpenPBS, Moab HPC Suite, TORQUE Resource Manager, or Platform LSF. All examples shown in this guide were developed using either PBS Pro 11.0, Moab HPC Suite, or TORQUE Resource Manager. Before copying any examples from this guide verbatim, verify which batch system your site uses and if required, adjust the instructions accordingly.

1.2 Logging In

User account setup and authentication policies vary widely from site to site. In general, you must contact your site administrator to get a login account on the system. Any site-specific security or authentication policies (for example, the correct use of an RSA SecurID token) should be explained to you at that time.

Once your user account is created, log in to the Cray system using SSH (Secure Shell), protocol version 2. SSH is a remote login program that encrypts all communications between the client and host and replaces the earlier `telnet`, `rlogin`, and `rsh` programs.

1.2.1 UNIX or Linux Users

If you use a UNIX or Linux workstation, the `ssh` utility is generally available at any command line and documented in the `ssh`(1) man page. To log in to the Cray system, enter:

```
% ssh -X hostname
```

The `-X` option enables X11 display forwarding. Automatic forwarding of X11 windows is highly recommended as many application development tools use GUI displays.

On some systems, you may be required to enter your user ID as well. This can be done in several different ways. For example:

`% ssh -X -l`***userID hostname***

Or

`% ssh -X` ***userID@hostname***

In any case, after you SSH to the system, you may have to answer one or more RSA or password challenges, and then you are logged into the system. A series of system status and MOTD (message of the day) messages may display, after which you are placed in your home directory on a login node.

`/users/`*userID>*

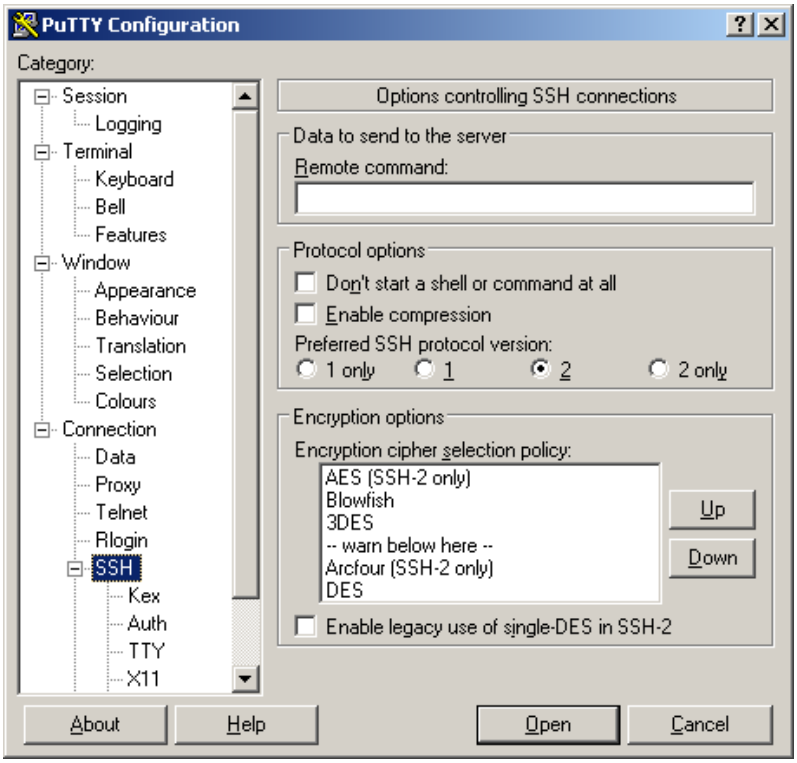
You are now ready to begin working. Jump to Navigating the File Systems on page 23.

1.2.2 Windows Users

If you use a Windows personal computer, you first need to obtain and install a client program for your system that supports SSH protocol 2, such as PuTTY for Windows. Your system administrator should be able to provide a list of accepted clients.

You may need to configure your client to support SSH protocol 2 and X11 forwarding. For example, if you are using PuTTY, you may need to click **SSH** in the left pane to see the preferred SSH protocol version:

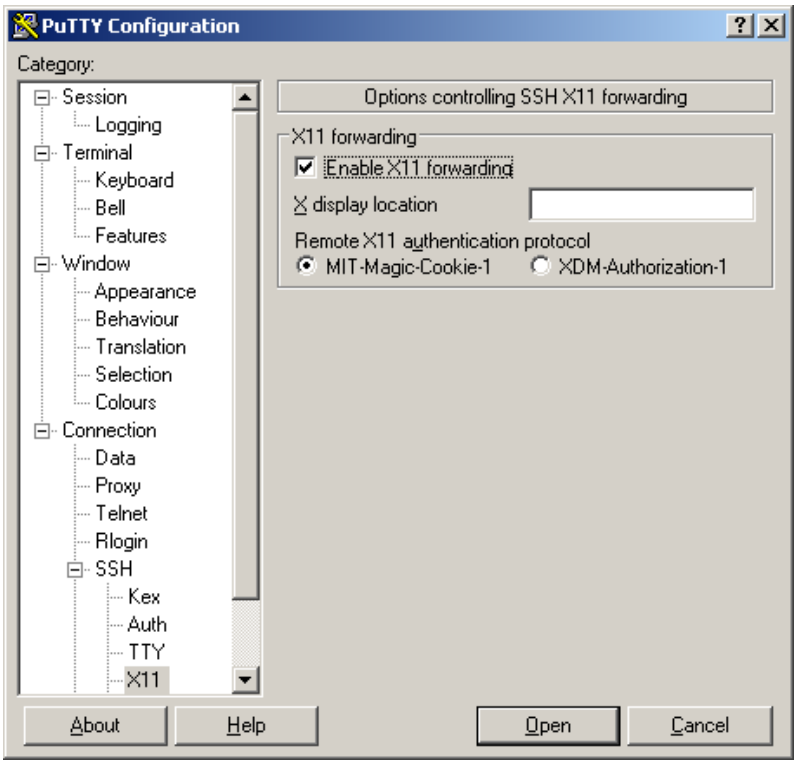
Figure 1. Selecting SSH Protocol



Verify that the **Preferred SSH protocol version** is set to 2.

Then click **X11** in the left pane to view the SSH X11 forwarding options:

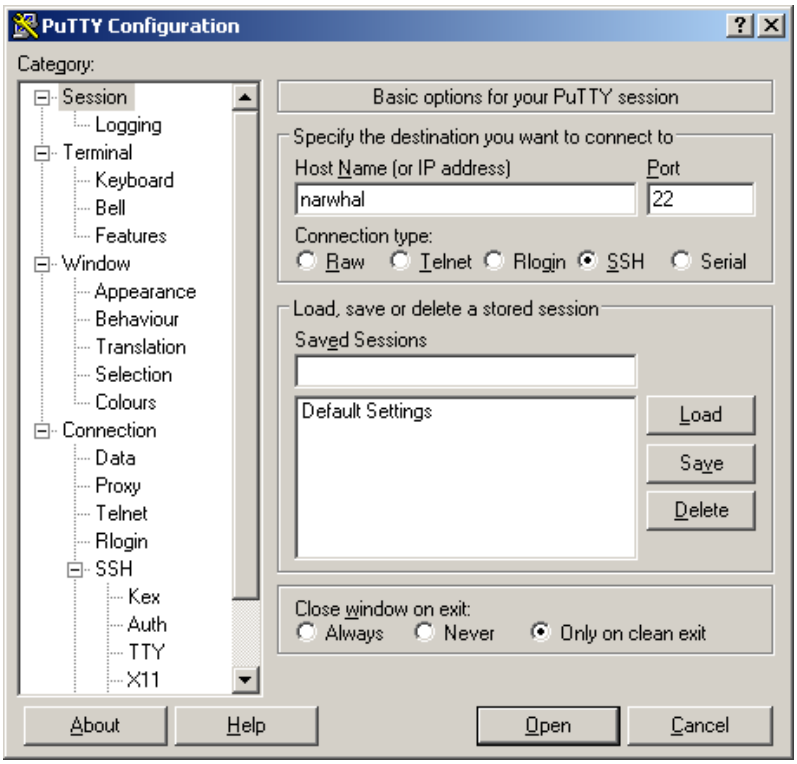
Figure 2. Enabling X11 Forwarding



If necessary, click the **Enable X11 forwarding** checkbox.

Then click **Session** in the left pane to return to the **Basic options** window.

Figure 3. Logging In



Enter the *hostname* in the **Host Name** field and click the **Open** button to begin your SSH session.

You may need to enter your *userID* and answer one or more RSA or password challenges, and then you are logged into the system. A series of system status and MOTD (message of the day) messages may display, after which you are placed in your home directory on a login node.

`/users/`*userID*`>`

You are now ready to begin working on the Cray system.

1.2.3 Apple Users

The Apple OS X operating system is based on UNIX. Therefore, to log in to the Cray system, open the Terminal application, and then use the `ssh` command to connect to the Cray system.

```
% ssh -X hostname
```

The `-X` option enables X11 display forwarding with X11 security extension restrictions. Automatic forwarding of X11 windows is highly recommended as many application development tools use GUI displays.

Note: The version of SSH found in OS X also supports the `-Y` argument, as well as the `-X` argument. The `-Y` argument enables "trusted" X11 forwarding and may work better than `-X` for some users.

On some systems, you may be required to enter your user ID as well. This can be done in several different ways. For example:

```
% ssh -X -luserID hostname
```

Or

```
% ssh -X userID@hostname
```

In any case, after you SSH to the system, you may have to answer one or more RSA or password challenges, and then you are logged into the system. A series of system status and MOTD (message of the day) messages may display, after which you are placed in your home directory on a login node.

`/users/`*userID*`>`

You are now ready to begin working on the Cray system.

1.3 Navigating the File Systems

When you first log in to the Cray system, you are placed in your home directory on a login node.

`/users/`*userID*`>`

At this point you have access to all the features and functions of the full Cray Linux Environment (CLE) operating system, such as the `sftp` and `scp` commands. Typically you will also have access to your full network file system. On most systems your home directory on the login node is defined as the environment variable `$HOME`, and this variable can be used in any file system command. For example, to return to your home directory from any other location in the file system(s), enter this command:

```
> cd $HOME
```

Remember, you can edit files, manipulate files, compile code, execute serial (single-process) programs, and otherwise work in your home directory on the login node. However, you cannot execute parallel programs on the login node.

Parallel programs must be run on the compute nodes, under the control of the batch system, and generally while mounted on the high-performance parallel file system. To do this, you must first identify the *nids* (node IDs) of the file system mount points. On the Lustre file system, this can be done in one of two ways.

Either enter the `df -t lustre` command to find the Lustre nodes and get a summary report on disk usage:

```
users/userID> df -t lustre
Filesystem           1K-blocks     Used    Available   Use% Mounted on
8@ptl:/narwhalnid8   8998913280 6946443260 1595348672  82% /lus/nid00008
```

Or enter the `lfs df` command to get more detailed information:

```
users/userID> lfs df
UUID                 1K-blocks      Used  Available  Use% Mounted on
nid00008_mds_UUID    179181084   2675664  166265604    1% /lus/nid00008[MDT:0]
ost0_UUID            1124864160 895207088 172517160   79% /lus/nid00008[OST:0]
ost1_UUID            1124864160 838067380 229656540   74% /lus/nid00008[OST:1]
ost2_UUID            1124864160 826599428 241124820   73% /lus/nid00008[OST:2]
ost3_UUID            1124864160 827914052 239801932   73% /lus/nid00008[OST:3]
ost4_UUID            1124864160 964324672 103398548   85% /lus/nid00008[OST:4]
ost5_UUID            1124864160 932986208 134738024   82% /lus/nid00008[OST:5]
ost6_UUID            1124864160 832715148 235009164   74% /lus/nid00008[OST:6]
ost7_UUID            1124864160 828631656 239092572   73% /lus/nid00008[OST:7]

filesystem summary:  8998913280 6946445632 1595338760   77% /lus/nid00008
```

Note: The above commands are specific to the Lustre high-speed parallel file system. If your site uses a different file system, adjust the instructions accordingly.

In this example, the Lustre mount point is `/lus/nid00008`. If you `cd` to this mount point:

```
users/userID> cd /lus/nid00008
Directory: /lus/nid00008
/lus/nid00008>
```

you are now on the high-performance parallel file system. At this point you can edit and manipulate files, compile code, and so on; and you can also execute programs on the compute nodes, typically by using the batch system.

Using Modules [2]

The Cray system uses the Modules environment management package to support dynamic modification of the user environment via *modulefiles*. Each modulefile contains information needed to configure the shell for a particular application. To make major changes in your user environment, such as switching to a different compiler, use the appropriate Modules commands to select the desired modulefiles.

The advantage in 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 modulefile and set automatically when you load the modulefile.

The simplest way to make certain that the elements of your application development environment function correctly together is by using the Modules software to keep track of paths and environment variables, rather than embedding specific directory paths into your startup files, makefiles, and scripts.

2.1 What Is Loaded Now?

When you first log in to the Cray system, a set of site-specific default modules is loaded. This set varies depending on system hardware, operating system release level, site policies, and installed software. To see which modules are currently loaded on your system, use the `module list` command.

```
users/yourname> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.7
  2) nodestat/2.2-1.0501.47138.1.78.ari
  3) sdb/1.0-1.0501.48084.4.48.ari
  4) alps/5.1.1-2.0501.8507.1.1.ari
  5) MySQL/5.0.64-1.0000.7096.23.2
  6) lustre-cray_ari_s/2.4_3.0.80_0.5.1_1.0501.7664.13.1-1.0501.14774.17.1
  7) udreg/2.3.2-1.0501.7914.1.13.ari
  8) ugni/5.0-1.0501.8253.10.22.ari
  9) gni-headers/3.0-1.0501.8317.12.1.ari
 10) dmapp/7.0.1-1.0501.8315.8.4.ari
 11) xpmem/0.1-2.0501.48424.3.3.ari
 12) hss-llm/7.1.0
 13) Base-opts/1.0.2-1.0501.47945.4.2.ari
 14) craype-network-aries
 15) craype/2.1.0.4
 16) cce/8.2.4
 17) totalview-support/1.1.5
 18) totalview/8.12.0.1
 19) cray-libsci/12.2.0.2
 20) pmi/5.0.2-1.0000.9906.117.2.ari
 21) rca/1.0.0-2.0501.48090.7.46.ari
 22) atp/1.7.1
 23) PrgEnv-cray/5.1.29
 24) cray-mpich/6.2.2
 25) craype-sandybridge
 26) moab/7.2.6-r12-b152-SUSE11
 27) torque/4.2.6
```

This list breaks down into three groups: operating system modules, programming environment modules, and support modules. For example, the `craype-sandybridge` module indicates that this development environment is set up to develop code for use on Sandy Bridge processors, while the `PrgEnv-cray` module indicates that the Cray Programming Environment, which includes the Cray Compiling Environment (CCE), is currently loaded.

2.2 What Is Available?

To see what modulefiles are available on your system, enter the command:

% **module avail** [*string*] [-*subsetflag*]

The `module avail` command produces an alphabetical listing of every modulefile in your `module use` path and has no option for "grepping." Therefore, it is usually more useful to use the command with an *string* argument. For example, if you are looking for a list of the available programming environments, you would enter this command:

```
users/yourname> module avail PrgEnv

---------------------------------------- /opt/cray/modulefiles -----------------------------------
PrgEnv-cray/5.1.08           PrgEnv-gnu/5.1.18           PrgEnv-intel/5.1.29
PrgEnv-cray/5.1.10           PrgEnv-gnu/5.1.21           PrgEnv-intel/5.2.07
PrgEnv-cray/5.1.11           PrgEnv-gnu/5.1.22           PrgEnv-intel/5.2.08
PrgEnv-cray/5.1.12b          PrgEnv-gnu/5.1.23a          PrgEnv-intel/5.2.09
PrgEnv-cray/5.1.14           PrgEnv-gnu/5.1.24           PrgEnv-intel/5.2.10
PrgEnv-cray/5.1.15           PrgEnv-gnu/5.1.25           PrgEnv-intel/5.2.12
PrgEnv-cray/5.1.16           PrgEnv-gnu/5.1.26           PrgEnv-intel/5.2.13(default)
PrgEnv-cray/5.1.17           PrgEnv-gnu/5.1.28           PrgEnv-pgi/5.1.08
PrgEnv-cray/5.1.18           PrgEnv-gnu/5.1.29           PrgEnv-pgi/5.1.10
PrgEnv-cray/5.1.21           PrgEnv-gnu/5.2.07           PrgEnv-pgi/5.1.11
PrgEnv-cray/5.1.22           PrgEnv-gnu/5.2.08           PrgEnv-pgi/5.1.12b
PrgEnv-cray/5.1.23a          PrgEnv-gnu/5.2.09           PrgEnv-pgi/5.1.14
PrgEnv-cray/5.1.24           PrgEnv-gnu/5.2.10           PrgEnv-pgi/5.1.15
PrgEnv-cray/5.1.25           PrgEnv-gnu/5.2.12           PrgEnv-pgi/5.1.16
PrgEnv-cray/5.1.26           PrgEnv-gnu/5.2.13(default)  PrgEnv-pgi/5.1.17
PrgEnv-cray/5.1.28           PrgEnv-intel/5.1.08         PrgEnv-pgi/5.1.18
PrgEnv-cray/5.1.29           PrgEnv-intel/5.1.10         PrgEnv-pgi/5.1.21
PrgEnv-cray/5.2.07           PrgEnv-intel/5.1.11         PrgEnv-pgi/5.1.22
PrgEnv-cray/5.2.08           PrgEnv-intel/5.1.12b        PrgEnv-pgi/5.1.23a
PrgEnv-cray/5.2.09           PrgEnv-intel/5.1.14         PrgEnv-pgi/5.1.24
PrgEnv-cray/5.2.10           PrgEnv-intel/5.1.15         PrgEnv-pgi/5.1.25
PrgEnv-cray/5.2.12           PrgEnv-intel/5.1.16         PrgEnv-pgi/5.1.26
PrgEnv-cray/5.2.13(default)  PrgEnv-intel/5.1.17         PrgEnv-pgi/5.1.28
PrgEnv-gnu/5.1.08            PrgEnv-intel/5.1.18         PrgEnv-pgi/5.1.29
PrgEnv-gnu/5.1.10            PrgEnv-intel/5.1.21         PrgEnv-pgi/5.2.07
PrgEnv-gnu/5.1.11            PrgEnv-intel/5.1.22         PrgEnv-pgi/5.2.08
PrgEnv-gnu/5.1.12b           PrgEnv-intel/5.1.23a        PrgEnv-pgi/5.2.09
PrgEnv-gnu/5.1.14            PrgEnv-intel/5.1.24         PrgEnv-pgi/5.2.10
PrgEnv-gnu/5.1.15            PrgEnv-intel/5.1.25         PrgEnv-pgi/5.2.12
PrgEnv-gnu/5.1.16            PrgEnv-intel/5.1.26         PrgEnv-pgi/5.2.13(default)
PrgEnv-gnu/5.1.17            PrgEnv-intel/5.1.28
```

One module is usually designated as the default version. Whether this is the most recent version of this module depends on your site's policies. Some sites always make the newest version the default, while others wait until after the new version has been tested and proven bug- and dependency-free.

Whenever a newer version of a module is installed, the older versions continue to remain available, unless the site administrator has explicitly chosen to delete them.

The [-*subsetflag*] option lets you list a subset of available modules. The following flags may be used alone or in combinations:

`-U`	List user modules
`-D`	List the current default modules
`-T`	List tool modules (debuggers, performance analysis utilities, and the like)
`-L`	List library modules (see Chapter 6, Libraries on page 55)
`-P`	List Programming Environment (compiler) modules
`-X`	List CPU and network targeting modules (Barcelona, Magny-Cours, Interlagos, and the like)

2.3 Loading and Unloading Modulefiles

If a PrgEnv module **is** already loaded in your module environment, then you must first unload the currently loaded PrgEnv module before loading a different version. For example, to change from the default version of the CCE compiler suite to another version, use the `module unload` command to remove the version currently loaded.

```
users/yourname> module unload PrgEnv-cray
```

Use the `module load` command to load a specific version.

```
users/yourname> module load PrgEnv-cray/version
```

If a PrgEnv module is not already loaded, this command loads the currently defined default version of the `PrgEnv-intel` module:

```
users/yourname> module load PrgEnv-intel
```

This command loads `PrgEnv-intel/`*version*`(default)` module:

```
users/yourname> module load PrgEnv-intel/version
```

Modules may be linked and related. If you enter the `module list` command after changing the programming environment, you may see that in addition to the programming environment version change, the supporting product versions may also have changed.

2.4 Swapping Compiler Modulefiles

Alternatively, you can use the `module swap` or `module switch` command to unload one module and load the comparable module. For example, to switch from the PGI to the Cray Programming Environment, enter this command:

```
users/yourname> module swap PrgEnv-pgi PrgEnv-cray
```

The `module list` command will show that a different set of supporting modules have been also been loaded automatically.

To swap to a non-default version of the CCE compiler:

```
users/yourname> module swap cce cce/8.2.2
```

2.5 Swapping Other Programming Environment Components

Be aware that for products that contain dynamically linked libraries, such as MPI, switching the MPI module environment does not completely change the run time environment because the dynamic libraries are located in the cache used by the run time linker, as specified by `/etc/ld.so.conf`. To use a non-default version of a dynamic library at run time the user should prepend `CRAY_LD_LIBRARY_PATH` to `LD_LIBRARY_PATH`. For more detail, see Modify Linking Behavior to Use Non-default Libraries on page 52.

The following commands revert the environment to an earlier version of 6.2 `cray-mpich`:

```
module swap cray-mpich/6.2.5 cray-mpich/6.2.0 module unload
LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}
```

If the module switch has reverted to an **older major version** of MPI (6.X->5.X), there may be other dependent libraries which need to be switched also. Refer to the release notes to find the compatible libsci and other dependent libraries. As shown below, you will also need to run `craype-pkgconfig` to reset environment variables used by the PE drivers (`cc`, `CC`, `ftn`).

```
# 6.X -> 5.X mpi
module swap cray-mpich/6.2.0.2 cray-mpich2/5.6.4 module unload
#  Reset environment variables needed by the cray pe drivers (cc, CC, ftn)
source craype-pkgconfig disable export
LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}
```

Also see Modify Linking Behavior to Use Non-default Libraries on page 52.

2.6 Using Targeting Modules

The targeting modules deserve special mention. To see which targeting modules are available on your system, use the `module avail -X` command. It returns a list like this, which shows the CPU, network-type, and accelerator modules currently available.

```
---------------------------- /opt/cray/craype/default/modulefiles ---------------
craype-abudhabi            craype-hugepages512M       craype-network-aries
craype-abudhabi-cu         craype-hugepages64M        craype-network-gemini
craype-accel-nvidia20      craype-hugepages8M         craype-sandybridge
craype-accel-nvidia35      craype-interlagos          craype-shanghai
craype-barcelona           craype-interlagos-cu       craype-target-compute_node
craype-hugepages128M       craype-istanbul            craype-target-local_host
craype-hugepages16M        craype-ivybridge           craype-target-native
craype-hugepages256M       craype-mc12                craype-target-petest
craype-hugepages2M         craype-mc8                 craype-xeon
```

2.6.1 Targeting for a Cray System

If you are working on a Cray system, your default environment should load the CPU-, network-, and accelerator-type modules that are correspond to your run time CPU, network, and accelerator platform. For example, if you have a Cray XC30 system with Sandy Bridge compute nodes, your default environment should include the `craype-network-aries` and `craype-sandybridge` modules.

To change the default CPU target, the system administrator must configure `/etc/*rc.local` to load the appropriate `craype-*` target module. On systems that have heterogeneous CPU types available, the user may wish to unload/load appropriate targeting modules. Otherwise, the user need not modify the default targeting environment.

If there are no default targeting modules loaded in the user's environment, the compiler driver scripts (`cc`, `CC`, `ftn`) set the CPU target to `x86`.

If you are working on a standalone Linux workstation or CDL node and developing executable code that will then be moved to and run on a Cray system, always make certain that your local development environment contains the correct targeting modules for the Cray system on which you plan to run your code. For example, code compiled with the wrong CPU module loaded, or with the wrong network module loaded, will not run correctly on the host system. For more information see About Cross-compilers on page 49.

Note: Alternatively, if your site has a heterogeneous system with more than one type of compute node (for example, a Cray XE6 system with both Magny-Cours and Interlagos compute nodes), load the targeting module for the type of compute node on which you intend to execute your code, and then make certain your job is placed only on the specified type of compute node. For more information about job placement, see *Workload Management and Application Placement for the Cray Linux Environment*.

2.6.1.1 Compiling Without the Cray Networking Libraries

If you are compiling an application to run on the Cray compute nodes, but do not wish to use any of the networking libraries such as MPI, or the PGAS languages, load `craype-network-none` instead of the other `craype-network-*` modules. Applications compiled without networking libraries can be run without `aprun`.

2.6.2 Targeting for a Standalone Linux workstation, CDL, or Service Node

If you are working on a standalone Linux workstation or CDL node and compiling code that will be run on a standalone Linux workstation, CDL, or Service Node, load the `craype-network-none` module instead of either of the other network modules, `craype-network-gemini` or `craype-network-aries`.

`craype-network-none` causes no network libraries, to be loaded and network library dependencies are ignored.

2.6.3 Targeting for an Accelerator

Use the accelerator targeting modules to compile an application that uses CUDA directly, or one of the APIs which enable the use of the accelerator, such as OpenACC (supported by CCE). Either load the `craype-accel-nvidia20` module to generate code for Fermi, equivalent to compute capability 2.0, or the `craype-accel-nvidia35` module to generate code for Kepler or Atlas, equivalent to compute capability 3.5. More information about compute capability levels for CUDA-enabled devices is available from NVIDIA. See https://developer.nvidia.com/cuda-gpus.

Load `craype-accel-nvidia*` **only** if you are developing code that will be executed on GPU nodes. Loading the accelerator module enables dynamic linking by default and loads the `libsci_acc` module, which causes increased overhead if the resulting code is executed on non-GPU nodes.

> **Note:** The user should be aware that they will need to ensure that buffers are properly synchronized to the GPU device before a transfer from a device buffer is initiated. See http://docs.nvidia.com/cuda/cuda-driver-api/index.html#r_main.

2.6.4 Targeting for Intel Xeon Phi

Cray XC30 systems equipped with first generation Intel Xeon Phi coprocessors codenamed Knights Corner have special requirements, and applications that use the Xeon Phi coprocessors can run in one of two modes on Cray XC30 systems: *offload* mode and *autonomous* mode.

Symmetric mode—that is, using the Xeon and KNC on the same node to run different programs—is not supported on Cray XC30 systems.

2.6.4.1 Offload Mode

In offload mode, the main part of the code runs on the X86 (host part of the node) while sections of the code may be "offloaded" to the KNC by the use of special Intel compiler directives. This mode is similar to the accelerator mode used for GPUs, although offload mode does not use OpenACC directives.

To use offload mode, load the `PrgEnv-intel` module, and configure your environment as shown below to access the Intel compiler directives.

```
> module load PrgEnv-intel
> source ${INTEL_PATH}/bin/compilervars.sh intel64
  (or for CSH) source ${INTEL_PATH}/bin/compilervars.csh intel64
```

Then compile and run the code as usual. For example:

```
> cc mycode.c
> aprun -n2 -d4 ./a.out
```

Note: Do **not** load the `craype-intel-knc` module, as this will cause the entire application to be targeted to the KNC. Also, note that in offload mode, dynamic linking is not enabled by default.

2.6.4.2 Autonomous Mode

In autonomous mode, the X86 does not execute any parts of the application; the entire application runs on the KNC. In order to use this mode, the user must have their environment set up for autonomous mode at build time.

To do so, you first must load the `PrgEnv-intel` module, then unload any PE products that might already be loaded and in conflict with KNC, and then load the KNC module. For example:

```
> module swap PrgEnv-cray PrgEnv-intel
> module unload cray-libsci atp craype-sandybridge craype-ivybridge
> module load craype-intel-knc
```

At runtime, simply add the `-k` option to `aprun`: for example,

```
> aprun -k -d4 ./a.out
```

Note: In autonomous mode, dynamic linking is enabled by default. Codes that use OpenMP must be linked dynamically because Intel supports only a dynamic version of the OpenMP library.

2.6.4.3 Known Limitations

Use of Intel Xeon Phi coprocessors is subject to these limitations.

- Developers must use the Intel Composer compiler suite. Other compilers do not support KNC at this time.
- Developers must use the Intel Math Kernel Library (MKL). Cray Scientific and Math Libraries (CSML) are not supported on KNC at this time.
- Cray Performance Measurement and Analysis Tools (CPMAT, a.k.a., "CrayPat") release 6.2 or later is supported on KNC, but subject to limitations as described in *Using Cray Performance Measurement and Analysis Tools*. CrayPat-lite is not supported on KNC at this time. Hardware performance counters (PAPI included) are not supported on KNC at this time. Reveal, being dependent on CCE (Cray Compiling Environment), is not supported on KNC at this time.
- Cray Debugging Support Tools (CDST) are not supported on KNC at this time.

2.7 Module Help

Most modules on the Cray system include *module help* that is specific to the module. The exact content of the module help varies from vendor to vendor and release to release, but generally includes release notes and late-breaking news, such as lists of bugs fixed in the release, known dependencies and limitations, and product usage information.

You can view the module help at any time for any module currently installed on the system. The module does not need to be loaded in order for you to view the module help.

To access the module help, use the `module help` command. For example, to see the module help associated with the default CCE module, enter this command:

```
users/yourname> module help cce
```

Note: Make certain you specify the exact module name (and if not the default, the module version) that you want. For example, `module help PrgEnv-cray` and `module help cce` display different information.

2.8 For More Information

The Modules subcommands are documented in the `module`(1) and `modulefiles`(4) man pages. A summary of Modules subcommands can be displayed by entering the `module help` command.

```
users/yourname> module help

  Modules Release 3.2.6.6 2007-02-14 (Copyright GNU GPL v2 1991):

  Usage: module [ switches ] [ subcommand ] [subcommand-args ]

Switches:
        -H|--help               this usage info
        -V|--version            modules version & configuration options
        -f|--force              force active dependency resolution
        -t|--terse              terse    format avail and list format
        -l|--long               long     format avail and list format
        -h|--human              readable format avail and list format
        -v|--verbose            enable  verbose messages
        -s|--silent             disable verbose messages
        -c|--create             create caches for avail and apropos
        -i|--icase              case insensitive
        -u|--userlvl <lvl>      set user level to (nov[ice],exp[ert],adv[anced])
  Available SubCommands and Args:
        + add|load              modulefile [modulefile ...]
        + rm|unload             modulefile [modulefile ...]
        + switch|swap           [modulefile1] modulefile2
        + display|show          modulefile [modulefile ...]
        + avail                 [modulefile [modulefile ...]]
        + use [-a|--append]     dir [dir ...]
        + unuse                 dir [dir ...]
        + update
        + refresh
        + purge
        + list
        + clear
        + help                  [modulefile [modulefile ...]]
        + whatis                [modulefile [modulefile ...]]
        + apropos|keyword       string
        + initadd               modulefile [modulefile ...]
        + initprepend           modulefile [modulefile ...]
        + initrm                modulefile [modulefile ...]
        + initswitch            modulefile1 modulefile2
        + initlist
        + initclear
```

Different versions of the Modules software are in use at different sites. Accordingly, the `module` command arguments and options available on your site may vary from those shown here.

Batch Systems and Program Execution [3]

At most sites, access to the compute node resources is managed by a batch control system, typically PBS Pro, Moab HPC Suite, TORQUE Resource Manager, or Platform LSF. Users run jobs either by using the `qsub` command to submit a job script (or the equivalent command for their batch control system), or else by using the `qsub` command (or its equivalent) to request an interactive session within the context of the batch system, and then using the `aprun` command to run the application within the interactive session.

User applications are always launched on compute nodes using the application launcher, `aprun`, which submits applications to the Application Level Placement Scheduler (ALPS) for placement and execution. The ALPS service is both very powerful and highly flexible, and a thorough discussion of it is beyond the scope of this manual. For more detailed information about ALPS and `aprun`, see the `intro_alps`(1) and `aprun`(1) man pages, and *Workload Management and Application Placement for the Cray Linux Environment*.

Running an application typically involves these steps.

1. Determine what system resources you will need. Generally, this means deciding how many cores and/or compute nodes you need for your job.
2. Use the `apstat` command to determine whether the resources you need are available. This is very important when you are planning to run in an interactive session. This is not as important if you are submitting a job script, as the batch system will keep your job script in the queue until the resources become available to run it.
3. Translate your resource request into the appropriate batch system and `aprun` command options, which are not necessarily the same. If running a batch job, modify your script accordingly.
4. For batch job submission, use the batch command (e.g., `qsub`) to submit your job script which contains launches your job.

5. For interactive job submission, there are two ways to reserve the needed resources and launch an application:

   - First, use the appropriate batch command with interactive option (e.g., `qsub -I`) to explicitly reserve resources. Then, enter the `aprun` command to launch your application.
   - Omit explicit reservation through `qsub -I`. Using `aprun` assumes interactive session and resources are implicitly reserved based on `aprun` options.

3.1 Interactive Mode

Interactive mode is typically used for debugging or optimizing code, but not for running production code. For example, to begin an interactive session on a system using PBS Pro, use the `qsub -I` command.

```
users/yourname> qsub -I
```

Useful `qsub` options include:

`-I` Start an interactive session.

`-A` *account* Charge the time to *account*.

`-q debug` Run in the debug queue.

`-V` Import any environment variables that were set in the user's shell.

`-l` *resource_list*

Allows user to request resources and specify job placement.

The `-l` *resource_list* argument supports a large number of options that are described in the `qsub`(1B) man page and expanded upon in the `pbs_resources`(7B) man page, and described in greater detail with examples in *Workload Management and Application Placement for the Cray Linux Environment*.

After you have launched an interactive session, use the `aprun` command to launch your application.

When you are finished, enter **logout** to exit the batch system and return to the command line.

3.1.1 Notes

- Pay attention to your file system mount points. You must be on the high-performance parallel file system (for example, if you are using the Lustre file system, /lus/nid*number*/*yourname*) in order to launch jobs on the compute nodes. However, when you launch an interactive batch session, you are by default placed in your home directory, typically on a login node. For example, /ufs/home/users/*yourname*. You may need to cd back to the parallel file system after launching an interactive session.
- After you launch an interactive batch session, a number of environment variables are automatically defined and exported to the job, as described on the qsub(1B) man page. For example, the environment variable $PBS_O_WORKDIR is set to the directory from which the batch job was submitted. This can be a handy way to return to your mount point if you cd to the parallel file system before invoking the interactive batch session.
- The qsub and aprun commands use different options to perform similar functions. These differences are touched on lightly in Using aprun on page 39 and described in detail in *Workload Management and Application Placement for the Cray Linux Environment*.
- When you launch an interactive batch session, you must request the maximum number of resources you expect to use. Once a batch session begins, you can only use **fewer** resources than initially requested. You cannot use the aprun command to use more resources than you reserved using the qsub command.

3.2 Batch Mode

Production jobs are typically run in batch mode. Batch scripts are shell scripts containing flags and commands to be interpreted by a shell and are used to run a set of commands in sequence.

To use PBS, load the pbs module:

```
users/yourname> module load pbs
```

To use Torque/MOAB, load the moab module:

```
users/yourname> module load moab
```

For example, to run a batch script using PBS Pro, use the qsub command.

```
users/yourname> qsub [-l resource_list] jobscript
```

The [-l *resource_list*] arguments are described in the pbs_resources(7B) man page.

A typical PBS Pro 11.0 batch script might look like this:

```
1: #!/bin/sh
2: #PBS -A account
3: #PBS -N job_name
4: #PBS -j oe
5: #PBS -l walltime=1:00:00,mppwidth=192
6:
7: cd $PBS_O_WORKDIR
8: date
9: aprun -n 192 ./a.out > my_output_file 2>&1
```

Parsing this script line-by-line, it would be interpreted as follows:

1. Invoke the shell to use to interpret the script.
2. Specify the account to which this time is billed.
3. Assign the job a name to use on messages and output.
4. Join the job's STDOUT and STDERR into STDOUT.

   Note: While your job is running, STDOUT and STDERR are written to a file or files in a system directory and the output is copied to your submission directory only after the job completes. Specifying the `-j oe` option here and redirecting the output to a file in line 9 makes it possible for you to view STDOUT and STDERR while the job is running. For more information about the `-j` option, see the qsub(1B) man page.
5. Reserve 192 processing elements for one hour.
6. This line is blank and is ignored.
7. `cd` to the submission directory, which presumably is a mount point on the Lustre file system.
8. Run the `date` command.
9. Execute the executable file `a.out` on 192 processing elements and redirect any output to *my_output_file*.

After the job is submitted using the `qsub` command, it goes into the queue, where it waits until the requested resources become available. When they do, the job is launched on the head node of the allocated resources, and it runs until either it reaches its planned completion or until the wall clock time (if specified) is up.

While the job is in the queue, a number of optional commands are available.

`qstat` Show the status of the job queue. This command is available at any time, whether or not you have a job in the queue.

`qdel` *job_id*

Delete job *job_id* regardless of its current state and remove it from the queue.

`qhold` *job_id*

Place a non-running job on hold. The job remains in the queue but will not execute. This command cannot be used once the job begins running.

`qrls` *job_id*

Release a job that is on hold.

`qalter` *job_id*

Alter the characteristics—name, account, number of requested cores, and so on—of a job in the queue. This command cannot be used once the job begins running.

`showq` (Moab only) Similar to `qstat` but providing more detail.

`checkjob` *job_id*

(Moab only) Check the status of a job currently in the queue.

`showstart` *job_id*

(Moab only) Show the estimated start time for a job in the queue.

`showbf` (Moab only) Show the current backfill. This can help you to build small jobs that can be backfilled immediately while you are waiting for the resources to become available for your larger jobs.

For more information about batch scripts, see your batch system's user documentation.

3.3 Using `aprun`

The `aprun` utility launches applications on compute nodes. The utility submits applications to the Application Level Placement Scheduler (ALPS) for placement and execution, forwards the login node environment to the assigned compute nodes, forwards signals, and manages the `stdin`, `stdout`, and `stderr` streams.

Verify that you are in a directory mounted on the high-speed parallel file system before using the `aprun` command.

In simplest form, the aprun command looks like this:

```
/lus/nid00008> aprun -n x ./program_name
```

The aprun command supports a large number of options that provide you with a high degree of control over just exactly how your job is placed and executed on the compute nodes. At a minimum, you must use the -n option to specify the number of cores on which to run the job.

Note: Remember, you use aprun within the context of a batch session and the maximum size of the job is determined by the resources you requested when you launched the batch session. You cannot use the aprun command to use more resources than you reserved using the qsub command.

The aprun and qsub commands support comparable but differently named options. This table lists some of the more commonly used aprun options and their qsub (PBS Pro 11.0) equivalents.

Table 2. aprun Versus qsub Options

aprun **Option**	qsub -l **Option**	**Description**
-n 4	-l mppwidth=4	Width (number of PEs)
-d 2	-l mppdepth=2	Depth (number of CPUs hosting OpenMP threads)
-N 1	-l mppnppn=1	Number of PEs per node
-L 5,6,7	-l mppnodes=\"5,6,7\"	Candidate node List
-m 1000m	-l mppmem=1000mb	Memory per PE

Note: The -B option forces aprun to inherit the values associated with the -n, -d, -N, and -m options from the batch session. The aprun command exits with an error if you specify any of these options and the -B option at the same time.

A full discussion of aprun options is beyond the scope of this manual. For more information see the aprun(1) man page, and for detailed explanations and examples, see *Workload Management and Application Placement for the Cray Linux Environment*. Also, note that the behavior of aprun and the aprun command options supported may vary depending on which version of the CLE operating system is installed on your system.

3.3.1 Special Considerations for Intel Xeon Phi

The `aprun` command supports the `-k` argument, which is used to specify that the application should be placed for execution on an Intel Xeon Phi coprocessor. Note that executable programs must be built specially for execution on a Xeon Phi. If you attempt to run a program not built specially, you will see the following message:

```
aprun: Binary not built for Xeon Phi. Cross-compile your application or use -b to run a command.
```

Commands that are already present on the Xeon Phi may be run by adding the `-b` switch to bypass copying the binary to the compute node.

If you attempt Xeon Phi application placement on a node which does not have a Xeon Phi coprocessor, the execution will fail and exit with an error message.

The default `aprun -cc cpu` option will cause all OpenMP threads to bind to a single KNC thread. Using the `-cc none` argument and `KMP_AFFINITY=disabled` will disable this binding, but may negatively impact code performance. If `-cc none` is specified, the `-d` and `-j` arguments are ignored.

At this time Cray recommends that for best performance, Xeon Phi code be executed with `aprun` arguments `-cc depth` and `-k`, and that `KMP_AFFINITY` be set to `balanced`. The `KMP_AFFINITY` values `scatter` or `compact` may yield better performance with some code, and the `aprun -d` and/or `-j` arguments may be used to fine-tune performance. For more information, see the `aprun`(1) man page.

Using Compilers [4]

The Cray system supports a variety of compilers from a variety of vendors and support for new compilers and languages is being added on an ongoing basis. The GNU Fortran, C, and C++ compilers are supplied with all systems, while all other compilers are available as optional and separately licensed add-ons. At present, the following compilers from the following vendors are supported on Cray systems.

- Cray Inc., Cray Compiling Environment (CCE) (Fortran, C, and C++)
- The Portland Group, Parallel Fortran, C, and C++
- Intel Inc., Intel Composer (Fortran and C++)
- Chapel Parallel Programming Language

The compilers available on your system depend on which products your site administration has chosen to license and install.

Note: At this time, in order to use the first generation Intel Xeon Phi coprocessors codenamed Knights Corner, the Intel Composer compiler suite must be used.

4.1 About Compiler Drivers

Because of the multiplicity of possible compilers, Cray supplies compiler drivers, wrapper scripts, and disambiguation 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.

Note that while you always use one of the above commands (either on the command line or in your make files) to invoke the compiler, the arguments used with the commands vary according to which compiler module is loaded. For example, the arguments and options supported by the PGI Fortran compiler are different from those supported by the Cray Fortran (CCE) compiler.

Important: Regardless of which compiler module you have loaded, **do not** use the native compiler commands. For example, if you are using the PGI compiler suite, do not use the `pgf95` command to invoke the Fortran compiler. If you do so, your code may appear to compile and link successfully, but it will be linked to the wrong libraries and the resulting program can be executed on login nodes only; it cannot be executed on compute nodes.

4.1.1 Bypassing the Compiler Drivers

In special cases you may want to bypass the compiler drivers and use the native compiler commands. Do not do so. Instead, load the special targeting module, `craype-target-native`, and then continue to use the `ftn`, `cc`, and `CC` commands as before. The `craype-target-native` module enables the compiler driver commands to function as native compiler commands—for example, if the PGI programming environment is loaded, the `ftn` command works as if it is `pgf95`, but eliminates all default library links while also preventing any linking to incorrect libraries.

To restore normal compiler driver behavior, unload the `craype-target-native` module.

4.2 About C/C++ Data Types

The C/C++ compilers differ on the size of the *long double* data type. The PGI and CCE compilers define long double as being 8 bytes. All other compilers define the long double as being 16 bytes.

Table 3. C/C++ Data Type Sizes

Data Type	Size in Bytes
unsigned char	1
signed char	1
unsigned short	2
signed short	2
unsigned int	4
signed int	4
unsigned long	8

Data Type	Size in Bytes
signed long	8
unsigned long long	8
signed long long	8
float	4
_float128	16
_float128 complex	32
double	8
long double	8 or 16, depending on compiler
char *	8
enum	4

4.3 About the Cray Compiling Environment (CCE)

Table 4. Cray Compiler Basics

Module:	`PrgEnv-cray`
Command:	`ftn`, `cc`, `CC`
Compiler-specific man pages:	`crayftn`(1), `craycc`(1), `crayCC`(1) **Note:** Compiler-specific man pages are available only when the compiler module is loaded.
Online help:	None provided
Documentation:	*Cray Fortran Reference Manual*, *Cray C and C++ Reference Manual*

4.3.1 Known Limitations

- If you use the Cray Fortran compiler with the PETSc (Portable, Extensible Toolkit for Scientific Computation) library, either add the directive `!dir$ PREPROCESS EXPAND_MACROS` to the source code or add the `-F` option to the `ftn` command line.
- At this time, CCE does not support the first generation Intel Xeon Phi coprocessors codenamed Knights Corner.

4.4 About PGI Compilers

Table 5. PGI Compiler Basics

Module:	`PrgEnv-pgi`
Command:	`ftn`, `cc`, `CC`
Compiler-specific man pages:	`pgf95`(1), `pgcc`(1), `pgCC`(1) **Note:** Compiler-specific man pages are available only when the compiler module is loaded.
Online help:	`pgf95 -help`, `pgcc -help`, `pgCC -help`,
Documentation:	`/opt/pgi/`*version*`/linux86-64/`*version*`/doc`

4.4.1 Known Limitations

- At this time, PGI compilers do not support the first generation Intel Xeon Phi coprocessors codenamed Knights Corner.
- The PGI compilers are not able to handle template-based libraries such as Tpetra.
- When linking in ACML routines, you must compile and link all program units with `-Mcache_align` or an aggregate option that incorporates `-Mcache_align` such as `fastsse`.
- The `-Mconcur` (auto-concurrentization of loops) option is not supported on Cray systems.
- The `-mprof=mpi`, `-Mmpi`, and `-Mscalapack` options are not supported.
- The PGI debugger, `PGDBG`, is not supported on Cray systems.
- The PGI profiling tools, `pgprof` and `pgcollect`, are not supported on Cray systems.
- The PGI Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.

4.5 About Intel Compilers

Table 6. Intel Composer Basics

Module:	`PrgEnv-intel`
Command:	`ftn`, `cc`, `CC`
Compiler-specific man pages:	`ifort`(1), `fpp`(1), `icc`(1), `icpc`(1) **Note:** Compiler-specific man pages are available only when the compiler module is loaded.
Online help:	`ifort --help`, `icc --help`
Documentation:	`/opt/intel/Compiler/`*version*`/Documentation/`*language*

4.5.1 Known Limitations

- The Intel Compiler Suite (Intel Composer) must be installed in the default location. The optional Intel C/C++ only installation is not supported because the Intel Fortran run time libraries are required by Cray libraries such as `libsci` when using the Intel compiler.
- The Intel Composer does not support the UPC or Coarray Fortran parallel programming models.
- The Intel Composer is the only compiler suite that supports first generation Intel Xeon Phi coprocessors codenamed Knights Corner.

4.6 About GNU Compilers

Table 7. GNU Compiler Basics

Module:	`PrgEnv-gnu`
Command:	`ftn`, `cc`, `CC`
Compiler-specific man pages:	`gfortran`(1), `gcc`(1), `g++`(1) **Note:** Compiler-specific man pages are available only when the compiler module is loaded.
Online help:	`gfortran --help`, `gcc --help`, `g++ --help`

4.6.1 Known Limitations

- The GNU compilers do not support the UPC or Coarray Fortran parallel programming models.
- At this time, the GNU compilers do not support the first generation Intel Xeon Phi coprocessors codenamed Knights Corner.

4.7 About the Chapel Parallel Programming Language

Chapel is an emerging parallel programming language whose design and development is being led by Cray Inc. Chapel is being developed as an open-source effort with contributions from academia, industry, and scientific computing centers. Chapel emerged from Cray's entry in the DARPA-led High Productivity Computing Systems program (HPCS).

Chapel is designed to improve the productivity of high-end computer users while also serving as a portable parallel programming model that can be used on commodity clusters or desktop multicore systems. Chapel strives to vastly improve the programmability of large-scale parallel computers while matching or beating the performance and portability of current programming models like MPI.

Chapel supports a multithreaded execution model via high-level abstractions for data parallelism, task parallelism, concurrency, and nested parallelism. Chapel's *locale* type enables users to specify and reason about the placement of data and tasks on a target architecture in order to tune for locality. Chapel supports *global-view* data aggregates with user-defined implementations, permitting operations on distributed data structures to be expressed in a natural manner. In contrast to many previous higher-level parallel languages, Chapel is designed around a *multiresolution* philosophy, permitting users to initially write very abstract code and then incrementally add more detail until they are as close to the machine as their needs require. Chapel supports code reuse and rapid prototyping via object-oriented design, type inference, and features for generic programming.

Chapel was designed from first principles rather than by extending an existing language. It is an imperative block-structured language, designed to be easy to learn for users of C, C++, Fortran, Java, Perl, Matlab, and other popular languages. While Chapel builds on concepts and syntax from many previous languages, its parallel features are most directly influenced by ZPL, High-Performance Fortran (HPF), and the Cray MTA/Cray XMT extensions to C and Fortran.

For more information about Chapel, see: http://chapel.cray.com.

4.8 About Cross-compilers

The Cray system supports using standalone Linux workstations as code development platforms for CLE 4.X systems. Install the Cray Application Developer's Environment (CADE), release 6.17 or later, on Cray XE or Cray XK systems running CLE release 4.0, 4.1, or 4.2.

The Cray system supports using Cray Development and Login (CDL) hosts as development platforms for CLE 5.X systems. Install the Cray Developer Toolkit (CDT), release 1.16 or later, on Cray XC30 systems running Cray Linux Environment (CLE) release 5.1 or later.

When the Cray Application Developer's Environment (CADE) is installed on a suitable Linux or CDL system, programs can be written and compiled on that system and then exported to the Cray system for subsequent execution, debugging, and optimization.

For instructions on installing CADE and CDT, please see *Cray Programming Environments Installation Guide*. After CADE or CDT is installed and your compilers are installed and configured on your Linux system, follow these steps to begin developing code.

Procedure 1. Setting Up the Programming Environment

1. Load the `cray-network` module corresponding to the Cray system for which you intend to develop code. If you are developing for a Gemini system (Cray XE5, Cray XE6, or Cray XK6), load the `cray-network-gemini` module. If you are developing for a XC30 system, load the `cray-network-aries` module.

   Note: On the actual Cray system, the network type is typically set by default and transparent to the user. When working on a standalone Linux system, you **must** set this manually. If the correct `cray-network` module is not selected, your code may appear to compile successfully on the Linux system but will not run on the Cray system.

2. Load the `cray-`*processor* module corresponding to the compute nodes on the Cray system for which you intend to develop code. Your choices are:

   - `craype-abudhabi`
   - `craype-abudhabi-cu`
   - `craype-mc8`
   - `craype-mc12`
   - `craype-interlagos`

- craype-interlagos-cu
- craype-sandybridge
- craype-ivybridge
- craype-intel-knc

Note: On the actual Cray system, the processor type is typically set by default and transparent to the user. When working on a standalone Linux system, you **must** set this manually. If you select a processor type with fewer cores than are actually present on the Cray compute nodes, your code will not make full use of the Cray system resources and may either run slowly or cause conflicts with aprun options and placement. If you select a processor type with more cores than are actually present on the Cray compute nodes, your application may appear to compile successfully but will not run.

3. **(Cray XK, XC30 system users only and CCE 8.0 or later only)** Load the accelerator module that is appropriate for the compute nodes on the Cray system for which you intend to develop code. Currently, the only supported options are craype-accel-nvidia20, craype-accel-nvidia35 and the only compiler that supports the accelerator module is CCE.

   The accelerator target is not set by default on either the standalone Linux system or the Cray system. You must set this manually. This module sets compiler options required to compile applications for the accelerator target.

   Load the accelerator module **only** if you are developing code that will be executed on GPU nodes. Loading the accelerator module enables dynamic linking and loads the libsci_acc module, which causes increased overhead if the resulting code is executed on non-GPU nodes.

   The first generation Intel Xeon Phi coprocessors codenamed Knights Corner are not accelerators, and therefore do not require loading accelerator support modules.

4. Load the PrgEnv-*vendor* module containing your compiler of choice.

You are now ready to begin writing, editing, and compiling code. Remember, however, that you must move your code to mount point on the Cray system (for example, /lus/nid00008) before you can run, debug, or optimize it.

Dynamic Linking [5]

Dynamic linking, or run time linking, potentially reduces memory allocated for a program on a compute node. Dynamically linked applications contain references to dynamic libraries, also known as *dynamic shared objects*. It allows the user to benefit from library upgrades, without having to recompile.

The module environment, in combination with the Cray compiler drivers (`CC`, `cc`, `ftn`), define the default link-type, cpu-target, network-target, accelerator-target, compiler and other information required to create an executable.

5.1 Implementation

There have been changes to the implementation of dynamic linking within the drivers and PE product installation, beginning with the 5.0 release of CLE, which includes a new `craype` package (formerly named `xt-asyncpe`). To the user, changes to the implementation of dynamic linking should be transparent but it is helpful to be aware of default behavior.

If a user wishes to modify the default run time programming environment for a dynamically linked application, they will need to modify the default search path for dynamically linked libraries.

5.2 Linking Defaults

By default, the compiler driver scripts, `CC`, `cc`, `ftn`, set the `-static` option. If one of the GPU targeting modules is loaded (`craype-accel-nvidia20`, `craype-accel-nvidia35`), or if the Intel Xeon Phi module is loaded (`craype-intel-knc`), the default linking behavior changes to dynamic because the required libraries are dynamic. To modify the linking behavior, use the `-dynamic` or `-static` option on the compiler driver script. Alternatively, a user may wish to set the shell environment variable `CRAYPE_LINK_TYPE` to `dynamic` or `static` to change the default link type.

Generally, when a new, non-compiler Cray product containing dynamic libraries is set as the default by the administrator, the installation process adds run time links to the `/opt/cray/lib64` directory and executes `/sbin/ldconfig` to add `/opt/cray/lib64` to the `ld` cache making these dynamic libraries accessible to the run time environment.

Runtime linking is against the list of links in /opt/cray/lib64. This set of run time links reflects the set of products installed as "default" by the system administrator. The non-compiler Cray product installation configures the run time linker's cache, not RPATH, nor LD_LIBRARY_PATH to define the run time link path for dynamically linked applications.

The compilers (pgi, cce, gcc, and intel) as well stat may control the run time link path and set the RPATH stored in the executable. For example, the CCE compiler script, craycc, passes the -Wl,-rpath= option to the compiler driver to specify the associated set of Craylibs to use during run time.

When a dynamically linked executable runs in the Cray environment, it finds dynamically linked libraries according to the following search order:

- LD_LIBRARY_PATH environment variable
- RPATH embedded in the header of the executable.
- Directories in /etc/ld.so.cache, the dynamic linker's cache, created by the ldconfig command which is run during the installation process. Cray programming environment products' dynamic libraries are installed in /opt/cray/lib64.

When there are entries in LD_LIBRARY_PATH, and RPATH those directory paths are searched first for each library that is referenced by the run time application, affecting the run time for applications, particularly at higher node counts. For this reason, the default programming environment does not use LD_LIBRARY_PATH.

5.3 Modify Linking Behavior to Use Non-default Libraries

The environment variable CRAY_LD_LIBRARY_PATH, is set to a colon-separated list of every product library path in the current environment, and is automatically generated by the module environment. When modules are unloaded and loaded, its value changes accordingly. It is **not** recognized by the loader, nor consumed by the programming environment, but it is provided to facilitate the modification of the user's LD_LIBRARY_PATH, if needed.

If a non-default product is loaded or if a user suspects an issue with the default version of a library, the user may wish to prepend the CRAY_LD_LIBRARY_PATH to LD_LIBRARY_PATH. For example, a user who wishes to run an application against a product set that contains some non-default product, can prepend CRAY_LD_LIBRARY_PATH to LD_LIBRARY_PATH as follows:

```
module load PrgEnv-cray
module swap cray-libsci cray-libsci/version
module swap cray-mpich cray-mpich/version
setenv LD_LIBRARY_PATH $CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
aprun ./a.out
```

The use of `LD_LIBRARY_PATH` does have a performance impact and should be reserved for special cases. Programs that needed to be locked to a specific version of a Programming Environment library should use the link option `-Wl,-rpath=`, or static linking, if possible.

Libraries [6]

Cray provides a large variety of libraries to support application development and interprocess communications on Cray systems. New libraries are being ported to the Cray system on an ongoing basis.

The following libraries can be used with all compilers currently supported on the Cray system, except where noted in Chapter 4, Using Compilers on page 43.

6.1 Cray Scientific and Math Libraries (CSML)

The Cray Scientific and Math Libraries (CSML, also known as *LibSci*) are a collection of numerical routines optimized for best performance on Cray systems. All programming environment modules load `cray-libsci` by default, except where noted. When possible, you should use calls to the CSML routines in your code in place of calls to public-domain or user-written versions.

Note: CSML is not supported for first generation Intel Xeon Phi coprocessors codenamed Knights Corner. When building a program that is to use the Intel Xeon Phi coprocessor, unload the `cray-libsci` module before loading the `craype-intel-knc` module. Developers writing code to run on Intel Xeon Phi coprocessors must use the Intel Math Kernel Library (MKL) instead of CSML.

6.1.1 Basic CSML Components

Table 8. CSML Basics

Module:	`cray-libsci`
Man pages:	`intro_libsci`(3s), `intro_libsci_acc`(3s), `intro_blas1`(3s), `intro_blas2`(3s), `intro_blas3`(3s), `intro_blacs`(3s), `intro_lapack`(3s), `intro_lapacke`(3s), `intro_scalapack`(3s), `intro_irt`(3), `intro_fft`(3s), `intro_fftw2`(3), `intro_fftw3`(3) **Note:** Library-specific man pages are available only when the associated module is loaded.

The CSML collection contains the following Scientific Libraries.

- BLAS (Basic Linear Algebra Subroutines)
- BLACS (Basic Linear Algebra Communication Subprograms)
- LAPACK (Linear Algebra Routines)
- LAPACKE (C interfaces to LAPACK Routines)
- ScaLAPACK (Scalable LAPACK)
- FFT (Fast Fourier Transform Routines)
- FFTW2 (the Fastest Fourier Transforms in the West, release 2)
- FFTW3 (the Fastest Fourier Transforms in the West, release 3)

In addition, the Cray LibSci collection contains three libraries developed by Cray.

- IRT (Iterative Refinement Toolkit)
- LibSci_ACC (Accelerated BLAS and LAPACK routines, optimized for use on systems with GPU accelerators)

6.1.2 BLAS and LAPACK

The BLAS and LAPACK libraries are loaded by default as part of the `cray-libsci` module. The BLAS (Basic Linear Algebra Subroutines) library contains three levels of optimized subroutines. Level 1 BLAS perform the following types of basic vector-vector operations:

- Dot products and various vector norms
- Scaling, copying, swapping, and computing linear combination of vector
- Generate or apply plane or modified plane rotations

For more information, see the `intro_blas1`(3s) man page.

Level 2 BLAS perform matrix-vector operations, and generally produce improved code performance when inlined. For more information about Level 2 BLAS, see the `intro_blas2`(3s) man page.

Level 3 BLAS perform matrix-matrix operations. For more information about Level 3 BLAS, see the `intro_blas3`(3s) man page.

LAPACK is a public domain library of subroutines for solving dense linear algebra problems, including the following:

- Systems of linear equations
- Linear least squares problems
- Eigenvalue problems
- Singular value decomposition (SVD) problems

LAPACK is the successor to the older LINPACK and EISPACK packages. It extends the functionality of these packages by including equilibration, iterative refinement, error bounds, and driver routines for linear systems, routines for computing and reordering the Schur factorization, and condition estimation routines for eigenvalue problems. Performance issues are addressed by implementing the most computationally-intensive algorithms using Level 2 and Level 3 BLAS.

For more information about LAPACK, see the `intro_lapack`(3s) man page.

6.1.2.1 Notes

- BLAS library behavior is dependent on the `craype-`*processor* module. At most sites this module is typically loaded by default and transparent to the user. However, if your site has multiple types of compute nodes, or if you are working in an unmanaged CDL or Linux cross-compiling environment, it may be necessary to load the `craype-`*processor* module corresponding to the compute nodes on the Cray system for which you intend to develop code in order to obtain best performance. Your choices are:

  - `craype-barcelona` (AMD quad core, Cray XT4)
  - `craype-shanghai` (AMD quad core, Cray XT5)
  - `craype-istanbul` (AMD six cores)
  - `craype-mc8` (AMD eight cores)
  - `craype-mc12` (AMD twelve cores)
  - `craype-interlagos` (AMD sixteen cores)
  - `craype-interlagos-cu` (AMD sixteen cores, optimized for Compute Unit Affinity)
  - `craype-abudhabi` (AMD sixteen cores)
  - `craype-abudhabi-cu` (AMD sixteen cores, optimized for Compute Unit Affinity)
  - `craype-sandybridge` (Intel eight cores/sixteen threads)
  - `craype-ivybridge` (Intel twelve cores/twenty-four threads)

  Note: If you select a processor type with fewer cores than are actually present on the Cray compute nodes, your code will not make full use of the Cray system resources and may either run slowly or cause conflicts with `aprun` options and placement. If you select a processor type with more cores than are actually present on the Cray compute nodes, your application may appear to compile successfully but will not run.

- If you require a C interface to BLAS and LAPACK but want to use Cray LibSci BLAS or LAPACK routines, use the Fortran interfaces.

- To obtain threading behavior, set `OMP_NUM_THREADS`, as described in BLACS and ScaLAPACK on page 58.

- You can access the Fortran interfaces from a C program by adding an underscore to the respective routine names and passing arguments by reference (rather than by value). For example, you can call the `dgetrf()` function as follows:

  ```
  dgetrf_(&uplo, &m, &n, a, &lda, ipiv, work, &lwork, &info);
  ```

- C programmers using the Fortran interface must order arrays in Fortran column-major order.

- Some older versions of the Cray BLAS and LAPACK libraries optimized to support older AMD processors include routines from the 64-bit `libGoto` library from the University of Texas. Use of `libGoto` library routines in Cray libraries is being phased out.

6.1.3 BLACS and ScaLAPACK

The BLACS (Basic Linear Algebra Communication Subprograms) and ScaLAPACK (Scalable LAPACK) libraries are loaded by default as part of the `cray-libsci` module. BLACS is a package of routines that provide the same functionality for message-passing linear algebra communication as the Basic Linear Algebra Subprograms (BLAS) provide for linear algebra computation. With these two packages, software for dense linear algebra can use calls to BLAS for computation and calls to BLACS for communication. The BLACS consist of communication primitives routines, global reduction routines, and support routines.

For more information about BLACS, see the `intro_blacs`(3s) man page.

The ScaLAPACK library uses BLACS primitives to provide optimized routines for solving real or complex general, triangular, or positive definite distributed systems; for reducing distributed matrices to condensed form and an eigenvalue problem solver for real symmetric distributed matrices; and to perform basic operations involving distributed matrices and vectors.

LU and Cholesky routines in ScaLAPACK have been modified to allow the user to choose an underlying broadcast algorithm during run time. It can be done either via an environment variable, or by calling a helper routine in a program.

For more information about ScaLAPACK, see the `intro_scalapack`(3s) man page.

6.1.3.1 Notes

- Some ScaLAPACK routines require the Basic Linear Algebra Communication Subprograms (BLACS) to be initialized. This can be done through a call to `BLACS_GRIDINIT`. Also, each distributed array that is passed as an argument to a ScaLAPACK routine requires a descriptor, which is set through a call to `DESCINIT`.
- The ScaLAPACK and BLACS libraries can be used in MPI and SHMEM applications. Cray LibSci also supports hybrid MPI/ScaLAPACK applications, which use threaded BLAS on a compute node and MPI between nodes. To use ScaLAPACK in a hybrid application:
  1. Adjust the process grid dimensions in ScaLAPACK to account for the decrease in BLACS nodes.
  2. Ensure that the number of BLACS processes required is equal to the number of nodes required, **not** the number of cores.
  3. Set the `OMP_NUM_THREADS` environment variable.

6.1.4 Iterative Refinement Toolkit (IRT)

The Iterative Refinement Toolkit (IRT) is a library of Fortran subroutines that provides solutions to linear systems using 32-bit factorizations while preserving accuracy through mixed-precision iterative refinement. IRT exploits the fact that single-precision solvers can be up to twice as fast as double-precision solvers, and uses an iterative refinement process to obtain solutions accurate to double-precision. IRT includes both serial and parallel implementations of the LU and Cholesky algorithms, and serial versions of the QR algorithm for real and complex matrices.

IRT includes the following features:

- Sophisticated stopping criteria
- Potential minimization of forward error
- Ability to return error bounds
- Return an estimate of the condition number of matrix A
- Return to the double-precision factorization-and-solve process if IRT cannot obtain a solution

IRT provides two interfaces:

- Benchmarking interface. The benchmarking interface routines replace the high-level drivers of LAPACK and ScaLAPACK. The names of the benchmark API routines are identical to their LAPACK or ScaLAPACK counterparts or replace calls to successive factorization and solver routines. This allows you to use the IRT process without modifying your application.

  For example, the IRT `dgesv()` routine replaces either the LAPACK `dgesv()` routine or the LAPACK `dgetrf()` and `dgetrs()` routines. To use the benchmarking interface, set the `IRT_USE_SOLVERS` environment variable to `1`.

  Note: Use this interface with caution; calls to the LAPACK LU, QR or Cholesky routines are intercepted and the IRT is used instead.

- Expert interface. The expert interface routines give you greater control of the iterative refinement process and provide details about the success or failure of the process. The format of advanced API calls is:

  `call irt_`*factorization-method_data-type_processing-mode*`(`*arguments*`)`

  such as: `call irt_po_real_parallel(`*arguments*`)`.

For more information about IRT, see the `intro_irt`(3) man page.

6.1.5 Fourier Transformations

Fast Fourier transforms are handled by using FFTW. Alternatively, on Cray XT, Cray XE, and Cray XK systems, FFT can be handled using ACML.

The `intro_fft`(3s) man page is a disambiguation page.

6.1.5.1 FFTW

FFTW is a C subroutine library with Fortran interfaces for computing the discrete Fourier transform in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, such as the discrete cosine/sine transforms). The Fast Fourier Transform algorithm is applied for many problem sizes.

Cray LibSci includes both version 2.1.5.*x* and multiple 3.2.*x* and 3.3.*x* versions of the Fastest Fourier Transform in the West (FFTW) library. By default, no version of FFTW is loaded.

The FFTW 3.3.x and FFTW 2.1.5.1 modules cannot be loaded at the same time. If a module is already loaded, you must first unload one module, before loading the desired module. For example, if you have loaded the FFTW 3.3.x library and want to use FFTW 2.1.5.1 instead, use:

```
% module swap fftw/3.3.0.2 fftw/2.1.5.1
```

For more information about FFTW, see the `intro_fftw2`(3) and `intro_fftw3`(3) man pages.

6.1.5.2 ACML

The AMD Core Math Library (ACML) is available for Cray XT, Cray XE, and Cray XK systems equipped with AMD Opteron CPUs only.

The ACML module is not loaded as part of the default Cray LibSci. However, if you need ACML for FFT functions, math functions, or random number generators, you can load the library using the `acml` module:

```
% module load acml
```

Note: If you load the `acml` module manually, you must also use `-l acml` option when compiling and linking to link in the ACML library.

ACML includes:

- A suite of Fast Fourier Transform (FFT) routines for real and complex data
- Fast scalar, vector, and array math transcendental library routines optimized for high performance
- A comprehensive random number generator suite:
  - Base generators plus a user-defined generator
  - Distribution generators
  - Multiple-stream support

ACML's internal timing facility uses the `clock()` function. If you run an application on compute nodes that uses the *plan* feature of FFTs, underlying timings will be done using the native version of `clock()`. On CNL, `clock()` returns the sum of user and system CPU times.

6.1.6 PETSc

Table 9. PETSc Basics

Modules:	`cray-petsc`, `cray-petsc-complex`, `cray-tpsl`
Man pages:	`intro_petsc`(3s) **Note:** Library-specific man pages are available only when the associated module is loaded.
Website:	http://www.mcs.anl.gov/petsc/petsc-as/

PETSc (Portable, Extensible, Toolkit for Scientific Computation) is an open source library of parallel linear and nonlinear equation solvers intended for use in large-scale C, C++, or Fortran applications. PETSc uses standard MPI functions for all message-passing communication.

The PETSc modules are **not** loaded by default. PETSC is dependent on the `cray-libsci` and `craype` modules. Make certain these modules are loaded before using PETSc. When you load the `petsc` module, the Third-Party Scientific Libraries (`cray-tpsl`) module is automatically loaded as well, to provide access to the libraries required to support PETSc.

Note: Always use the `cray-tpsl` module that is linked to the PETSc module. PETSc and Trilinos are asynchronous products and may at times use different versions of the TPSL libraries.

PETSc provides many of the mechanisms needed for parallel applications, such as simple parallel matrix and vector assembly routines that allow the overlap of communication and computation. In addition, PETSc includes support for parallel distributed arrays useful for finite difference methods, such as:

- Parallel vectors, including code for communicating ghost points
- Parallel matrices, including several sparse storage formats
- Scalable parallel preconditioners
- Krylov subspace methods
- Parallel Newton-based nonlinear solvers
- Parallel time-stepping ordinary differential equation (ODE) solvers

The following packages are included in PETSc/TPSL.

- MUMPS (MUltifrontal Massively Parallel sparse direct Solver) is a package of parallel, sparse, direct linear-system solvers based on a multifrontal algorithm. For further information, see http://graal.ens-lyon.fr/MUMPS/.
- SuperLU is a sequential version of SuperLU_dist (not included with `petsc-complex`), and a sequential incomplete LU preconditioner that can accelerate the convergence of Krylov subspace iterative solvers. For further information, see http://crd.lbl.gov/~xiaoye/SuperLU/.
- SuperLU_dist is a package of parallel, sparse, direct linear-system solvers (available in Cray LibSci). For further information, see http://crd.lbl.gov/~xiaoye/SuperLU/.
- ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering) is a library of routines that partition unstructured graphs and meshes and compute fill-reducing orderings of sparse matrices. For further information, see http://glaros.dtc.umn.edu/gkhome/views/metis/.

- HYPRE is a library of high-performance preconditioners that use parallel multigrid methods for both structured and unstructured grid problems (not included with `petsc-complex`). For further information, see http://www.llnl.gov/CASC/linear_solvers/.
- SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) consists of 5 solvers: CVODE, CVODES, IDA, IDAS, and KINSOL. In addition, SUNDIALS provides a MATLAB interface to CVODES, IDAS, and KINSOL that is called `sundialsTB`. For further information, see https://computation.llnl.gov/casc/sundials/main.html.
- Scotch is a software package and libraries for sequential and parallel graph partitioning, static mapping, sparse matrix block ordering, and sequential mesh and hypergraph partitioning. For further information, see http://www.labri.fr/perso/pelegrin/scotch/.

Note: Although you can access these packages individually, Cray supports their use only through the PETSc or Trilinos interface.

6.1.6.1 Notes

- If you use PETSc with the Cray Fortran compiler, either add the directive `!dir$ PREPROCESS EXPAND_MACROS` to the source code or add the `-F` option to the `ftn` command line.
- The solvers in Cray PETSc are heavily optimized using the Cray Adaptive Sparse Kernels (CASK) library. CASK is an auto-tuned library within the Cray PETSc package that is transparent to the application developer, but improves the performance of most PETSc iterative solvers. You can expect the largest performance improvements when using blocked matrices (BAIJ or SBAIJ), but may also see large gains when using standard compressed sparse row (CSR) AIJ PETSc matrices.

6.1.7 Trilinos

Table 10. Trilinos Basics

Modules:	`cray-trilinos`, `cray-tpsl`
Man pages:	`intro_trilinos`(1), **Note:** Library-specific man pages are available only when the associated module is loaded.
Website:	http://trilinos.sandia.gov/

Trilinos is a separate module, comparable to PETSc, that provides abstract, object-oriented interfaces to established libraries such as Metis/ParMetis, SuperLU, Aztec, BLAS, and LAPACK. Trilinos also includes a set of Cray Adaptive Sparse Kernels (CASK) that perform SpMV, and include optimized versions of single- and multiple-vector matrix vector multiplies.

The Trilinos module is **not** loaded by default. Trilinos is dependent on the `cray-libsci` and `craype` modules. Make certain these modules are loaded before using Trilinos. When you load the `cray-trilinos` module, the Third-party Scientific Libraries (`cray-tpsl`) module is automatically loaded as well, to provide access to the libraries required to support Trilinos.

Note: Always use the `cray-tpsl` module that is linked to the Trilinos module. Trilinos and PETSc are asynchronous products and may at times use different versions of the TPSL libraries.

To use the Trilinos packages, load your compiling environment of choice, and then load the Trilinos module.

```
% module load cray-trilinos
```

After you load the Trilinos module, all header and library locations are set automatically and you are ready to compile your code. No Trilinos-specific linking information is required on the command line.

If linking to more than one Trilinos package, the libraries are linked automatically in the correct order of package dependency. For more information about link order, see http://trilinos.sandia.gov/packages/interoperability.html.

6.1.8 Cray LibSci_ACC

Cray LibSci_ACC is a library of BLAS, LAPACK and ScaLAPACK routines optimized for use on Cray systems equipped with GPU accelerators (i.e., Cray XK systems and future CPU/GPU hybrid systems). These routines enhance user application performance by generating and executing autotuned kernels for GPUs. Cray LibSci_ACC also provides a C language API to allow pass-by-value semantics for input parameters. Cray LibSci_ACC provides both automatic selection of the appropriate CPU or GPU algorithm based on problem size and data layout, and manual selection for programmers who want to control the accelerator resources used by their applications.

The Cray LibSci_ACC library is supported in the Cray and GNU programming environments by CCE 8.0 or later.

Table 11. Cray LibSci_ACC Basics

Modules:	`cray-libsci-acc`, `cray-libsci`, `PrgEnv-`*PE_type*, `craype-accel-`*GPU_type*
Man pages:	`intro_libsci_acc`(3s), `intro_libsci`(3s), `intro_blas1`(3s), `intro_blas2`(3s), `intro_blas3`(3s), `intro_lapack`(3s), `intro_scalapack`(3s) **Note:** Library-specific man pages are available only when the associated module is loaded.

The Cray LibSci_ACC module, `cray-libsci-acc`, is not loaded by default. Cray LibSci_ACC requires that the following modules be loaded:

- `cray-libsci` (loaded by default)
- A programming environment module, either `PrgEnv-cray` or `PrgEnv-gnu`
- The correct accelerator module for the type of GPU present, either `craype-accel-nvidia20` for systems with NVIDIA Fermi GPUs or `craype-accel-nvidia35` for systems with NVIDIA Kepler GPUs.

6.2 MPT

Table 12. MPT Basics

Modules:	`cray-mpich` and `cray-shmem`,
Man pages:	`intro_mpi`(3), `intro_shmem`(3) **Note:** Library-specific man pages are available only when the associated module is loaded.
Documentation:	*Getting Started on MPI I/O*
Websites:	http://www.mpi-forum.org/ http://openshmem.org/

The Cray Message Passing Toolkit (MPT) consists of two components.

- MPI (Message-Passing Interface)
- SHMEM (SHared MEMory)

MPI is a widely used parallel programming model that establishes a practical, portable, efficient, and flexible standard for passing messages between ranks in parallel processes. Cray MPI is derived from Argonne National Laboratory MPICH and implements the MPI-3.0 standard as documented by the MPI Forum in *MPI: A Message Passing Interface Standard, Version 3.0*, with the exceptions noted in MPI Usage Notes on page 67. Cray MPI is supported on all current Cray systems, for use with the Cray (CCE), GNU, Intel, and PGI compilers.

SHMEM is a similar parallel programming model, except based on using data-passing routines to put and get data in the Partitioned Global Address Space (PGAS). SHMEM was originally developed by Cray Research for use on the Cray T3D system, but has since become an open standard. Cray SHMEM is fully compliant with OpenSHMEM 1.0 and supported on all current Cray systems, for use with the Cray (CCE), GNU, Intel, and PGI compilers.

Programmers can use MPI independently of SHMEM, SHMEM independently of MPI, or both together. This has implications for module usage and linking options. For more information, see Using MPI and SHMEM Modules on page 66.

Support for MPI and SHMEM varies depending on whether you are using a Cray XC30 system with Aries interconnect, a Cray XE or Cray XK system with Gemini interconnect, or a Cray XT system with SeaStar interconnect, and depending on which version of CLE your site uses. Because of these issues, Cray provides a variety of different MPI and SHMEM modules. These modules are hardware- and OS-dependent, so your system administrator should install only the modules that support your hardware on your system.

To see which functions and environment variables are supported on your system, always check the `intro_mpi`(3) or `intro_shmem`(3) man pages.

6.2.1 Using MPI and SHMEM Modules

No MPT-related modules are loaded by default.

- If your code uses MPI code only, load the `cray-mpich` module before compiling or linking. This ensures that your code is linked using the `-lmpich` option.
- If your code uses SHMEM code only, load the `cray-shmem` module before compiling or linking. This ensures that your code is linked using the `-lsma` option.
- If your code uses both MPI and SHMEM, load both the `cray-mpich` and `cray-shmem` modules. Your code will be linked using both the `-lmpich` and `-lsma` options.

6.2.2 MPI Usage Notes

On Cray XE, Cray XK, and Cray XC30 systems, rank 0 on each node may appear to some software (for example, `malloc`) to be multi-threaded, even if the code is not actually multi-threaded. This is caused by the MPICH internal error thread that runs on each node and uses the threading library, `libpthreads`, and may safely be ignored.

Cray MPT does not support MPI 2.2 dynamic process management.

The following process-creation functions are not supported.

- `MPI_CLOSE_PORT` and `MPI_OPEN_PORT`
- `MPI_COMM_ACCEPT`
- `MPI_COMM_CONNECT` and `MPI_COMM_DISCONNECT`
- `MPI_COMM_SPAWN` and `MPI_COMM_SPAWN_MULTIPLE`
- `MPI_COMM_GET_ATTR` - with attribute `MPI_UNIVERSE_SIZE`
- `MPI_COMM_GET_PARENT`
- `MPI_LOOKUP_NAME`
- `MPI_PUBLISH_NAME` and `MPI_UNPUBLISH_NAME`

The `MPI_LONG_DOUBLE` data type is supported for Intel and GNU compilers only. It is not supported for CCE (Cray) or PGI compilers.

MPT supports the `-default64` argument for the PGI and Cray CCE compilers only. See the `ftn`(1) man page for more information about this argument. MPI-3 features, in particular new MPI-3 subroutine calls, are not supported for `-default64`.

6.2.3 SHMEM Usage Notes

On Cray XE, Cray XK, and Cray XC30 systems, rank 0 on each node may appear to some software (for example, `malloc`) to be multi-threaded even if the code is not actually multi-threaded. This is caused by the DMAPP internal error thread that runs on each node and uses the threading library, `libpthreads`, and may safely be ignored.

Typically, target or source arrays that reside on remote processing elements (PEs) are identified by passing the address of the corresponding data object on the local PE. The local existence of a corresponding data object implies that a data object is *symmetric*.

Symmetric accessible data objects passed to SHMEM routines can be arrays or scalars. A *symmetric* data object is one where the local and remote addresses have a known relationship. You can use SHMEM routines to access remote symmetric data objects by using the address of the corresponding data object on the local PE.

The following data objects are symmetric:

- Fortran data objects in common blocks or with the SAVE attribute.
- Non-stack C and C++ variables.
- Fortran arrays allocated with `shpalloc`(3f)
- C and C++ data allocated by `shmalloc`(3c)

Only local addresses can be used as arguments to SHMEM calls. For instance, using an address obtained by exchanging addresses between PEs is not supported.

Whether the addresses returned by a collective call to `shmalloc` are the same for all PEs depends on the configuration of Virtual Memory Randomization (VMR), a Linux kernel feature:

- Default configuration (VMR is turned OFF): The addresses returned by a collective call to `shmalloc` may or may not return the same addresses for all PEs, depending on application characteristics. For a well-behaved application, the addresses may be the same for all PEs.
- Non-default configuration (VMR is turned ON): The Cray SHMEM library prints an informational message similar to `LIBSMA INFO: PE0: Linux VM Randomization is turned on on this system`. A collective call to `shmalloc` does not return the same addresses for all PEs.

A SHMEM application must call `start_pes` or `shmem_init` as the very first SHMEM routine called within the application to guarantee that lower-level resources are set up correctly. Otherwise, the SHMEM application does not execute correctly. Similarly, a SHMEM application must call `shmem_finalize` as the very last SHMEM routine called within the application to guarantee correct cleanup of previously allocated network protocol resources.

SHMEM routines can be used in conjunction with Message Passing Interface (MPI) routines in the same application. Programs that use both MPI and SHMEM should call `MPI_Init` followed by `start_pes` or `shmem_init`. At the end of the program, `shmem_finalize` should be called followed by `MPI_Finalize`. SHMEM processing element numbers are equal to the MPI rank within the `MPI_COMM_WORLD` communicator, if the MPI job consists of a single application.

Note: Alternatively, you can use the `MPICH_GNI_DMAPP_INTEROP` environment variable to control MPI, SHMEM, UPC, and Coarray Fortran interoperability.

Note: Multi-binary SHMEM jobs are not currently supported on CNL.

The SHMEM routines reside in `libsma.a`. The following command lines compile programs that include SHMEM routines:

```
cc c_program.c
CC cplusplus_program.C
ftn fortran_program.f
```

6.2.4 GPU-to-GPU Communications

The GPU-to-GPU feature allows an MPI application to use GPU pointers in MPI point-to-point and collective communication routines. The GPU-to-GPU feature improves performance by pipelining data transfers between GPU, host CPU, and network, and simplifies the code by eliminating explicit CUDA calls for moving data between GPU and host.

The following example illustrates the steps needed to compile and execute a program that performs a reduction of data residing in GPU buffers and stores the results in the rank 0 host buffer.

Procedure 2. Using GPU-to-GPU Communications

1. Load the `cudatoolkit` module.

   ```
   $ module load cudatoolkit
   ```

   Load the accelerator module.

2. If necessary, determine which accelerator module is required, and then load the accelerator module.

   ```
   $ cnselect -L subtype
   nVidia_Kepler
   $ module load craype_accel_nvidia35
   ```

   For more information about available accelerator modules, see Targeting for an Accelerator on page 31.

3. Compile the program. (An example program is provided later in this section.) If the Cray CCE compiler is used, the `-h gnu` option is required in order to compile source files that include `cuda.h`.

   ```
   $ cc -h gnu -o reduce_g2g reduce_g2g.c
   ```

4. Set the `MPICH_RDMA_ENABLED_CUDA` environment variable to enable the GPU-to-GPU feature, as described in the `intro_mpi`(3) man page, and set the `CRAY_CUDA_PROXY` environment variable to allow CUDA calls to be made by multiple MPI processes on the same node. For example, the following batch script specifies eight PEs, at two per node.

   ```
   #!/usr/bin/env sh
   #PBS -l mppwidth=8,mppnppn=2,walltime=1:00:00
   cd $PBS_O_WORKDIR
   export CRAY_CUDA_PROXY=1 MPICH_RDMA_ENABLED_CUDA=1
   aprun -n 8 -N 2 reduce_g2g
   ```

5. Submit the job, reserving nodes with the desired GPU type.

   ```
   $ qsub -V -l mppnodes=\"$(cnselect -e
   subtype=nVidia_Kepler)\" example.job
   ```

   This results in the following output:

   ```
   $ cat reduce_g2g.sh.o6587140
   28 28 28 28
   ```

Here is a listing of the code used in the above example.

```
#include <cuda.h>
#include <cuda_run time.h>
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    const int ARRAY_SIZE = 4;
    const int ARRAY_BYTES = ARRAY_SIZE*sizeof(int);

    int i, rank, *gpu_buffer, host_buffer[ARRAY_SIZE];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Initialize each element of host_buffer to the MPI rank.
    for (i = 0; i < ARRAY_SIZE; i++)
        host_buffer[i] = rank;

    // Allocate GPU buffer.
    cudaMalloc((void**)&gpu_buffer, ARRAY_BYTES);
    // Copy host_buffer to gpu_buffer
    cudaMemcpy(gpu_buffer, host_buffer, ARRAY_BYTES, cudaMemcpyHostToDevice);

    // Global reduction: sum elements of gpu_buffer into rank 0 host_buffer.
    MPI_Reduce(gpu_buffer, host_buffer, ARRAY_SIZE, MPI_INT, MPI_SUM, 0,
               MPI_COMM_WORLD);

    MPI_Finalize();

    if (rank == 0) {
        for (i = 0; i < ARRAY_SIZE-1; i++)
            printf("%d ", host_buffer[i]);
        printf("%d\n", host_buffer[ARRAY_SIZE-1]);
    }
}
```

6.3 Hugepages

Table 13. Hugepages Basics

Modules:	`craype-hugepages`*pagesize*
Man pages:	`intro_hugepages`(1)

Hugepages are virtual memory pages which are bigger than the default base page size of 4KB. Hugepages can improve memory performance for common access patterns on large data sets. Access to hugepages is provided through a virtual file system called `hugetlbfs`. Every file on this file system is backed by huge pages and is directly accessed with `mmap()` or `read()`.

The `libhugetlbfs` library allows an application to use huge pages more easily than it could by directly accessing the `hugetlbfs` file system. A user may use `libhugetlbfs` to back application text and data segments.

Due to differing memory management mechanisms on Cray XT and Cray XE/Cray XK systems, the implementation of the `libhugetlbfs` library differs on these two architectures. Due to the different router chips, implementation on Gemini-based (Cray XE/Cray XK) and Aries-based (Cray XC30) systems also differs.

6.3.1 When to Use Hugepages

- For SHMEM applications, map the static data and/or private heap onto huge pages.
- For applications written in Unified Parallel C, Coarray Fortran, and other languages based on the PGAS programming model, map the static data and/or private heap onto huge pages.
- For MPI applications, map the static data and/or heap onto huge pages.
- For an application which uses shared memory, which needs to be concurrently registered with the high speed network drivers for remote communication.
- For an application doing heavy I/O.
- To improve memory performance for common access patterns on large data sets.

6.3.2 When to Avoid Using Hugepages

Applications sometimes consist of many steering programs in addition to the core application. Applying huge page behavior to all processes would not provide any benefit and would consume huge pages that would otherwise benefit the core application.

6.3.3 Cray XC30 Usage

On Cray XC30 systems, huge pages are available by default. Modules `craype-hugepages2M`, `craype-hugepages4M`, `craype-hugepages8M`, `craype-hugepages16M`, `craype-hugepages32M`, `craype-hugepages64M`, `craype-hugepages128M`, `craype-hugepages256M`, and `craype-hugepages512M` set the necessary link options and environment variables (e.g., `HUGETLB_DEFAULT_PAGE_SIZE`, `HUGETLB_MORECORE`, `HUGETLB_ELFMAP`) to facilitate the usage of 2MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256BMB, and 512MB huge pages, respectively.

In Cray systems that have the Aries NIC, the Aries IO Memory Management Unit (IOMMU) provides hardware support for memory protection and address translation. The Aries IOMMU uses an entirely different memory translation mechanism than Gemini uses: the IOMMU is divided into 16 translation context registers (TCRs). Each translation context (TC) supports a single page size. The TCRs can independently address different page sizes and present that to the network as a contiguous memory domain. The TCR entries are used to set and clear the page table entries (PTEs) used by GNI. PTE entries are cached in Aries NIC memory in a page table. Up to 512 PTEs can be used by applications. 512MiB (largest hugepage size) x 512 PTEs = 256GiB of addressable memory per node on Aries systems.

For more detailed information and examples see the `intro_hugepages`(1), and `aprun`(1) man pages, and *Workload Management and Application Placement for the Cray Linux Environment*.

6.3.4 Cray XE and Cray XK Usage

By default, the system is configured to have huge pages available. Pre-allocated huge pages are reserved inside the kernel and cannot be used for other purposes. On Cray XE and Cray XK systems, PGAS, SHMEM and MPI applications are likely to require the usage of huge pages for static data and/or the heap.

Modules `craype-hugepages128K`, `craype-hugepages512K`, `craype-hugepages2M`, `craype-hugepages8M`, `craype-hugepages16M`, and `craype-hugepages64M` set the necessary link options and environment variables (e.g., `HUGETLB_DEFAULT_PAGE_SIZE`, `HUGETLB_MORECORE`, `HUGETLB_ELFMAP`) to facilitate the usage of 128KB, 512KB, 2MB, 8MB, 16MB, or 64MB huge pages, respectively.

It is not required to use the `-m` option on the `aprun` command on the Cray XE/Cray XK system to allocate huge pages, because the kernel allows the dynamic creation of huge pages. However, it is advisable to specify this option and preallocate an appropriate number of huge pages, when memory requirements are known, to reduce operating system overhead.

For more detailed information and examples see the intro_hugepages(1), and aprun(1) man pages, and *Workload Management and Application Placement for the Cray Linux Environment*.

6.3.5 Cray XT Usage

Nodes do not have huge pages allocated by default.

To use hugepages, link an application with the libhugetlbfs library.

At run time, define the environment variable HUGETLB_MORECORE=yes.

The application launcher, aprun, must be told that a given application wants to use huge pages. Specify a per-PE huge page memory requirement on the aprun invocation line using the [-m *size*[h|hs]] option.

For more information see the aprun(1) man page, and *Workload Management and Application Placement for the Cray Linux Environment*.

6.3.6 Running Independent Software Vendor (ISV) Applications

To enable a dynamically linked executable, that **was not** originally linked with libhugetlbfs, to use Cray's libhugetlbfs library at run time, you must first load a hugepages module and set the environment variable LD_PRELOAD so that it contains the libhugetlbfs pathname:

```
module load craype-hugepages2M
export LD_PRELOAD=/usr/lib64/libhugetlbfs.so
```

If an ISV application is already using LD_PRELOAD to set dynamic library dependencies, then use a white-space separated list. For example:

```
export LD_PRELOAD="/usr/lib64/libhugetlbfs.so /directory_name/lib.so"
```

To confirm the usage of hugepages, one may set HUGETLB_VERBOSE to 3 or higher:

```
export HUGETLB_VERBOSE=3
```

Statically linked executables can only use Cray's libhugetlbfs if they are linked with it. Statically linked executables do not process LD_PRELOAD; therefore statically linked ISVs must be relinked with libhugetlbfs.

The nm and ldd commands are useful for determining the contents and dynamic dependencies of executables.

Note: ISV applications sometimes consist of scripts which run several executables, only some of which need to run with huge pages. The environment variable HUGETLB_RESTRICT_EXE enables the libhugetlbfs library to selectively map only the named executables onto huge pages.

6.3.7 Known Issues

Huge pages are a per-node resource, not a per-job resource, nor a per-process resource. There is no guarantee that the requested number of huge pages will be available on the compute nodes. If the memory pool becomes fragmented, which it can over time, the number of free blocks **that are equal to or larger than the huge page size** can decrease below the number needed to service the request, even though there may be enough free memory in the pool when summing free blocks of all sizes. For this reason, use huge page sizes no larger than needed.

If the heap is mapped to huge pages (by setting `HUGETLB_MORECORE` to `yes`) and if a `malloc` call requires that the heap be extended, and if there are not enough free blocks in the memory pool large enough to support the required number of huge pages, `libhugetlbfs` will issue the following `WARNING` message and then `glibc` will fall back to allocating base pages.

```
libhugetlbfs [nid000xx:xxxxx]: WARNING: New heap segment map at
0x10000000 failed: Cannot allocate memory
```

This is a warning and jobs are able to continue running. The allocated base pages use GART entries; however, because there are a limited number of GART entries, future memory requests may fail altogether due to lack of available GART entries.

With `craype-hugepages` modules loaded, it is no longer necessary to include `-lhugetlbfs` on the link line. Doing so will result in messages indicating multiple definitions, such as:

```
//usr/lib64/libhugetlbfs.a(elflink.o): In function
`__libhugetlbfs_do_remap_segments':

/usr/src/packages/BUILD/cray-libhugetlbfs-2.11/elflink.c:2012:
multiple definition of `__libhugetlbfs_do_remap_segments'

//usr/lib64/libhugetlbfs.a(elflink.o):/usr/src/packages/BUILD/
cray-libhugetlbfs-2.11/elflink.c:2012: first defined here
```

Adjust makefiles or build scripts accordingly.

Debugging Code [7]

The Cray system supports a variety of debugging options ranging from simple command-line debuggers to separately licensed third-party GUI tools. These options are capable of performing a variety of tasks ranging from analyzing core files to setting breakpoints and debugging running parallel programs.

As a rule, your code must be compiled using the `-g` command line option before you can use any of the debuggers to produce meaningful information. If, however, you are using both a compiler and a debugger that support Fast-track Debugging, the `-g` option is replaced by using the `-G fast` option.

Note: The PGI debugger, `PGDBG`, is not supported on Cray systems.

Note: At this time, systems equipped with Intel Xeon Phi coprocessors are supported by DDT and TotalView only. The Cray Debugger Support Tools and Cray Fast-track Debugging are not supported on Intel Xeon Phi systems. If your site is configured to load the `atp` module by default, you must unload this module before loading the `craype-intel-knc` module.

7.1 Cray Debugger Support Tools

Cray provides a collection of basic debugging packages that are referred to collectively as the *Cray Debugger Support Tools* and are installed as a single rpm, but loaded and used as individual modules. These packages are:

- CCDB: Cray's comparative debugger features a graphical user interface and extends the comparative debugging capabilities of `lgdb`, allowing users to easily compare data structures between two executing applications.
- LGDB: a GDB-based parallel debugger used to debug applications compiled with CCE, PGI, GNU, Intel Fortran, C and C++ compilers. It allows programmers to either launch an application or attach to an already-running application that was launched with `aprun`.
- ATP: a system that monitors user applications and replaces the core dump with a more comprehensive stack backtrace and analysis.
- STAT: stack trace analysis tool

7.1.1 Using CCDB

Table 14. `ccdb` Basics

Module:	`cray-ccdb`
Command:	`ccdb`
Man page:	`ccdb`(1) **Note:** Tool-specific man pages are available only when the associated module is loaded.
Online help:	Embedded in user interface.

To begin using CCDB on the Cray system, load the `cray-ccdb` module:

```
> module load cray-ccdb
```

Launch the CCDB application using the `ccdb` command. After you do so, the main control and display window pops open.

```
> ccdb
```

Note: Users running CCDB remotely on a workstation may be prompted for a password when attempting to access a Cray system.

CCDB includes an integrated help system that includes all additional information about using CCDB. Help is accessible whenever CCDB is running by either clicking the **?** button in the current window or selecting **Help** from the menu bar of the CCDB monitor window.

Note: CCDB is a GUI tool that requires your workstation to support the X Window System. Depending on your system configuration, you may need to use the `ssh -X` option to enable X Window System support in your shell session. Depending on your workstation configuration, you may also need to enable X Window System hosting on your workstation or load an X Window client such as Xming.

7.1.2 Using LGDB

Table 15. `lgdb` Basics

Module:	`cray-lgdb`
Command:	`lgdb`
Man page:	`lgdb`(1) **Note:** Tool-specific man pages are available only when the associated module is loaded.
Online help:	Embedded in user interface.

To initiate an LGDB session, enter the `lgdb` command. After you do so, the debugger command prompt is displayed:

```
lgdb 2.3 - Cray Line Mode Parallel Debugger
With Cray Comparative Debugging Technology.
Copyright 2007-2013 Cray Inc. All Rights Reserved.
Copyright 1996-2013 Monash University. All rights Reserved.

Type "help" for a list of commands.
dbg all>
```

Once `lgdb` is running and the command prompt is displayed, the program uses a command-line interface similar to that used by `gdb`.

LGDB includes extensive online help. Enter `help` at the command prompt to display the list of help topics.

```
dbg all> help
assign          Change the value of a program variable.
attach          Attach to an application under debugger control.
backtrace       Print backtrace of all stack frames.
break           Set breakpoint at specified line or function.
build           Build an assertion script.
compare         Compare the contents of two variables.
continue        Continue program being debugged, after signal or breakpoint.
decomposition   Define a decomposition scheme.
defset          Create a set of processes.
delete          Delete a breakpoint.
disable         Disable a breakpoint.
down            Move down one or more stack frames.
enable          Enable a breakpoint.
finish          Execute program being debugged until current function returns.
focus           Set the current process set.
frame           Print the currently selected stack frame.
gdbmode         Enter gdb direct mode (experimental).
halt            Halt execution of application under debugger control.
help            Display help information about commands.
info            Display information about the application being debugged.
kill            Kill an application under debugger control.
launch          Launch an application under debugger control.
list            List source for specified function or line.
maint           Commands for use by ccdb maintainers.
next            Step program, proceeding through subroutine calls.
print           Print the value of an expression.
quit            Exit the debugger.
release         Release an application from debugger control.
session         Create a new WLM session.
set             Set information about the debugger environment.
show            Show information about the debugger environment.
source          Read debugger commands from a file.
start           Start executing a dataflow graph.
step            Step program until it reaches a different source line.
stop            Stop the currently executing dataflow graph.
tbreak          Set a temporary breakpoint at a specified line or function.
unset           Unset information in the debugger environment.
                     up             Move up one or more stack frames.
```

```
usage           Display usage information about deferred mode commands.
viewset         Display information about process sets.
watch           Set watchpoint on specified expression.
whatis          Print the data type of an expression.
```

Enter `help` *topic* for more information about a given *topic*.

For more information about using the `launch` command to launch an application from within LGDB or using the `attach` command to attach the debugger to an already-running process, see the `lgdb`(1) man page and the `launch` and `attach` topics in the help system.

7.1.3 Using Abnormal Termination Processing (ATP)

Table 16. `atp` Basics

Module:	`atp`
Commands:	`ataprun`
Man page:	`intro_atp`(1)
	Note: Tool-specific man pages are available only when the associated module is loaded.

Abnormal Termination Processing (ATP) monitors user applications. When the `atp` module is loaded and ATP is enabled, ATP is launched when a job is started and delivers a heuristically determined set of core files in the event of an application crash. If an application takes a system trap, ATP performs analysis on the dying application. All stack backtraces of the application processes are gathered into a merged stack backtrace tree and written to disk as the file, `atpMergedBT.dot`. The stack backtrace tree for the first process to die is sent to `stderr` as is the number of the signal that caused the application to fail.

The `atpMergedBT.dot` file can be viewed with `stat-view`, (the Stack Trace Analysis Tool viewer). The merged stack backtrace tree provides a concise yet comprehensive view of what the application was doing at the time of its termination. For more information about using `stat-view`, see the `stat-view`(1) man page.

Note: The `stat-view` command and man page are available only when the `stat` module is loaded.

ATP is designed to analyze failing applications. It does not play any role with commands. That is, an application must use a supported parallel programming model, such as MPI, SHMEM, OpenMP, CAF, or UPC, in order to benefit from ATP analysis. When the `atp` module is loaded, ATP sets the `MPICH_ABORT_ON_ERROR`, `SHMEM_ABORT_ON_ERROR`, and `DMAPP_ABORT_ON_ERROR` environment variables. This enables MPI, SHMEM, and DMAPP applications to raise a signal when they discover usage errors—rather than only printing to `stderr` and exiting—which, therefore, enables ATP to notice the problem and perform its analysis.

Note: Using ATP disables the Linux standard of dumping core and replaces it with dumping a set of files named `atp.core.`*apid*`.`*rank* for application crashes.

ATP is site-configurable to be enabled by default and launched automatically whenever a job is launched using the `aprun` command. For more information about using ATP, see the `intro_atp`(1) man page.

7.1.4 Using STAT

Table 17. STAT Basics

Module:	`stat`
Commands:	`stat-gui`, `stat-view`
Man pages:	`intro_stat`(1), `stat-gui`(1), `stat-view`(1) **Note:** Tool-specific man pages are available only when the associated module is loaded.
Online help:	http://www.paradyn.org/STAT/STAT.html

Note: The STAT command naming convention has changed; `statgui` and `statview` are deprecated and will be removed in the next release of STAT. They are replaced by `stat-gui` and `stat-view`.

STAT (Stack Trace Analysis Tool) gathers and merges the stack traces from a parallel application's processes and produces 2D spatial and 3D spatial-temporal call graphs that encode the calling behavior of the application processes in the form of a prefix tree. The 2D graph represents a single snapshot of the entire application, while the 3D form represents a series of snapshots from the application taken over time.

7.2 Using Cray Fast-track Debugging

Normally, code must be compiled using the `-g` option before it can be debugged using a conventional debugger. The `-g` option typically disables all compiler optimizations, producing an executable containing full DWARF information that can be breakpointed, stepped-through, or paused and restarted anywhere, but at the cost of a much larger and far slower-running program.

Cray Fast-track Debugging significantly increases the speed of the debugging process by producing executables that contain both full DWARF information **and** run at optimized-code speed. Essentially this is done by producing two parallel executables: one that is fully optimized, and another that is not. While this combined executable is considerably larger than a normal executable, when it is executed under the control of a debugger that supports fast-track debugging, it runs at optimized-code speed until it hits a break point—at which time it switches to the unoptimized code, and allows you to set breakpoints, examine registers, pause, resume, and step through the code as if the entire program was compiled using the `-g` option.

As far as the debugger is concerned, there are no user interface changes, aside from the possibility that you might pursue a backtrace far enough back to begin seeing internal names instead of user names for variables. (For example, instead of `foo`, you might see `debug$foo`.) All the work involved in using fast-track debugging is done on the compiler side.

Procedure 3. Using Cray Fast-track Debugging

Using Cray Fast-track Debugging is a two-step process, requiring both a compiler and a debugger that support fast-track debugging. The steps are:

1. Compile your program using your compiler's fast-track debugging option. For example, if you are using the Cray Fortran compiler, compile and link your code using the `-G fast` option:

   ```
   users/yourname> ftn -Gfast myapp.f
   ```

2. Execute your program using your debugger's normal control method. For example, if you are using the `lgdb` command line debugger to debug a single-rank application, first execute `lgdb` and then use the `launch` command to launch the application:

   ```
   $ lgdb
   dbg all> launch $a ./myapp
   ```

   Once the debugging session is launched, fast-track debugging is transparent to the user. Sections of the code that contain breakpoints execute slowly, as if compiled using the `-g` option. Other sections of the code that do not contain breakpoints execute at normal speed, as if compiled using normal optimizations.

   For more information about `lgdb`, see the `lgdb`(1) man page and the help system within `lgdb`.

7.2.1 Supported Compilers and Debuggers

At this time, Cray Fast-track Debugging is supported on the front end by the Cray Compiling Environment (CCE) compilers.

On the back end, Cray Fast-track Debugging is supported by `ccdb`, `lgdb` and Allinea DDT debuggers.

7.3 About Core Files

On Cray systems, in the absence of ATP, when an application fails, one core file is generated for the first failing process. If a file named `core` already exists in the current working directory, it is not overwritten.

On large MPP systems where an application might be running thousands of processes, a conventional core file may not be sufficient to indicate the actual cause of the failure. For this reason, Cray systems support Abnormal Termination Processing (ATP). If ATP is enabled, a failing application generates a heuristically determined set of core files in place of a single core file. A core file is created for each set of processes that have the same backtrace, which is determined by comparing routine names (not line numbers or memory addresses) and after pruning out system routines. For SPMD (Single Program, Multiple Data) programs, this typically causes only a small handful of core files to be generated.

For more information about ATP, see Using Abnormal Termination Processing (ATP) on page 78.

7.4 Using DDT

Table 18. DDT Basics

Module:	`ddt`
Commands:	`ddt`
Man page:	`ddt`(1) **Note:** Tool-specific man pages are available only when the associated module is loaded.
Online help:	The DDT GUI includes an extensive online help system accessible by selecting **Help** from the menu. If you have problems displaying the help, see the *DDT User Guide* for information about configuring X-Windows forwarding and VNC connections.
Documentation:	`/opt/cray/ddt/`*version*`/doc`

DDT, an optional product from Allinea Software is a scalable debugger with a graphical user interface. It can be used to debug Fortran, C, and C++ programs, including MPI and OpenMP code, and to launch and debug programs, attach to already running programs, or open and debug core files. DDT is compatible with the Cray, PGI, GCC, and Intel compilers.

The DDT GUI requires either X-Windows forwarding or VNC in order to work.

DDT can be used either within an interactive shell or via the batch system. Submission through a batch system requires the use of template files that specify the batch parameters. Sample template files are found in `/opt/cray/ddt/`*version*`/templates`.

In an interactive shell, the fastest way to launch DDT is by entering the `ddt` command:

```
users/yourname> ddt
```

Assuming X-Windows forwarding is configured correctly, the main program window displays a pop-up menu offering the following options.

- Run and Debug a Program
- Debug a Multi-Process Non-MPI Program
- Attach to a Running Program
- Open a Core File
- Restore a Checkpoint
- Cancel

Select the option you want to use, or **Cancel** to close the pop-up menu and proceed to the DDT main window. All the above options are also available through the **Sessions** menu **Run** option.

7.4.1 Known Limitations

DDT has a number of defaults that affect batch queue submission behavior. When you begin a DDT session, either by selecting **Run and Debug a Program** from the pop-up **Welcome** menu or by selecting **Run** from the **Session** menu in the DDT main window, the Queue Submission Mode window displays. If you use a batch queuing system such as PBS Pro, **always verify the Queue Submission Parameters** before proceeding.

In particular, verify that the default Queue name and Procs Per Node match your system's configuration. The default Queue name is generally site-specific, while the Procs Per Node value must match your system's processor types: dual-core, quad-core, and so on.

If you need to change the Queue Submission Parameters, click the **Change** button on the Queue Submission Mode window to do so for the duration of the current session. Alternatively, you can create a template file that stores your preferred parameters. Instructions for creating and using template files are provided in the *DDT User Guide*.

To change your system's default Queue Submission Parameters for all users, contact your site administrator. Default configuration information is stored in the `/opt/cray/ddt/`*version*`/default-config.ddt` file. This information includes the name of the default template, which currently is `/opt/cray/ddt/`*version*`/templates/xt4.qtf`. The actual default queue submission parameters are specified in the default template file.

7.5 Using TotalView

Table 19. TotalView Basics

Module:	`totalview`
Commands:	`totalview`, `totalviewcli`
Man page:	`totalview`(1) **Note:** Tool-specific man pages are available only when the associated module is loaded.
Online help:	The TotalView GUI contains an extensive HTML online help system but requires that `$TV_HTMLHELP_VIEWER` be defined before use. For more information, see the Totalview documentation.
Documentation:	`/opt/totalview/`*version*`/doc`

TotalView is an optional product from Rogue Wave Software that provides source-level debugging of applications running on multiple compute nodes. TotalView is compatible with the Cray, PGI, GNU, and Intel compilers.

TotalView can be launched in either or two modes: in GUI mode (using the `totalview` command), or in command-line mode (using the `totalviewcli` command). TotalView is typically run interactively. If your site has not designated any compute nodes for interactive processing, use the `qsub -I` command to reserve the number of compute nodes you want to use in interactive mode.

Example 1. Using TotalView to control program execution

To debug an application on the Cray system, use TotalView to launch `aprun`, which in turn launches the application to be debugged.

```
users/yourname> totalview aprun -a [aprun_arguments] ./myapp
[myapp_arguments]
```

The `-a` option is a TotalView option indicating that the arguments that follow apply to `aprun`, not TotalView.

Example 2. Debugging a core file

To use TotalView to examine a core file, use the `totalview` command to launch the GUI. Then, in the **New Program** window, click the **Open a core file** button and use the browse functions to find, select, and open the core file you want to examine.

Example 3. Attaching TotalView to a running process

To attach TotalView to a running process, you must be logged in to the same login node that you used to launch the process, and then you must attach to the instance of `aprun` that was used to launch the process, not the process itself. To do so:

1. Use the `totalview` command to launch the GUI.
2. In the **New Program** window, click the **Attach to process** button. The list of processes currently running displays.
3. Select the instance of `aprun` that you want, and click **OK**. TotalView displays a process window showing both `aprun` and the program threads that were launched by that instance of `aprun`.

7.5.1 Known Limitations

The TotalView debugging suite for Cray systems differs in functionality from the standard TotalView implementation. It does not support:

- Debugging `MPI_Spawn()`, OpenMP, or Cray SHMEM programs.
- Compiled `EVAL` points and expressions.
- Type transformations for the PGI C++ compiler standard template library collection classes.
- Exception handling for the PGI C++ compiler run time library.
- Spawning a process onto the compute processors.
- Machine partitioning schemes, gang scheduling, or batch systems.

Optimizing Code [8]

After your code is compiled, debugged, and capable of running to completion or planned termination, you can begin looking for ways in which to improve execution speed. In general, the opportunities for optimization fall into three categories, which require progressively more programmer effort. These categories are:

- Improving overall I/O
- Improving use of compiler-generated optimizations
- Analyzing code behavior and rewriting code to optimize performance

8.1 Improving I/O

8.1.1 Using `iobuf`

Table 20. IOBUF Basics

Module:	`iobuf`
Man page:	`iobuf`(3)
Environment variable:	`IOBUF_PARAMS`

IOBUF is an I/O buffering library that can reduce the I/O wait time for programs that read or write large files sequentially. IOBUF intercepts standard I/O calls such as `read` and `open` and adds a layer of buffering, thus improving program performance by enabling asynchronous prefetching and caching of file data.

IOBUF can also gather run time statistics and print a summary report of I/O activity for each file.

IOBUF is not suitable for all I/O styles. IOBUF does not maintain coherent buffering between processes that open the same file. For this reason, do not use IOBUF with shared file I/O, such as MPI-IO routines like `MPI_File_write_all`. IOBUF is not thread-safe, so do not use it with multithreaded programs in which the threads perform buffered I/O. IOBUF can be linked into programs that use these I/O styles, but buffering should not be enabled on those files.

In general, no program source changes are needed in order to take advantage of IOBUF. Instead, IOBUF is implemented by following these steps:

1. Load the IOBUF module:

   ```
   % module load iobuf
   ```

2. Relink the program.

3. Set the IOBUF_PARAMS environment variable as needed.

   ```
   % setenv IOBUF_PARAMS='*:verbose'
   ```

4. Execute the program.

If a memory allocation error occurs, buffering is reduced or disabled for that file and a diagnostic is printed to `stderr`. When the file is opened, a single buffer is allocated if buffering is enabled. The allocation of additional buffers is done when a buffer is needed. When a file is closed, its buffers are freed (unless asynchronous I/O is pending on the buffer and `lazyclose` is specified).

The behavior of IOBUF is controlled by the use of environment variables:

- `IOBUF_PARAMS`

  Selects files and sets parameters for buffering. If this environment variable is not set, the default state is no buffering and the I/O call is passed on to the next layer without intervention.

  The simplest parameter specification is **`export IOBUF_PARAMS='*'`**. This setting matches all files and enables buffering with default parameters. For more information about valid `IOBUF_PARAMS` parameters and their usage, see the `iobuf`(3) man page.

- `IOBUF_DEBUG`

  If set, library debugging output is printed. This feature is mainly for debugging the library itself. Valid settings are `read`, `write`, `heap`, `open`, `buffers`, `wait`, `params`, and `all`.

- `IOBUF_MAX_FILES`

  The maximum file descriptor number managed by IOBUF. Roughly equivalent to the maximum number of files which can be open with buffering at the same time. The default is 256. Each file descriptor requires about 512 bytes, so the default setting requires 32 KB of memory.

See the `iobuf`(3) man page for more information regarding the usage of the IOBUF library.

8.1.2 Improving MPI I/O

Table 21. MPI I/O Basics

Module:	`cray-mpich2`
Man page:	`intro_mpi`(3)
Environment variables:	`MPICH_MPIIO_HINTS`, `MPICH_RANK_REORDER_METHOD`, others
Documentation:	*Getting Started on MPI I/O*

When working with MPI code, one of the most effective ways to realize significant improvements in program execution speed is by fine-tuning MPI rank placement and I/O usage. The Cray Message Passing Toolkit (MPT) provides more than forty environment variables designed to help you do just that, the two most significant of which are `MPICH_MPIIO_HINTS` and `MPICH_RANK_REORDER_METHOD`. For a listing of the MPI environment variables and their valid values and uses, see the `intro_mpi`(3) man page.

A full discussion of MPI I/O optimization is beyond the scope of this document. For more information on this subject, including detailed explanations and examples, see *Getting Started on MPI I/O*.

Optimizing MPI rank placement can require considerably more detailed analysis. Alternately, you can use Cray Performance Analysis Tools to instrument your program to study MPI behavior, and then to generate suggested MPI rank reordering information. For more details, see the `intro_craypat`(1), `pat_build`(1), and `pat_report`(1) man pages, and *Using Cray Performance Measurement and Analysis Tools*.

8.2 Using Compiler Optimizations

This section collects some of the more common tips and tricks for getting better-performing code out of the compilers. This section will be expanded as information is developed.

8.2.1 Cray Compiling Environment (CCE)

The Cray Fortran and C/C++ compilers are optimizing compilers that perform substantial analysis during compilation and generate highly optimized code automatically. The Cray compilers also support a large number of command-line arguments that enable you to exert manual control over compiler optimizations, and fine-tune the behavior of the compiler.

For more detailed information about the Cray Fortran, C, and C++ compiler command-line arguments, see the `crayftn`(1), `craycc`(1), and `crayCC`(1) man pages, respectively.

Two of the most useful compiler command-line arguments are the Fortran `-rd` and C/C++ `-h list=m` options that instruct the compiler to generate annotated loopmark listings showing what optimizations were performed and their locations. Together with the `-h negmsgs` option that generates listings showing potential optimizations that were **not** performed, and why, these arguments can help you zero-in on areas in your code that are compiling without error, but not with maximum efficiency.

For more detailed information about generating and reading loopmark listings, see the *Cray Fortran Reference Manual* and *Cray C and C++ Reference Manual*.

The Cray compilers also support a large number of pragmas and directives that enable you to exert manual control over compiler optimization behavior. In many cases, code that is not optimizing well can be corrected without substantial changes to the code itself, but simply by applying the right pragmas or directives.

For more information about Cray compiler pragmas and directives, see the `intro_directives`(1) man page.

8.3 Using the Cray Performance Measurement and Analysis Tools

Table 22. Performance Analysis Basics

Module:	`perftools`, `perftools-lite`
Commands:	`pat_build`, `pat_report`, `app2`
Man pages:	`intro_craypat`(1), `craypat-lite`(1), `pat_build`(1), `pat_report`(1), `pat_help`(1), `app2`(1), `reveal`(1), `intro_papi`(3), `hwpc`(5), `nwpc`(5), `accpc`(5), `rapl`(5) **Note:** Tool-specific man pages are available only when the associated module is loaded.
Online help:	CrayPat includes an extensive online help system that features many examples and the answers to many frequently asked questions. To access the help system, enter **`pat_help`** at the command line.
Documentation:	*Using Cray Performance Measurement and Analysis Tools*

Note: The PGI profiling tools, `pgprof` and `pgcollect`, are not supported on Cray systems.

Note: The GNU profiling tool, `gprof` is not supported on Cray systems.

After you have compiled and debugged your program, you are ready to begin analyzing its performance. The Cray Performance Analysis Tools are a suite of optional utilities that enable you to capture and analyze performance data generated during the execution of your program in order to help you to find answers to two fundamental questions: *How fast is my program running?* and *How can I make it run faster?*

The performance analysis process consists of three basic steps.

1. **Instrument** your program, to specify what kind of data you want to collect under what conditions.
2. **Execute** your instrumented program, to generate and capture the desired data.
3. **Analyze** the resulting data.

Accordingly, the Cray Performance Measurement and Analysis Tools suite consists of these major components:

- **CrayPat-lite**: a new, simplified, and easier-to-use front-end for CrayPat that provides basic performance analysis information automatically, with a minimum of user interaction.
- **CrayPat**: the full-featured performance analysis tool set, which enables users to instrument programs, capture performance data during program execution, and generate extensive text reports from the resulting data
- **Cray Apprentice2**: the second-level data analysis tool, used to visualize, manipulate, explore, and compare sets of program performance data in a GUI environment
- **Reveal**: the next-generation integrated performance analysis and code optimization tool, which enables users to correlate performance data captured during program execution directly to the original source, and identify opportunities for further optimization
- **PAPI**: the Performance Application Programming Interface

Note: At this time, CrayPat-lite is not supported on systems equipped with Intel Xeon Phi coprocessors. The full version of CrayPat is supported on such systems, but with reduced functionality. In particularly, hardware counter data collection and all functions and reports related to the `PAT_RT_PERFCTR` set of environment variables are not currently supported.

8.3.1 About CrayPat-lite

CrayPat-lite is a simplified, easy-to-use version of the Cray Performance Measurement and Analysis Tool set. CrayPat-lite provides basic performance analysis information automatically, with a minimum of user interaction, and yet offers information useful to users wishing to explore their program's behavior further using the full CrayPat tool set.

To use CrayPat-lite, load the `perftools-lite` module.

CrayPat-lite supports three basic experiments:

- `sample_profile` — A sampling experiment, which reports execution time, aggregate MFLOP count, the top time-consuming functions and routines, MPI behavior in user functions (if the application is an MPI program), and generates the data files listed above. This is the default experiment.
- `event_profile` — A tracing experiment, which generates a profile of the top functions traced as well as node observations and possible rank order suggestions.
- `gpu` — Tracing experiments that focus on the program's use of GPU accelerators.

To switch from using CrayPat-lite to using the full CrayPat tool set, unload the `perftools-lite` module and load the `perftools` module.

For more information about CrayPat-lite, see *Using Cray Performance Measurement and Analysis Tools*.

8.3.2 About CrayPat

CrayPat is the full-featured performance analysis tool set, and consists of three major components:

- *pat_build*, which is the utility used to instrument programs for data capture
- the *CrayPat run time environment*, which controls the conditions under which the program executes and the amounts and types of data captured
- *pat_report*, which is the utility used to generate reports from the resulting captured data

To begin working with the performance analysis tools, first load your programming environment of choice, and then load the `perftools` module.

```
users/yourname> module load perftools
```

For successful results, the `perftools` module must be loaded before you compile the program to be instrumented, instrument the program, execute the instrumented program, or generate a report. If you want to instrument a program that was compiled before the `perftools` module was loaded, you may under some circumstances find that relinking is sufficient, but as a rule it's best to load the `perftools` module and then recompile.

8.3.2.1 Instrumenting the Program

After the program is compiled and linked, use the `pat_build` command to instrument the program for performance analysis. In simplest form, `pat_build` is used like this:

```
> pat_build executable
```

This produces a copy of your original program, which is named *executable*`+pat` (for example, `a.out+pat`) and instrumented for the default experiment, Automatic Profiling Analysis. Your original executable remains untouched.

The `pat_build` command supports a large number of options and directives, including an API that enables you to instrument specified regions of your code. These options and directives are summarized in the `pat_build`(1) man page and documented more extensively in *Using Cray Performance Measurement and Analysis Tools*.

8.3.2.2 Collecting Data

Instrumented programs are executed just like any other program; either by using the `aprun` command if your site permits interactive sessions or by using your system's batch commands.

CrayPat supports more than fifty optional run time environment variables that enable you to control instrumented program behavior and data collection during execution. For example, if you use the C shell and want to collect data in detail rather than in aggregate, consider setting the `PAT_RT_SUMMARY` environment variable to `0` (off) before launching your program.

```
/lus/nid00008> setenv PAT_RT_SUMMARY 0
```

Doing so records data with timestamps, which makes additional reports available in Cray Apprentice2, but at the cost of potentially much larger data file sizes and somewhat increased overhead.

The CrayPat run time environment variables are summarized in the `intro_craypat`(1) man page and documented more extensively in *Using Cray Performance Measurement and Analysis Tools*.

8.3.2.3 Analyzing Data

Assuming your instrumented program runs to completion or planned termination, CrayPat outputs one or more data files. The exact number, location, and content of the data file(s) varies depending on the nature of your program, the type of experiment for which it was instrumented, and the run time environment variable settings in effect at the time of program execution.

All initial data files are output in `.xf` format, with a generated file name consisting of your original program name, plus `pat`, plus the execution process ID number, plus a code string indicating the type of data contained within the file. Depending on the program run and the types of data collected, CrayPat output may consist of either a single `.xf` data file or a directory containing multiple `.xf` data files. If the program was instrumented for Automatic Profiling Analysis, a file with the suffix `.apa` is also generated. This file is a customized template for this program and is created for use with future instrumentation experiments.

To begin analyzing the captured data, use the `pat_report` command. In simplest form, it looks like this:

```
/lus/nid00008> pat_report myprog+pat+PID-nodes.xf
```

The `pat_report` command accepts either a file or directory name as input and processes the `.xf` file(s) to generate a text report. In addition, it also exports the `.xf` data to a single `.ap2` file, which is both a self-contained archive that can be reopened later using the `pat_report` command and the exported-data file format used by Cray Apprentice2.

The `pat_report` command provides more than thirty predefined report templates, as well as a large variety of user-configurable options. These reports and options are summarized in the `pat_report`(1) man page and documented more extensively in *Using Cray Performance Measurement and Analysis Tools*.

8.3.2.4 For More Information

In addition to *Using Cray Performance Measurement and Analysis Tools* and the `intro_craypat`(1), `pat_build`(1), and `pat_report`(1) man pages, there is a substantial amount of information, including an FAQ and examples, in the CrayPat online help system. The help system is accessible whenever the `perftools` module is loaded; to access the help system, enter `pat_help` at the command line. For more information about using the help system, see the `pat_help`(1) man page.

8.3.3 About Cray Apprentice2

Cray Apprentice2 is an optional GUI tool that is used to visualize and manipulate the performance analysis data captured during program execution. Cray Apprentice2 can be run either on the Cray system or, optionally, on a standalone Linux desktop machine. Cray Apprentice2 can display a wide variety of reports and graphs, depending on the type of program being analyzed, the way in which the program was instrumented for data capture, and the data that was collected during program execution.

Cray Apprentice2 is not directly integrated with CrayPat. You cannot launch Cray Apprentice2 from within CrayPat, nor can you set up or run performance analysis experiments from within Cray Apprentice2. Rather, use CrayPat first, to instrument your program and capture performance analysis data, and then use Cray Apprentice2 to visualize and explore the resulting data files.

The number and appearance of the reports that can be generated using Cray Apprentice2 is determined by the kind and quantity of data captured during program execution, which in turn is determined by the way in which the program was instrumented and the environment variables in effect at the time of program execution. For example, changing the `PAT_RT_SUMMARY` environment variable to `0` before executing the instrumented program nearly doubles the number of reports available when analyzing the resulting data in Cray Apprentice2.

To run Cray Apprentice2, load the `perftools` module, if it is not already loaded.

```
users/yourname> module load perftools
```

Then use the `app2` command to launch Cray Apprentice2.

```
users/yourname> app2 [datafile.ap2] &
```

Note: Cray Apprentice2 requires that your workstation be configured to host X Window System sessions. If the `app2` command returns an `"unable to open display"` error, contact your system administrator for help in configuring X Window System hosting and forwarding.

At this point the GUI takes over. If you specified a data file name with the `app2` command, the file is opened and parsed and the **Overview** report is displayed. If you did not specify a data file name, the **Open File** window opens and you can use standard GUI tools to browse through the file system and select the data file you want to open.

For more information about using Cray Apprentice2, see the `app2`(1) man page, the Cray Apprentice2 help system, and *Using Cray Performance Measurement and Analysis Tools*.

8.3.4 About Reveal

Reveal is Cray's next-generation integrated performance analysis and code optimization tool. Reveal extend's Cray's existing performance measurement, analysis, and visualization technology by combining run time performance statistics and program source code visualization with Cray Compiling Environment (CCE) compile-time optimization feedback.

Note: Reveal requires use of CCE, and therefore is not supported on Intel Xeon Phi coprocessor systems at this time.

Reveal supports source code navigation using whole-program analysis data and program libraries provided by the Cray Compiling Environment, coupled with performance data collected during program execution by the Cray performance tools, to understand which high-level serial loops could benefit from improved parallelism. Reveal provides enhanced loopmark listing functionality, dependency information for targeted loops, and assists users optimizing code by providing variable scoping feedback and suggested compiler directives.

To begin using Reveal on the Cray system, verify that the `perftools` module is loaded:

```
> module load perftools
```

Launch the Reveal application using the `reveal` command:

```
> reveal
```

Note: Reveal requires that your workstation be configured to host X Window System sessions. If the `reveal` command returns an "`cannot open display`" error, contact your system administrator for help in configuring X Window System hosting.

You can specify data files to be opened when you launch Reveal. For example, this command launches Reveal and opens both the compiler-generated program library file and the CrayPat-generated run time performance data file, thus enabling you to correlate performance data captured during program execution with specific lines and loops in the original source code:

```
> reveal my_program_library.pl my_performance datafile.ap2
```

Alternately, Reveal opens a file selection window and you can then select the data file(s) you want to open.

For more information about using the `reveal` command, see the `reveal`(1) man page.

8.3.5 About PAPI

CrayPat uses PAPI, the Performance API. This interface is normally transparent to the user. However, if you want more information about PAPI, see the `intro_papi`(3) and `papi_counters`(5) man pages.

Note: To access PAPI functions and utilities directly, you must first unload the `perftools` module and then load the `papi` module. However, after you do so, Cray-originated man pages, the `pat_help` command, and the `$CRAYPAT_ROOT` path will be unavailable. Therefore, if you plan to use utilities or develop programs that use PAPI directly, plan your work and save your reference information accordingly.

Additional information about using PAPI is available through the PAPI website, at http://icl.cs.utk.edu/papi/.

glibc Functions [A]

The supported glibc functions and system calls are listed in Table 23. For further information, see the man pages.

Note: Some `fcntl()` commands are not supported for applications that use Lustre. The supported commands are:

- `F_GETFL`
- `F_SETFL`
- `F_GETLK`
- `F_SETLK`
- `F_SETLKW64`
- `F_SETLKW`
- `F_SETLK64`

Also, asynchronous I/O (aio) calls are not supported for applications that use Lustre.

Table 23. Supported glibc Functions

a64l	abort	abs	access
addmntent	alarm	alphasort	argz_add
argz_add_sep	argz_append	argz_count	argz_create
argz_create_sep	argz_delete	argz_extract	argz_insert
argz_next	argz_replace	argz_stringify	asctime
asctime_r	asprintf	atexit	atof
atoi	atol	atoll	basename
bcmp	bcopy	bind_textdomain_codeset	bindtextdomain
bsearch	btowc	bzero	calloc
catclose	catgets	catopen	cbc_crypt
chdir	chmod	chown	clearenv
clearerr	clearerr_unlocked	close	closedir

confstr
copysign
copysignf
copysignl
creat
ctime
ctime_r
daemon
daylight
dcgettext
dcngettext
des_setparity
dgettext
difftime
dirfd
dirname
div
dngettext
dprintf
drand48
dup
dup2
dysize
ecb_crypt
ecvt
ecvt_r
endfsent
endmntent
endttyent
endusershell
envz_add
envz_entry
envz_get
envz_merge
envz_remove
envz_strip
erand48
err
errx
exit
fchmod
fchown
fclose
fcloseall
fcntl
fcvt
fcvt_r
fdatasync
fdopen
feof
feof_unlocked
ferror
ferror_unlocked
fflush
fflush_unlocked
ffs
ffsl
ffsll
fgetc
fgetc_unlocked
fgetgrent
fgetpos
fgetpwent
fgets
fgets_unlocked
fgetwc
fgetwc_unlocked
fgetws
fgetws_unlocked
fileno
fileno_unlocked
finite
flock
flockfile
fnmatch
fopen
fprintf
fputc
fputc_unlocked
fputs
fputs_unlocked
fputwc
fputwc_unlocked
fputws
fputws_unlocked
fread
fread_unlocked
free
freopen
frexp
fscanf
fseek
fseeko
fsetpos
fstat
fsync
ftell
ftello
ftime
ftok
ftruncate
ftrylockfile
funlockfile
fwide
fwprintf
fwrite
fwrite_unlocked
gcvt
get_current_dir_name
getc
getc_unlocked
getchar
getchar_unlocked
getcwd
getdate
getdate_r
getdelim
getdirentries
getdomainname
getegid
getenv
geteuid
getfsent
getfsfile
getfsspec
getgid
gethostname
getline
getlogin
getlogin_r
getmntent
getopt
getopt_long

getopt_long_only
getpagesize
getpass
getpid
getprotoent
getprotobyname
getprotobynumber
getrlimit
getrusage
gettext
gettimeofday
getttyent
getttynam
getuid
getusershell
getw
getwc
getwc_unlocked
getwchar
getwchar_unlocked
gmtime
gmtime_r
gsignal
hasmntopt
hcreate
hcreate_r
hdestroy
hsearch
iconv
iconv_close
iconv_open
imaxabs
index
initstate
insque
ioctl
isalnum
isalpha
isascii
isblank
iscntrl
isdigit
isgraph
isinf
islower
isnan
isprint
ispunct
isspace
isupper
iswalnum
iswalpha
iswblank
iswcntrl
iswctype
iswdigit
iswgraph
iswlower
iswprint
iswpunct
iswspace
iswupper
iswxdigit
isxdigit
jrand48
kill
l64a
labs
lcong48
ldexp
lfind
link
llabs
localeconv
localtime
localtime_r
lockf
longjmp
lrand48
lsearch
lseek
lstat
malloc
mblen
mbrlen
mbrtowc
mbsinit
mbsnrtowcs
mbsrtowcs
mbstowcs
mbtowc
memccpy
memchr
memcmp
memcpy
memfrob
memmem
memmove
memrchr
memset
mkdir
mkdtemp
mknod
mkstemp
mktime
modf
modff
modfl
mrand48
nanosleep
ngettext
nl_langinfo
nrand48
on_exit
open
opendir
passwd2des
pclose
perror
pread
printf
psignal
putc
putc_unlocked
putchar
putchar_unlocked
putenv
putpwent
puts
putw
putwc
putwc_unlocked
putwchar
putwchar_unlocked
pwrite

qecvt
qecvt_r
qfcvt
qfcvt_r
qgcvt
qsort
raise
rand
random
re_comp
re_exec
read
readdir
readlink
readv
realloc
realpath
regcomp
regerror
regexec
regfree
registerrpc
remove
remque
rename
rewind
rewinddir
rindex
rmdir
scandir
scanf
seed48
seekdir
setbuf
setbuffer
setegid
setenv
seteuid
setfsent
setgid
setitimer
setjmp
setlinebuf
setlocale
setlogmask
setmntent
setrlimit
setstate
setttyent
setuid
setusershell
setvbuf
sigaction
sigaddset
sigdelset
sigemptyset
sigfillset
sigismember
siglongjmp
signal
sigpending
sigprocmask
sigsuspend
sleep
snprintf
sprintf
srand
srand48
srandom
sscanf
ssignal
stat
stpcpy
stpncpy
strcasecmp
strcat
strchr
strcmp
strcoll
strcpy
strcspn
strdup
strerror
strerror_r
strfmon
strfry
strftime
strlen
strncasecmp
strncat
strncmp
strncpy
strndup
strnlen
strpbrk
strptime
strrchr
strsep
strsignal
strspn
strstr
strtod
strtof
strtok
strtok_r
strtol
strtold
strtoll
strtoq
strtoul
strtoull
strtouq
strverscmp
strxfrm
svcfd_create
swab
swprintf
symlink
syscall
sysconf
tdelete
telldir
textdomain
tfind
time
timegm
timelocal
timezone
tmpfile
toascii
tolower
toupper
towctrans
towlower
towupper

truncate	tsearch	ttyslot	twalk
tzname	tzset	umask	umount
uname	ungetc	ungetwc	unlink
unsetenv	usleep	utime	vasprintf
vdprintf	verr	verrx	versionsort
vfork	vfprintf	vfscanf	vfwprintf
vprintf	vscanf	vsnprintf	vsprintf
vsscanf	vswprintf	vwarn	vwarnx
vwprintf	warn	warnx	wcpcpy
wcpncpy	wcrtomb	wcscasecmp	wcscat
wcschr	wcscmp	wcscpy	wcscspn
wcsdup	wcslen	wcsncasecmp	wcsncat
wcsncmp	wcsncpy	wcsnlen	wcsnrtombs
wcspbrk	wcsrchr	wcsrtombs	wcsspn
wcsstr	wcstok	wcstombs	wcswidth
wctob	wctomb	wctrans	wctype
wcwidth	wmemchr	wmemcmp	wmemcpy
wmemmove	wmemset	wprintf	write
writev	xdecrypt	xencrypt	