# Cray Application Developer's Environment User's Guide

S–2396–601

RECORD OF REVISION

S–2396–601 Published August 2011 Supports general availability (GA) release of Cray XT, Cray XE, and Cray XK systems running Cray Linux Environment (CLE) release 3.0 or later.

S–2396–60 Published July 2011 Supports general availability (GA) release of Cray XT, Cray XE, and Cray XK systems running Cray Linux Environment (CLE) release 3.0 or later.

5.0 Published June 2010 Restructured document now packaged with Cray Application Developer's Environment (CADE) release 5.0 and supporting Cray XT and Cray XE systems running Cray Linux Environment (CLE) release 2.2 or later.

2.2 Published July 2009 Supports general availability (GA) release of Cray XT systems running the Cray XT Programming Environment and CLE 2.2 releases.

2.1 Published November 2008 Supports general availability (GA) release of Cray XT systems running the Cray XT Programming Environment and CLE 2.1 releases.

2.1 Published July 2008 Supports limited availability (LA) release of Cray XT systems running the Cray XT Programming Environment and CLE 2.1 releases.

2.0 Published October 2007 Supports general availability (GA) release of Cray XT systems running the Cray XT Programming Environment 2.0 and UNICOS/lc 2.0 releases.

2.0 Published June 2007 Supports limited availability (LA) release of Cray XT systems running the Cray XT Programming Environment 2.0 and UNICOS/lc 2.0 releases.

1.5 Published November 2006 Supports general availability (GA) release of Cray XT systems running the Cray XT Programming Environment 1.5 and UNICOS/lc 1.5 releases.

1.5 Published August 2006 Supports limited availability (LA) release of Cray XT systems running the Cray XT Programming Environment 1.5 and Cray Linux Environment (CLE) 1.5 releases.

1.4 Published April 2006 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.4 and UNICOS/lc 1.4 releases.

1.3 Published November 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.3 and UNICOS/lc 1.3 releases.

1.2 Published August 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.2 and UNICOS/lc 1.2 releases.

1.1 Published June 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.1 and UNICOS/lc 1.1 releases.

1.0 Published March 2005 Draft documentation to support Cray XT3 limited-availability systems.

1.0 Published December 2004 Draft documentation to support Cray XT3 early-production systems.

# Changes to this Document

*Cray Application Developer's Environment User's Guide* S–2396–601

This rewrite of the *Cray Application Developer's Environment User's Guide* supports the 6.01 and later releases of the Cray Application Developer's Environment (CADE).

**S–2396–601**

Added information

- Chapter 7, formerly "Performance Analysis," is renamed Chapter 7, Optimizing Code on page 73, and has been expanded to include information about Using `iobuf` on page 73, Improving MPI I/O on page 74, and Using Compiler Optimizations on page 75.

Revised information

- Minor typographical corrections have been made throughout this guide. In particular, coding errors which caused invalid links to appear in the HTML version of this guide have been corrected.

**S–2396–60**

Added information

- Preliminary information regarding support for Cray XK systems has been added throughout this guide.
- Preliminary information regarding support for AMD Interlagos processors has been added through this guide.
- Use of Cray CPU and network-type targeting modules on Cray systems is described in Using Targeting Modules on page 27. Use of targeting modules on standalone Linux workstations is described in About Cross-compilers on page 43.
- A new library of eigensolvers, the Cray Adaptive Simple Eigensolvers (CASE), has been added. For more information, see Cray Adaptive Simple Eigensolvers (CASE) on page 52.
- The default behavior of MPI and SHMEM modules has been changed significantly. For more information, see MPT on page 59.
- Abnormal Termination Processing (ATP) is now enabled by default. For more information, see Using Abnormal Termination Processing (ATP) on page 65.

Revised information

- Information regarding the use of hugepages has been revised and expanded. For more information, see Hugepages on page 61.
- Beginning with Cray Scientific Libraries Release 11.0.00, the way in which dynamic libraries are found at execution time has changed. For more information, see New in LibSci Release 11.0 on page 47.
- Beginning with PETSc Release 3.1.08, the Third-Party Scientific Libraries (TPSL) module is loaded automatically along with the PETSc module. This change affects both PETSc and Trilinos, and is described in PETSc on page 56 and Trilinos on page 58.

Deleted information

- Support for Cray Linux Environment (CLE) 2.1 is discontinued. Therefore all references to CLE 2.1 have been removed from this guide.
- Beginning with Cray Scientific Libraries Release 11.0.00, Cray no longer provides a version of LibSci optimized to support the PathScale Compiler Suite.

# Contents

*Page*

## Figures

## Tables

# Introduction [1]

This guide describes the software environment and tools used to develop, debug, and run applications on Cray XT, Cray XE, and Cray XK 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 considerably 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. System hardware and software configurations therefore vary considerably from site to site, so for specific information about your site and its installed hardware, software, and usage policies, always contact your site administrator.

## 1.1 What You Must Know About Your Site

The Cray XT, Cray XE, and Cray XK systems are:

- massively parallel (MPP)
- distributed memory
- non-uniform memory access (NUMA)
- commodity processor-based supercomputers

These systems are designed to scale to immense size and run applications requiring high-performance, large-scale processing, high network bandwidth, and complex inter-processor communications. All systems use 64-bit AMD Opteron processors as the basic computational engines, although the exact number of cores per AMD Opteron varies from site to site and sometimes even from cabinet to cabinet, depending on your site's installation, expansion, and upgrade history.

### 1.1.1 Cray XT, Cray XE, or Cray XK?

From the programmer's point of view, the most significant differences between the systems are:

- Cray XT systems use SeaStar or SeaStar2+ application-specific integrated circuits (ASICs) to manage inter-processor communications
- Cray XE systems use Gemini ASICs to manage inter-processor communications
- Cray XK systems also use Gemini ASICs for inter-processor communications, but combine AMD Interlagos CPUs and NVIDIA Tesla GPUs on compute nodes

Because of the differences in the network ASICs and accompanying network APIs, Cray XT and Cray XE systems use different versions of the MPI and SHMEM libraries, which offer different sets of optional controls to the application developer. Cray XK systems use the same MPI and SHMEM libraries as Cray XE systems.

The differences between the versions of MPI and SHMEM are discussed in more detail in MPT on page 59.

Cray XE and Cray XK systems support the Generic Network Interface (GNI) and Distributed Shared Memory Application (DMAPP) APIs, which offer the programmer API-level access to the advanced features of the Gemini network interconnect. The Gemini GNI and DMAPP APIs are discussed in detail in *Using the GNI and DMAPP APIs* (S–2446).

### 1.1.2 How Many Cores?

There are significant differences are between models.

- Cray XT4 systems use one dual- or quad-core ("Barcelona") AMD Opteron CPU per compute node
- Cray XT5 and Cray XE5 systems use two four-core ("Shanghai") or six-core ("Istanbul") AMD Opteron CPUs per compute node
- Cray XT6 and Cray XE6 systems use two eight- or twelve-core ("Magny-Cours") or two sixteen-core ("Interlagos") AMD Opteron CPUs per compute node
- Cray XK systems use one sixteen-core ("Interlagos") AMD Opteron CPU and one NVIDIA Tesla GPU per compute node

Because of these hardware differences, you can invoke different options when compiling and executing your programs. These differences are discussed in more detail in *Workload Management and Application Placement for the Cray Linux Environment* (S–2496).

## 1.1.3 Which Operating System?

All current Cray systems run the Cray Linux Environment (CLE) operating system on the login nodes and a lightweight kernel called 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 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.
- All other systems may be running CLE release 2.2, 3.0, 3.1, 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 two other ways to determine the CLE release number.

- On CLE 2.*x* systems, check to see which version of the `xt-os` module is loaded by default. (For more information about using `modules`, see Chapter 2, Using Modules on page 23.) If no `xt-os` module is present, your system is running CLE release 3.0 or later.
- 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. If this file or directory does not exist, your system is running CLE release 2.*x*.

**Note:** CLE 2.*x* is based on SLES 10. CLE 3.*x* and CLE 4.*x* are based on SLES 11.

### 1.1.4 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, all specialized for different purposes. While some are dedicated to administrative or networking functions and therefore off-limits to application programmers, the two you will deal with most often are:

- *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*.

- *compute nodes* — The nodes on which production jobs are executed. Compute nodes run a lightweight operating system called Compute Node Linux (CNL), can be accessed only by submitting jobs through the batch management system (typically PBS Pro, Moab/TORQUE, or Platform LSF), generally have access only to the high-performance parallel file system (typically Lustre or Panasas), 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 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.

### 1.1.5 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 such as the Panasas PanFS 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. Then if required, adjust the instructions accordingly.

### 1.1.6 Which Batch System?

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

### 1.1.7 What is the *hostname*?

All Cray systems have a unique *hostname*. While it is possible to log in to a given system by using its IP address (e.g., `127.0.0.1`), it is easier if you use the *hostname*.

The form of the *hostname* varies from site to site. Some sites use a single word; e.g., `narwhal`. Others use a fully qualified URL; e.g., `narwhal.univ-alaska-barrow.edu`. You should be told which name and form to use when your user account is set up.

## 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 (such as, say, the correct use of an RSA SecurID token, if provided) 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 forward 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 -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>
```

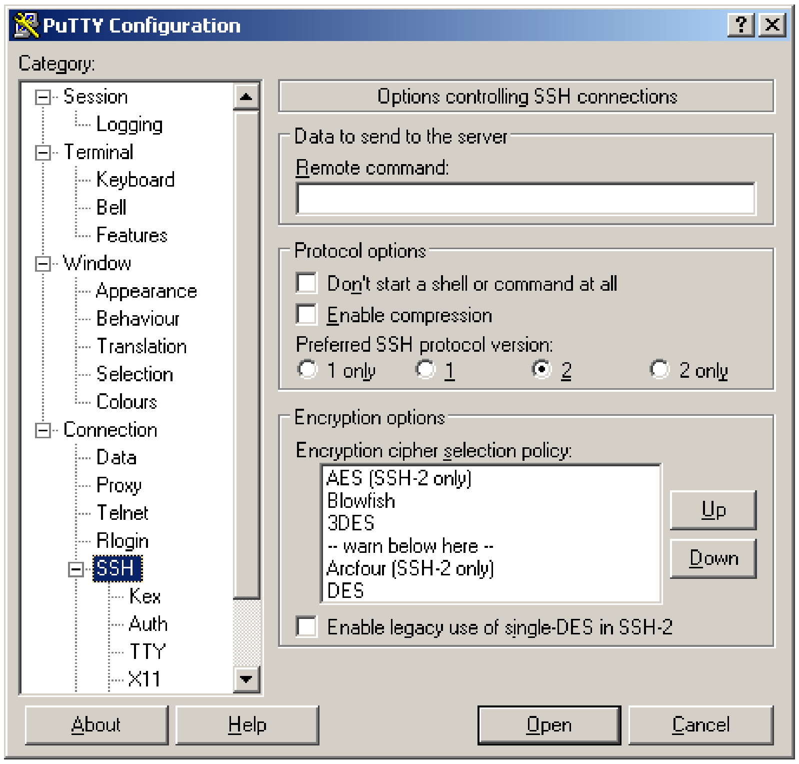
You are now ready to begin working. Jump to Navigating the File Systems on page 20.

## 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**



Options controlling SSH connections

Data to send to the server

Remote command:

Protocol options

- [ ] Don't start a shell or command at all
- [ ] Enable compression

Preferred SSH protocol version: 1 only 1 [✓ 2] 2 only

Encryption options

Encryption cipher selection policy:

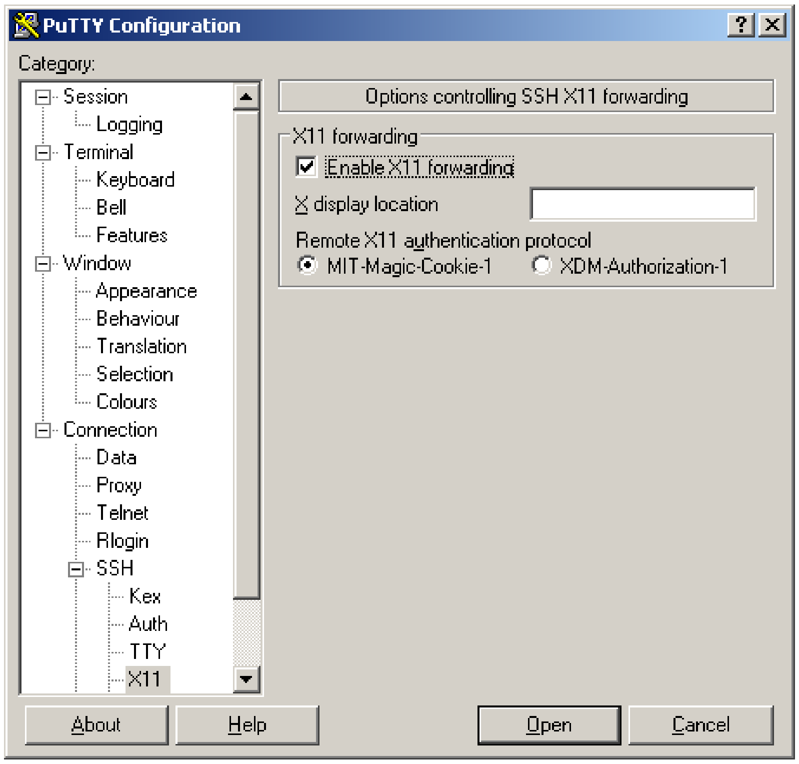
AES (SSH-2 only)
Blowfish
3DES
-- warn below here --
Arcfour (SSH-2 only)
DES

- [ ] Enable legacy use of single-DES in SSH-2

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**



Options controlling SSH X11 forwarding

X11 forwarding

[✓] Enable X11 forwarding

X display location

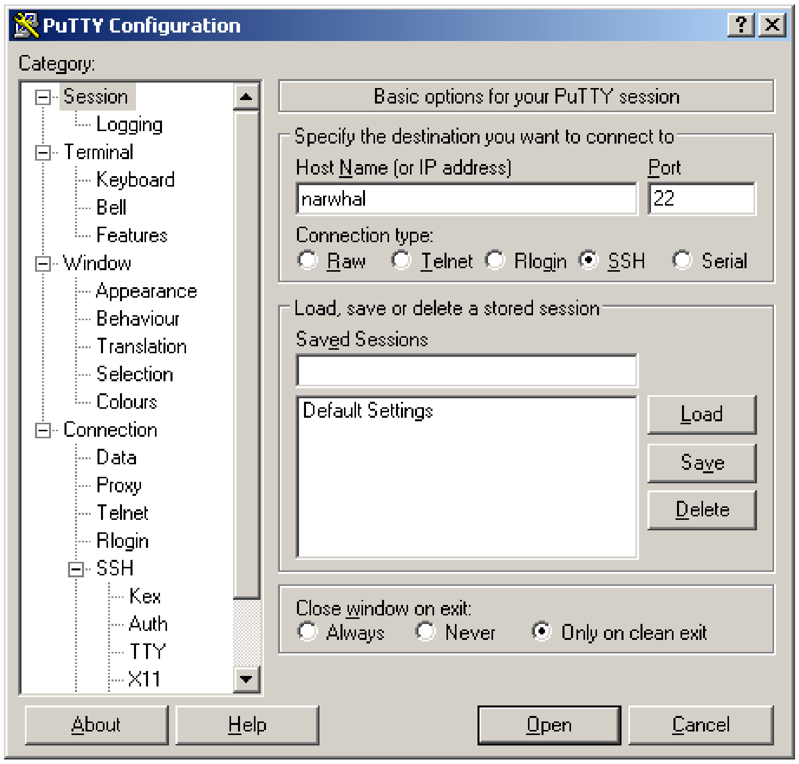
Remote X11 authentication protocol

[● MIT-Magic-Cookie-1] XDM-Authorization-1

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**



Host Name (or IP address): narwhal
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Close window on exit: Always Never [✓ Only on clean exit]

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 Macintosh Users

The Apple Macintosh 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 forward 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, and also typically 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 either case, 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 files, manipulate files, compile code, and so on, but you can also execute programs on the compute nodes, generally 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 all the 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 or a different version of a library, 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.

In general, the simplest way to make certain that the elements of your application development environment function correctly together is by using the Modules software and trusting it to keep track of paths and environment variables, and avoiding 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/3.2.6.6
  2) nodestat/2.2-1.0400.25564.5.2.gem
  3) sdb/1.0-1.0400.27358.12.17.gem
  4) MySQL/5.0.64-1.0000.2899.19.2
  5) lustre-cray_gem_s/1.8.4_2.6.32.36_0.5.2_1.0400.5941.3.1-1.0400.27356.3.114
  6) udreg/2.3.1-1.0400.3052.6.12.gem
  7) ugni/2.2-1.0400.3340.9.29.gem
  8) gni-headers/2.1-1.0400.3411.8.1.gem
  9) dmapp/3.1-1.0400.3387.13.25.gem
 10) xpmem/0.1-2.0400.27172.6.5.gem
 11) Base-opts/1.0.2-1.0400.25719.5.1.gem
 12) xtpe-network-gemini
 13) pgi/11.6.0
 14) totalview-support/1.1.2
 15) xt-totalview/8.9.1
 16) xt-libsci/10.5.02
 17) pmi/2.1.2-1.0000.8396.13.5.gem
 18) xt-asyncpe/5.00.59
 19) atp/1.2.0
 20) PrgEnv-pgi/4.0.12A
 21) pbs/10.4.0.101257
 22) xtpe-mc12
 23) xt-mpich2/5.3.0
```

This list breaks down into three groups: operating system modules, programming environment modules, and support modules. For example, the `xtpe-mc12` module indicates that this development environment is set up to develop code for use on 12-core Magny-Cours processors, while the `PrgEnv-pgi` module indicates that PGI compiler suite is currently loaded.

## 2.2 What is Available?

To see which modulefiles are available on your system, enter this 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 specific version of the PGI compiler suite, you would enter this command:

```
users/yourname> module avail PrgEnv-pgi

------------------------------------ /opt/modulefiles ------------------------------------
PrgEnv-pgi/3.1.35           PrgEnv-pgi/3.1.37E          PrgEnv-pgi/3.1.61
PrgEnv-pgi/3.1.37AA         PrgEnv-pgi/3.1.37G          PrgEnv-pgi/4.0.12A(default)
PrgEnv-pgi/3.1.37C          PrgEnv-pgi/3.1.49A
```

One module is usually designated as the default version. Whether this is the most recent version of this module depends on your site 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 5, Libraries on page 47) |
| `-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 modulefile is **not** already loaded, use the `module load` command to load it.

```
users/yourname> module load modulefile
```

This command loads the currently defined default version of the module, unless you specify otherwise. For example, if the modules listed in What is Available? on page 24 are present on your system but `PrgEnv-pgi` is not already loaded, then:

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

loads `PrgEnv-pgi/4.0.12A(default)` module. To load a non-default version of the module, you must specify the full module name.

```
users/yourname> module load PrgEnv-pgi/3.1.61
```

However, if a module **is** already loaded, then you must first unload the currently loaded module before loading a different version. For example, to change from the default to version of the PGI compiler suite to another version, use the `module unload` command to remove the version currently loaded.

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

Modules may be linked and related. If you enter the `module list` command now and compare the results to those shown in What is Loaded Now? on page 24:

```
users/yourname> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.6
  2) nodestat/2.2-1.0400.25564.5.2.gem
  3) sdb/1.0-1.0400.27358.12.17.gem
  4) MySQL/5.0.64-1.0000.2899.19.2
  5) lustre-cray_gem_s/1.8.4_2.6.32.36_0.5.2_1.0400.5941.3.1-1.0400.27356.3.114
  6) udreg/2.3.1-1.0400.3052.6.12.gem
  7) ugni/2.2-1.0400.3340.9.29.gem
  8) gni-headers/2.1-1.0400.3411.8.1.gem
  9) dmapp/3.1-1.0400.3387.13.25.gem
 10) xpmem/0.1-2.0400.27172.6.5.gem
 11) Base-opts/1.0.2-1.0400.25719.5.1.gem
 12) xtpe-network-gemini
 13) pbs/10.4.0.101257
 14) xtpe-mc12
 15) xt-mpich2/5.3.0
```

You can see that along with `PrgEnv-pgi`, the `pgi`, `totalview-support`, `xt-totalview`, `xt-libsci`, `pmi`, `xt-asyncpe`, and `atp` modules were also unloaded. If you now load a different version of the `PrgEnv-pgi` module:

```
users/yourname> module load PrgEnv-pgi/3.1.61
```

Then use the `module list` command again, you can see that the associated modules have been reloaded automatically, but this time using the versions that are appropriate for use with the 3.1.61 version of the PGI compiler.

## 2.4 Swapping 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 using the PGI Compiler Suite to the Cray Compiling Environment, enter this command:

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

If you then list the modules that are loaded after this swap and compare them to the results shown in What is Loaded Now? on page 24.

```
users/yourname> module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.6
  2) nodestat/2.2-1.0400.25564.5.2.gem
  3) sdb/1.0-1.0400.27358.12.17.gem
  4) MySQL/5.0.64-1.0000.2899.19.2
  5) lustre-cray_gem_s/1.8.4_2.6.32.36_0.5.2_1.0400.5941.3.1-1.0400.27356.3.114
  6) udreg/2.3.1-1.0400.3052.6.12.gem
  7) ugni/2.2-1.0400.3340.9.29.gem
  8) gni-headers/2.1-1.0400.3411.8.1.gem
  9) dmapp/3.1-1.0400.3387.13.25.gem
 10) xpmem/0.1-2.0400.27172.6.5.gem
 11) Base-opts/1.0.2-1.0400.25719.5.1.gem
 12) xtpe-network-gemini
 13) pbs/10.4.0.101257
 14) xtpe-mc12
 15) PrgEnv-cray/4.0.12A
 16) atp/1.2.0
 17) xt-asyncpe/5.00.59
 18) rca/1.0.0-2.0400.27173.6.23.gem
 19) pmi/2.1.2-1.0000.8396.13.5.gem
 20) xt-libsci/10.5.02
 21) acml/4.4.0
 22) xt-totalview/8.9.1
 23) totalview-support/1.1.2
 24) cce/7.4.1.103
```

You can see that a different set of supporting modules have been loaded automatically.

## 2.5 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 and network-type modules currently available.

```
---------------------------- /opt/cray/xt-asyncpe/default/modulefiles ----------------------------
xtpe-barcelona       xtpe-mc12            xtpe-network-seastar
xtpe-interlagos      xtpe-mc8             xtpe-shanghai
xtpe-istanbul        xtpe-network-gemini  xtpe-target-native
```

If you are working on a Cray system, your default environment should load the CPU and network-type modules that are correct for your hardware. For example, if you have a Cray XE6 system with eight-core Magny-Cours compute nodes, your default environment should include the `xtpe-network-gemini` and `xtpe-mc8` modules, and no others. If the correct modules are loaded, do not change them.

However, if you are working on a standalone Linux workstation and developing executable code which 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 `xtpe-network-seastar` module instead of the `xtpe-network-gemini` module loaded, will not run correctly on a Cray XE system.

For more information about using standalone Linux workstations to develop code, see About Cross-compilers on page 43.

**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* (S–2496).

**Note:** The `xtpe-target-native` module is not a CPU or network-type module and used only in limited and special cases. For more information, see Bypassing the Compiler Drivers on page 38.

## 2.6 Release Notes and 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 usage information received too late to be documented elsewhere.

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 PGI module, enter this command:

```
users/yourname> module help pgi
```

**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-pgi` and `module help pgi` display different information.

## 2.7 For More Information

The Modules commands are documented in the `module`(1) and `modulefiles`(4) man pages. A summary of Modules commands 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 on 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]

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* (S–2496).

At most sites, access to the compute node resources is managed by a batch control system, typically PBS Pro, Moab/TORQUE, 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.

In either case, 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 (i.e., `qsub`) and `aprun` command options, which are not necessarily the same. If running a batch job, modify your script accordingly.
4. Use the batch command either to submit your job script or to request an interactive session.
5. If using an interactive session, use the `aprun` command to launch your application.

## 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 -IVl mppwidth=cores
```

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_type*=*specification*

Specify the resources you want to reserve for this session. For example, you can use this option to reserve 24 compute cores for one hour.

```
-l mppwidth=24,walltime=1:00:00
```

The `-l` *resource_type*=*specification* argument supports a large number of options which 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* (S–2496).

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 normal command line.

### 3.1.1 Notes

- You must use the `-l mppwidth=` option to specify at least one core when you start the interactive session. If you do not, your request for an interactive session will pause indefinitely.
- 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/nid00008/`*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, which typically is on a login node. (For example, `/ufs/users/home/`*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 35 and described in detail in *Workload Management and Application Placement for the Cray Linux Environment* (S–2496).
- 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.

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

```
users/yourname> qsub [-l resource_type=specification] jobscript
```

The `[-l` *resource_type=specification*] arguments are described in the `pbs_resources`(7B) man page.

A typical batch script might look like this:

```
#!/bin/bash
#PBS -A account
#PBS -N job_name
#PBS -j oe
#PBS -l walltime=1:00:00,mppwidth=192

cd $PBS_O_WORKDIR
date
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 a 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:

```
% 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` equivalents.

**Table 1. `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 1000` | `-l mppmem=1000` | 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* (S–2496). 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.

# 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++)
- PathScale Inc., PathScale Compiler Suite (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. The different compilers have different strengths and weaknesses, and their relative strengths and weaknesses are constantly changing as new revisions and updates are released.

## 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 compiler.

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, `xtpe-target-native`, and then continue to use the `ftn`, `cc`, and `CC` commands as before. The `xtpe-target-native` module enables the compiler driver commands to function as native compiler commands—for example, if the PGI programming 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 `xtpe-target-native` module.

## 4.2 About C/C++ Floating Point 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 2. C/C++ Data Type Sizes**

| Data Type | Size in Bytes |
|---|---|
| unsigned char | 1 |
| signed char | 1 |
| unsigned short | 2 |
| signed short | 2 |

| **Data Type** | **Size in Bytes** |
|---|---|
| unsigned int | 4 |
| signed int | 4 |
| unsigned long | 8 |
| signed long | 8 |
| unsigned long long | 8 |
| signed long long | 8 |
| float | 4 |
| double | 8 |
| long double | 8 or 16, depending on compiler |
| char * | 8 |
| enum | 4 |

## 4.3 About the Cray Compiling Environment (CCE)

**Table 3. 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.

## 4.4 About PGI Compilers

**Table 4. PGI Compiler Basics**

| | |
|---|---|
| Module: | `PrgEnv-pgi` |
| Command: | `ftn`, `cc`, `CC` |
| Compiler-specific man pages: | `pgf95`(1), `pgcc`(1), `pgCC`(1)<br>**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 produce code optimized for AMD Interlagos processors. Code compiled with the PGI compilers can be expected to suffer performance degradation if executed on compute nodes with Interlagos CPUs. This limitation is expected to be removed in a future release.
- 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 5. 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.

## 4.6 About PathScale Compilers

**Table 6. PathScale Compiler Basics**

| | |
|---|---|
| Module: | `PrgEnv-pathscale` |
| Command: | `ftn`, `cc`, `CC` |
| Compiler-specific man pages: | `pathf95`(1), `pathf90`(1), `pathcc`(1), `pathCC`(1), `eko`(7) |
| | **Note:** Compiler-specific man pages are available only when the compiler module is loaded. |
| Online help: | `pathf95 --help`, `pathf90 --help`, `pathcc --help`, `pathCC --help` |
| Documentation: | `/opt/pathscale/share/doc/`*version* |

### 4.6.1 Known Limitations

- The PathScale Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.
- The PETSc 3.1.0.8 library of parallel linear and nonlinear solvers does not support the PathScale Compiler Suite. Use PETSc 3.0.*x* instead.
- Beginning with PathScale release 4.0.9, Cray no longer provides PathScale-specific versions of the Cray Scientific Libraries (LibSci), or of PETSc, Trilinos, or other third-party scientific libraries (TPSL). Users are advised to obtain third-party scientific libraries directly from the respective library developers.

## 4.7 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)<br>**Note:** Compiler-specific man pages are available only when the compiler module is loaded. |
| Online help: | `gfortran --help`, `gcc --help`, `g++ --help` |

### 4.7.1 Known Limitations

The GNU compilers do not support the UPC or Coarray Fortran parallel programming models.

### 4.7.2 Known Warnings

Code compiled with `gfortran` using the options `--whole-archive,-lpthread` gets the following warning message issued by `libpthread.a(sem_open.o)`:

```
warning: the use of 'mktemp' is dangerous, better use 'mkstemp'
```

The `--whole-archive` option is necessary to avoid a runtime segmentation fault when using OpenMP libraries. This warning can be safely ignored.

## 4.8 About the Chapel Parallel Programming Language

*Chapel* is a new parallel programming language being developed by Cray Inc. as part of 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 extensions to C and Fortran.

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

## 4.9 About Cross-compilers

The Cray system supports using standalone Linux workstations as code development platforms. When the Cray Application Developer's Environment (CADE) is installed on a suitable Linux system, programs can be written and compiled on the Linux system and then exported to the Cray system for subsequent execution, debugging, and optimization.

You cannot simply install CADE on any Linux system, however. Before you begin, note the following considerations.

- The Linux system must meet the minimum hardware requirements listed in the *Cray Application Developer's Environment Installation Guide*.
- The Linux system must be running a version of the SLES operating system corresponding to the Cray system for which you are developing code. If the Cray system is running CLE 2.2, the Linux system must be running SLES 10. If the Cray system is running CLE 3.0 or later, the Linux system must be running SLES 11. If you attempt to cross operating system versions, your code will compile successfully on the Linux system but not run on the Cray system.
- The behavior of CADE when installed on Linux distributions other than SLES 10 or SLES 11 is untested and therefore unsupported.
- With the exception of the GNU compilers, all compilers are licensed and installed separately. See your site administrator or compiler vendor for further information about license management.
- Along with CADE, you must install the Cray Application Developer's Environment Supplement (CADES, or sometimes CADEsup), which requires satisfying a number of dependencies that are contingent on the compilers you are installing. These dependencies are described in the *Cray Application Developer's Environment Installation Guide*.
- You must install and configure `modules` on the Linux system.

After CADE and your compilers are installed and configured on your Linux system, follow these steps to begin developing code.

1. Load the `xtpe-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 `xtpe-network-gemini` module. If you are developing for a SeaStar system (all others), load the `xtpe-network-seastar` 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 `xtpe-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 `xtpe-`*processor* module corresponding to the compute nodes on the Cray system for which you intend to develop code. Your choices are:

   - `xtpe-barcelona` (quad core, Cray XT4)
   - `xtpe-shanghai` (quad core, Cray XT5 or Cray XE5)
   - `xtpe-istanbul` (six cores)
   - `xtpe-mc8` (eight cores)
   - `xtpe-mc12` (twelve cores)
   - `xtpe-interlagos` (sixteen cores)

   **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 system users only)** Load the `xtpe-`*accelerator* module that is appropriate for the compute nodes on the Cray system for which you intend to develop code. Currently, the only supported option is `xtpe-accel-nvidia20`.

   **Note:** 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 and loads CUDA.

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.

# Libraries [5]

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 37.

## 5.1 Basic Scientific Libraries (LibSci)

Cray LibSci is a collection of numerical routines optimized for best performance on Cray systems. All programming environment modules load `xt-libsci` by default, except where noted. When possible, you should use calls to the Cray LibSci routines in your code in place of calls to public-domain or user-written versions.

### 5.1.1 New in LibSci Release 11.0

Beginning with Scientific Libraries Release 11.0.0, the way in which dynamic libraries are found at execution time is changed.

In previous releases, `RPATH` was included in the dynamic executable, with a path to the libraries to be used at execution time. Beginning with release 11.0.0, `RPATH` is no longer used by default. Instead, at execution time, dynamic libraries are found in `/opt/cray/lib64`. The version defined by the `xt-libsci` module is not used.

This change is not expected to have noticeable consequences for most users, and is intended to improve overall system performance when using dynamic linking at high PE counts. However, in the event that this change does appear to have undesirable consequences, users can restore the previous dynamic linking behavior either by setting the environment variable `CRAY_ADD_RPATH` to `yes`, or else by swapping to an `xt-libsci` module earlier than release 11.0.0.

Statically linked applications are not affected by this change.

## 5.1.2 Basic LibSci Components

**Table 8. LibSci Basics**

| | |
|---|---|
| Module: | `xt-libsci` |
| Man pages: | `intro_libsci`(3s), `intro_blas1`(3s), `intro_blas2`(3s), `intro_blas3`(3s), `intro_blacs`(3s), `intro_lapack`(3s), `intro_scalapack`(3s), `intro_irt`(3), `intro_case`(3s)`intro_crafft`(3s), `intro_fft`(3s), `intro_fftw2`(3), `intro_fftw3`(3)<br>**Note:** Library-specific man pages are available only when the associated module is loaded. |

The Cray LibSci collection contains the following Scientific Libraries.

- BLAS (Basic Linear Algebra Subroutines)
- BLACS (Basic Linear Algebra Communication Subprograms)
- LAPACK (Linear Algebra 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)
- CASE (Cray Adaptive Sparse Eigensolvers)
- CRAFFT (Cray Adaptive Fast Fourier Transform Routines)

## 5.1.3 BLAS and LAPACK

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.

#### 5.1.3.1 Notes

- The BLAS and LAPACK libraries include routines from the 64-bit `libGoto` library from the University of Texas.
- BLAS library behavior is dependent on the `xtpe-`*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 a Linux cross-compiling environment, it may be necessary to load the `xtpe-`*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:
  - `xtpe-barcelona` (quad core, Cray XT4)
  - `xtpe-shanghai` (quad core, Cray XT5)
  - `xtpe-istanbul` (six cores)
  - `xtpe-mc8` (eight cores)

  – `xtpe-mc12` (twelve cores)
  – `xtpe-interlagos` (sixteen cores)

  **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 50.
- 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.

### 5.1.4 BLACS and ScaLAPACK

The BLACS (Basic Linear Algebra Communication Subprograms) library 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 (Scalable LAPACK) 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.

#### 5.1.4.1 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.

### 5.1.5 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.

## 5.1.6 Cray Adaptive Simple Eigensolvers (CASE)

CASE is a collection of simplified interfaces into high-performance LAPACK- and ScaLAPACK-style routines that calculate the eigenvalues and eigenvectors of a symmetric or hermitian matrix. CASE is written in Fortran but has interfaces for C/C++ users.

CASE is designed for C/C++ or Fortran users who are looking for a fast and simple way to calculate the eigenvalues and/or eigenvectors of a matrix and would otherwise call ScaLAPACK or LAPACK eigensolver routines. CASE does this by providing an easy interface for calling the ScaLAPACK and LAPACK eigensolvers that requires only a few simple arguments. From these arguments, CASE interfaces construct the sometimes complicated calling sequences and work arrays required when calling LAPACK and ScaLAPACK eigensolvers directly.

CASE is provided for serial or parallel problems, with single- or double-precision real or complex data types. For more information about CASE, see the `intro_case`(3s) man page.

## 5.1.7 Fourier Transformations

Fast Fourier transforms are handled either by using CRAFFT and FFTW or by using ACML.

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

#### 5.1.7.1 CRAFFT

CRAFFT is a library of Fortran subroutines that compute the discrete Fourier transform in one, two, or three dimensions; of arbitrary input size; and of both real and complex data. CRAFFT provides a simplified interface to several FFT libraries and allows automatic, dynamic selection of the fastest FFT kernel. CRAFFT also provides routines for two- and three-dimensional distributed FFT.

**Note:** CRAFFT requires that module `fftw/3.2.0` or higher be loaded.

To use CRAFFT, add a Fortran `use crafft` statement to any source file that calls a CRAFFT routine, and then call `crafft_init()` before any other CRAFFT routine. This sets up the CRAFFT library for run-time use.

You have a choice of how much planning of FFT kernels you wish to perform. You can set the `CRAFFT_PLANNER` environment variable to `0`, `1` or `2` before execution. Alternately, you can use the `crafft_set_planner()` and `crafft_get_planner()` subroutines to alter and query, respectively, the value of `CRAFFT_PLANNER` during program execution.

For more information about using CRAFFT, see the `intro_crafft`(3s) man page.

#### 5.1.7.2 FFTW

The Programming Environment includes versions 2.1.5 and 3.2.x of the Fastest Fourier Transform in the West (FFTW) library. 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.

Up through LibSci version 10.3.2, loading the LibsSci module automatically loaded the `fftw/3.1.1` module. Beginning with `xt-libsci` modules 10.3.3, FFTW is no longer loaded automatically when you load the LibSci module.

Distributed-memory parallel FFTs are available only in FFTW 2.1.5.1.

The FFTW 3.2.x and FFTW 2.1.5.1 modules cannot be loaded at the same time. You must first unload the other module, if already loaded, before loading the desired one. For example, if you have loaded the FFTW 3.2.x library and want to use FFTW 2.1.5.1 instead, use:

```
% module swap fftw/3.2.2.1 fftw/2.1.5.1
```

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

#### 5.1.7.3 ACML

The AMD Core Math Library (ACML) module is no longer loaded as part of the default PrgEnv environment. BLAS and LAPACK functionality is now provided by 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
```

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.

## 5.2 Fast Math Intrinsics

**Table 9. Fast_mv Basics**

| | |
|---|---|
| Module: | `libfast` |
| Man pages: | `intro_fast_mv`(3),<br>**Note:** Library-specific man pages are available only when the associated module is loaded. |

Fast_mv is a library of high-performance math intrinsic functions. The functions can be used in PGI, CCE, GNU, and PathScale applications. You can use these functions without changing your programs, or you can call them directly.

**Note:** The use of Fast_mv with Intel compilers is not currently supported.

To use Fast_mv functions, load the `libfast` module:

```
% module load libfast
```

Currently, the library contains the following functions.

**Table 10. Fast_mv Math Intrinsics**

| Function Name | Description |
|---|---|
| `cos()` | 64-bit cosine function |
| `exp()` | 64-bit exponential function (the constant *e* raised to a power) |
| `expf()` | 32-bit exponential function (the constant *e* raised to a power) |
| `fastcos()` | scalar 64-bit cosine |
| `fastsin()` | scalar 64-bit sine |
| `fastsincos()` | scalar 64-bit sine and cosine |
| `frda_exp()` | 64-bit exponential function for all elements in an array |
| `frda_log()` | array version of the 64-bit base *e* logarithm |
| `frsa_expf()` | 32-bit exponential function for all elements in an array |
| `frsa_logf()` | array version of the 32-bit base *e* logarithm |
| `log()` | 64-bit logarithm (base *e*) function |
| `logf()` | 32-bit logarithm (base *e*) function |
| `sin()` | 64-bit sine function |
| `sincos()` | 64-bit sine and cosine function |
| `vrda_log()` | array version of the 64-bit base *e* logarithm |
| `__vrd4_log()` | four-element version of the 64-bit base *e* logarithm |
| `vrsa_logf()` | array version of the 32-bit base *e* logarithm |

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

## 5.3 PETSc

**Table 11. PETSc Basics**

| | |
|---|---|
| Modules: | `petsc`, `petsc-complex`, `tpsl` |
| Man pages: | `intro_petsc`(3s)<br>**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 module is dependent on the `xt-libsci` and `xt-asyncpe` modules. Make certain these modules are loaded before using PETSc. When you load the `petsc` module, the Third-party Scientific Libraries (`tpsl`) module is automatically loaded as well, to provide access to the libraries required to support PETSc.

**Note:** Always use the `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 4.9.2. 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 4.1. 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 interative solvers. For further information, see http://crd.lbl.gov/~xiaoye/SuperLU/.
- SuperLU_dist 2.5. 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 3.2. 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 2.7.0. 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 2.4.0 (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 5.1.1. 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 interface.

### 5.3.1 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 PathScale Compiler Suite does not support PETSc 3.1.0.8. If you are using the PathScale compilers, use PETSc 3.0.

- The solvers in Cray PETSc 3.1 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.

## 5.4 Trilinos

**Table 12. Trilinos Basics**

| | |
|---|---|
| Modules: | `trilinos`, `tpsl` |
| Man pages: | `intro_trilinos`(1),<br>**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 dependent on the `xt-libsci` and `xt-asyncpe` modules. Make certain these modules are loaded before using Trilinos. When you load the `trilinos` module, the Third-party Scientific Libraries (`tpsl`) module is automatically loaded as well, to provide access to the libraries required to support Trilinos.

**Note:** Always use the `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 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.

## 5.5 MPT

**Table 13. MPT Basics**

| | |
|---|---|
| Modules: | `xt-mpich2`, `xt-shmem`, `xt-mpt` (deprecated) |
| Man pages: | `intro_mpi`(3), `intro_shmem`(3)<br><br>**Note:** Library-specific man pages are available only when the associated module is loaded. |
| Documentation: | *Getting Started on MPI I/O* (S–2490) |

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

- MPI
- SHMEM

Support for MPI and SHMEM varies significantly depending on whether you are using a Cray XE or Cray XK system with Gemini interconnect or a Cray XT system with SeaStar interconnect, and whether your code uses static or dynamic libraries. Because of these issues, Cray provides a variety of different MPI and SHMEM modules. These modules are hardware-dependent, so your system administrator should install only the modules that support your hardware on your system.

Nonetheless, there are very significant differences in the ways that MPI and SHMEM are implemented on Cray XE/Cray XK and Cray XT systems. These differences are reflected in the functions that are available, and most visibly in the environment variables available on the two types of systems. To see which functions and environment variables are supported on your system, always check the `intro_mpi`(3) or `intro_shmem`(3) man pages.

### 5.5.1 Using MPI and SHMEM Modules

Cray has changed the way that MPI and SHMEM are supported and the default module configurations. These changes were implemented in different releases of CLE and affect your usage of the MPT-related modules.

#### 5.5.1.1 CLE 4.0 Systems

No MPT-related modules are loaded by default.

- If your code uses MPI code only, load the `xt-mpich2` 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 `xt-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 `xt-mpich2` and `xt-shmem` modules. Your code will be linked using both the `-lmpich` and `-lsma` options.
- There is no need to differentiate between code that uses static libraries and code that uses dynamic or shared libraries when selecting and using MPI or SHMEM modules.

The `xt-mpt` module is provided to support legacy code and make files but is deprecated and planned to be removed in a future release.

#### 5.5.1.2 CLE 3.1 Systems

By default, all Cray programming environment modules (`PrgEnv-`*compiler*) load the `xt-mpich2` module. This module supports MPI code only.

- If your code uses MPI code only, verify that the `xt-mpich2` module is loaded before compiling or linking. This ensures that your code is linked using the `-lmpich` option.
- If your code uses SHMEM code only, unload the `xt-mpich2` module and load the `xt-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, verify that both the `xt-mpich2` and `xt-shmem` modules are loaded. Your code will be linked using both the `-lmpich` and `-lsma` options.
- There is no need to differentiate between code that uses static libraries and code that uses dynamic or shared libraries when selecting and using MPI or SHMEM modules.

The `xt-mpt` module is provided to support legacy code and make files but is deprecated and planned to be removed in a future release.

#### 5.5.1.3 CLE 3.0 Systems

By default, all Cray programming environment modules load the `xt-mpt` module. The `xt-mpt` module supports both MPI and SHMEM code, but causes the compiler drivers to link code using both the `-lmpich` and `-lsma` options. This can cause undesirable run time dependencies when using dynamic or shared libraries.

- If your code uses static libraries only, use the `xt-mpt` module. There is no need to differentiate between MPI and SHMEM code.
- If your code uses dynamic libraries and MPI code only, unload the `xt-mpt` module and load the `xt-mpich2` module before compiling or linking. This ensures that your code is linked using the `-lmpich` option.
- If your code uses dynamic libraries and SHMEM code only, unload the `xt-mpt` module and load the `xt-shmem` module before compiling or linking. This ensures that your code is linked using the `-lsma` option.
- If your code uses dynamic libraries and both MPI and SHMEM, unload the `xt-mpt` module and load both the `xt-mpich2` and `xt-shmem` modules. Your code will be linked using both the `-lmpich` and `-lsma` options.

#### 5.5.1.4 CLE 2.2 Systems

By default, all Cray programming environment modules load the `xt-mpt` module. Depending on which version of MPT is installed on your system, the `xt-mpich2` and `xt-shmem` modules may not be available.

Be advised that if the `xt-mpt` module is loaded, the compiler drivers will link code using both the `-lmpich` and `-lsma` options. This can cause undesirable run time dependencies when using dynamic or shared libraries.

## 5.6 Hugepages

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 systems, the implementation of the `libhugetlbfs` library differs on these two architectures.

### 5.6.1 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* (S–2496).

### 5.6.2 Cray XE Usage

By default, the running 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 systems, PGAS, SHMEM and MPI applications are likely to require the usage of huge pages for static data and/or the heap.

Modules `craype-hugepages2m` and `craype-hugepages8m` (released in xt_asyncpe 4.8) set the necessary link options and environment variables (e.g., `HUGETLB_DEFAULT_PAGE_SIZE`, `HUGETLB_MORECORE`, `HUGETLB_ELFMAP`) to facilitate the usage of 2 or 8 MB huge pages, respectively.

It is not required to use the `-m` option on the `aprun` command on the Cray XE 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* (S–2496).

# Debugging Code [6]

The Cray system supports a variety of debugging options ranging from simple command-line debuggers to separately licensed third-party GUI tools and 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. However, if 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.

## 6.1 Cray Debugger Support Tools

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

- `lgdb`: a modified version of `gdb` that interfaces with `aprun` and works on Cray system compute nodes
- `atp`: a system that monitors user applications and replaces the core dump with a more comprehensive stack backtrace and analysis
- MRNet: a software overlay network that provides multicast and reduction communications for parallel and distributed tools and systems
- STAT: stack trace analysis tool

### 6.1.1 Using `lgdb`

**Table 14. `lgdb` Basics**

| | |
|---|---|
| Module: | `xt-lgdb` |
| Command: | `lgdb` |
| Man page: | `lgdb`(1)<br><br>**Note:** Tool-specific man pages are available only when the associated module is loaded. |
| Online help: | `lgdb --help` |

The `lgdb` command is used to launch an application and related `gdb` server processes on remote nodes for debugging purposes. After `lgdb` is launched, it provides directions regarding how to run the `gdb` debugger and attach to application processes.

When using `lgdb` on a Cray system, remember that you use `lgdb` to launch an instance of `aprun`, which in turn launches the application to be debugged. You cannot use `lgdb` to launch an application directly.

**Example 1. Launching and debugging a single-rank application**

```
$ lgdb --pes=0 --command="aprun -n1 ./myapp" --lib=/lib64/libthread_db.so.1
```

You must specify the number of PEs (ranks) to which you want to attach `gdbservers`. Since this is a single-rank application, specify `0` for the rank. The `--lib=`*path_to_library* is optional and needs to be specified only for certain systems, such as X86_64 Linux, where the `libthread_db` library is required.

**Example 2. Attaching to an already-running application**

```
$ lgdb --pes=0 --pid=aprun_pid --lib=/lib64/libthread_db.so.1
```

Remember, you use `lgdb` to launch an instance of `aprun`, which in turn launches the application to be debugged. Therefore to attach to an already-running application, you must specify the *pid* of the instance of `aprun` that launched the application, not the application itself.

For more information, see the `lgdb`(1) man page.

## 6.1.2 Using Abnormal Termination Processing (ATP)

**Table 15. `atp` Basics**

| | |
|---|---|
| Module: | `atp` |
| Commands: | `ataprun`, `atpFrontend` (deprecated) |
| Man page: | `intro_atp`(1)<br>**Note:** Tool-specific man pages are available only when the associated module is loaded. |
| Online help: | None required |

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 `statview`, (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.

**Note:** The `statview` command is 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 core dumping.

### 6.1.2.1 On CLE 3.0 or Later

On CLE 3.0 and later systems, when the `atp` module is loaded, ATP is launched automatically whenever a job is launched by using the `aprun` command.

Do not use the `atpaprun` or `atpFrontend` commands on CLE 3.0 or later systems. These commands are for use on CLE 2.2 or earlier systems only.

#### 6.1.2.2 On CLE 2.2 or Earlier

On CLE 2.2 and earlier systems, ATP must be invoked manually. This is done by loading the `atp` module and then using either the `atpaprun` command to launch your application, or after the application has been launched by using the `atpFrontend` command to attach to an already-running application.

**Example 3. Launching an application and invoking ATP**

`atpaprun` *aprun_parameters*

**Example 4. Attaching ATP to an already-running application**

`atpFrontend` *application_PID*

For more information, see the `intro_atp`(1) man page.

### 6.1.3 Using MRNet

**Table 16. MRNet Basics**

| | |
|---|---|
| Module: | `mrnet` |
| Online help: | http://www.paradyn.org/mrnet/ |

MRNet is a customizable, high-throughput communication software system for parallel tools and applications with a master/slave architecture. MRNet reduces the cost of these tools' activities by incorporating a tree-based overlay network (TBON) of processes between the tool's front-end and back-ends. MRNet uses the TBON to distribute many important tool communication and computation activities, reducing analysis time and keeping tool front-end loads manageable.

MRNet-based tools send data between front-end and back-ends on logical flows of data called streams. MRNet internal processes use filters to synchronize and aggregate data sent to the tool's front-end. Using filters to manipulate data in parallel as it passes through the network, MRNet can efficiently compute averages, sums, and other more complex aggregations on back-end data.

### 6.1.4 Using STAT

**Table 17. STAT Basics**

| | |
|---|---|
| Module: | `stat` |
| Commands: | `statview` |
| Online help: | http://www.paradyn.org/STAT/STAT.html |

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.

## 6.2 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 that contains full DWARF information and 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 both contain 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 1. 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:

   ```
   $ lgdb --pes=0 --command="aprun -n1 ./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.

### 6.2.1 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 the `lgdb` and Allinea DDT debuggers.

## 6.3 About Core Files

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 overwritten.

On large MPP systems where an application might be running thousands of processes, a conventional core file may not 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 data, and in place of a core file, all stack backtraces 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.

For more information about ATP, see Using Abnormal Termination Processing (ATP) on page 65.

## 6.4 Using TotalView

**Table 18. TotalView Basics**

| | |
|---|---|
| Module: | `xt-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 TotalView Technologies, LLC, that provides source-level debugging of applications running on multiple compute nodes. TotalView is compatible with the Cray, PGI, GCC, PathScale, 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 5. 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 6. 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 7. 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`.

### 6.4.1 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.

## 6.5 Using DDT

**Table 19. DDT Basics**

| | |
|---|---|
| Module: | `ddt` |
| Commands: | `ddt` |
| Man page: | `ddt`(1)<br>**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 is an optional product from Allinea Software, and 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, PathScale, 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, along with 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.

### 6.5.1 Known Limitations

DDT does not support UPC.

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.

# Optimizing Code [7]

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

## 7.1 Improving I/O

### 7.1.1 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 replaces the `stdio` (`glibc`, `libio`) layer of buffering with an additional layer of buffering, thus improving program performance by enabling asynchronous prefetching and caching of file data.

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

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. Recompile (or optionally, relink) the program.
3. Set the `IOBUF_PARAMS` environment variable as needed.

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

   **Note:** The simplest parameter specification is:

   ```
   % setenv IOBUF_PARAMS='*'
   ```

   This setting matches all files and enables buffering with default parameters.

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.

The behavior of IOBUF is controlled by the use of environment variables, the most significant of which is `IOBUF_PARAMS`. 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.

For more information about valid `IOBUF_PARAMS` parameters and their usage, see the `iobuf`(3) man page.

### 7.1.2 Improving MPI I/O

**Table 21. MPI I/O Basics**

| | |
|---|---|
| Module: | `xt-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* (S–2490).

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 Analysis Tools* (S–2376).

## 7.2 Using Compiler Optimizations

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

### 7.2.1 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, which cause the compiler to generate annotated loopmark listings showing which optimizations were performed where in the code. Together with the `-h negmsgs` options, which generate listings showing which potential optimizations 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* (S–3901) and *Cray C and C++ Reference Manual* (S–2179).

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.

## 7.3 Using the Cray Performance Analysis Tools

**Table 22. Performance Analysis Basics**

| | |
|---|---|
| Module: | `perftools` |
| Commands: | `pat_build`, `pat_report`, `app2` |
| Man pages: | `intro_craypat`(1), `pat_build`(1), `pat_report`(1), `pat_help`(1), `app2`(1), `intro_papi`(3), `hwpc`(5), `nwpc`(5)<br><br>**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 Analysis Tools* |

**Note:** The PGI profiling tools, `pgprof` and `pgcollect`, are 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 Cray Performance Analysis Tools suite consists of three components:

- **CrayPat**: the program instrumentation, data capture, and basic text reporting tool
- **Cray Apprentice2**: the graphical analysis and data visualization tool
- **PAPI**: the Performance Application Programming Interface

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
```

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, CrayPat consists of the following major components:

- `pat_build`, the utility used to instrument programs
- The CrayPat run time environment, which collects the specified performance data
- `pat_report`, the first-level analysis tool used to produce text reports or export data for more sophisticated analysis
- `pat_help`, the command-line driven online help system

All CrayPat components, including the man pages and help system, are available only when the `perftools` module is loaded.

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 re-linking is sufficient, but as a rule it's best to load the `perftools` module and then recompile.

When instrumenting a program, CrayPat requires that the object (`.o`) files created during compilation be present, as well as the library (`.a`) files, if any. However, most compilers automatically delete the `.o` and `.a` files when working with single source files and compiling and linking in a single step, therefore it is good practice to compile and link in separate steps and use the compiler command line option to preserve these files. For example, if you are using the Cray Compiling Environment (CCE) Fortran compiler, compile using either of these command line options:

```
> ftn -c sourcefile.f
```

Alternatively:

```
> ftn -h keepfiles sourcefile.f
```

Then link the object files to create the executable program:

```
> ftn -o executable sourcefile.o
```

### 7.3.1 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 -O apa 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. 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 Analysis Tools*.

### 7.3.2 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 Analysis Tools*.

### 7.3.3 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 with the `-O apa` option, 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 Analysis Tools*.

### 7.3.4 For More Information

In addition to *Using Cray Performance 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.

## 7.4 Using 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 Analysis Tools*.

## 7.5 Using PAPI

The Performance API (PAPI) is a standard API for accessing microprocessor registers. CrayPat uses PAPI to interface to the Cray system hardware; therefore the module `papi` is normally loaded as part of the `perftools` module.

The interface between PAPI and CrayPat is normally transparent to the user. However, advanced users may want to bypass CrayPat and work with PAPI directly. In this case, you must unload the `perftools` module and then reload only the `papi` module.

```
> module unload perftools
> module load papi
```

**Note:** CrayPat and direct PAPI commands cannot be used at the same time. Therefore, `pat_build` does not allow you to instrument a program that has also been instrumented using calls to PAPI functions.

For more information about PAPI, see the `intro_papi`(3) and `papi_counters`(5) man pages. To see what sorts of information PAPI is capable of capturing on your system, use the `papi_avail` and `papi_native_avail` commands.

**Note:** Effective with PAPI version 3.7.2, The PAPI Group has discontinued providing man pages for individual PAPI functions.

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