The gnulicinfo(7) man page contains the Open Source Software licenses (the "Licenses"). Your use of this software release constitutes your acceptance of the License terms and conditions.

U.S. GOVERNMENT RESTRICTED RIGHTS NOTICE

The Computer Software is delivered as "Commercial Computer Software" as defined in DFARS 48 CFR 252.227-7014.

All Computer Software and Computer Software Documentation acquired by or for the U.S. Government is provided with Restricted Rights. Use, duplication or disclosure by the U.S. Government is subject to the restrictions described in FAR 48 CFR 52.227-14 or DFARS 48 CFR 252.227-7014, as applicable.

Technical Data acquired by or for the U.S. Government, if any, is provided with Limited Rights. Use, duplication or disclosure by the U.S. Government is subject to the restrictions described in FAR 48 CFR 52.227-14 or DFARS 48 CFR 252.227-7013, as applicable.

Technical Data acquired by or for the U.S. Government, if any, is provided with Limited Rights. Use, duplication or disclosure by the U.S. Government is subject to the restrictions described in FAR 48 CFR 52.227-14 or DFARS 48 CFR 252.227-7013, as applicable.

Cray, LibSci, PathScale, and UNICOS are federally registered trademarks and Active Manager, Baker, Cascade, Cray Apprentice2, Cray Apprentice2 Desktop, Cray C++ Compiling System, Cray CX, Cray CX1, Cray CX1-iWS, Cray CX1-LC, Cray CX1000, Cray CX1000-C, Cray CX1000-G, Cray CX1000-S, Cray CX1000-SC, Cray CX1000-SM, Cray CX1000-HN, Cray Fortran Compiler, Cray Linux Environment, Cray SHMEM, Cray X1, Cray X1E, Cray X2, Cray XD1, Cray XE, Cray XE6, Cray XMT, Cray XR1, Cray XT, Cray XTm, Cray XT3, Cray XT4, Cray XT5, Cray XT5i, Cray XT5m, Cray XT6, Cray XT6m, Cray Doc, CrayPort, CRInform, ECOphlex, Gemini, Libsci, NodeKARE, RapidArray, SeaStar, SeaStar2, SeaStar2+, Threadstorm, UNICOS/lc, UNICOS/mk, and UNICOS/mp are trademarks of Cray Inc.

AMD Opteron is a trademark of Advanced Micro Devices, Inc.. GNU is a trademark of The Free Software Foundation. Intel is a trademark of Intel Corporation or its subsidiaries in the United States and other countries. Java is a trademark of Oracle and/or its affiliates. Linux is a trademark of Linus Torvalds. Lustre is a trademark of Oracle and/or its affiliates. Moab and TORQUE are trademarks of Adaptive Computing Enterprises, Inc. MySQL is a trademark of Oracle and/or its affiliates. Opteron is a trademark of Advanced Micro Devices, Inc. PBS Professional is a trademark of Altair Grid Technologies. PETSc is a trademark of Copyright (C) 1995-2004 University of Chicago. PGI is a trademark of The Portland Group Compiler Technology, STMicroelectronics, Inc. Platform is a trademark of Platform Computing Corporation. RSA is a trademark of RSA Security Inc. SecurID is a trademark of RSA Security Inc. TotalView is a trademark of TotalView Technology, LLC. Windows is a trademark of Microsoft Corporation. UNIX, the “X device,” X Window System, and X/Open are trademarks of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

Version 1.0 Published December 2004 Draft documentation to support Cray XT3 early-production systems.
Version 1.0 Published March 2005 Draft documentation to support Cray XT3 limited-availability systems.
Version 1.1 Published June 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.1 and UNICOS/lc 1.1 releases.
Version 1.2 Published August 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.2 and UNICOS/lc 1.2 releases.
Version 1.3 Published November 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.3 and UNICOS/lc 1.3 releases.
Version 1.4 Published April 2006 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.4 and UNICOS/lc 1.4 releases.
Version 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.

Version 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.

Version 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.

Version 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.

Version 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.

Version 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.

Version 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.

Version 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.
CADE 5.0  For CADE release 5.0 this guide has been completely redesigned and rewritten, with more emphasis on the kinds of information users new to Cray systems need in order to get started and more links to external sources of more information. The detailed aprun, ALPS, batch system, and other OS-dependent information has been moved to a new guide, Workload Management and Application Placement for the Cray Linux Environment, which is packaged with the operating system.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong> [1]</td>
<td>11</td>
</tr>
<tr>
<td>1.1 What You Must Know About Your Site</td>
<td>11</td>
</tr>
<tr>
<td>1.1.1 Cray XE or Cray XT?</td>
<td>12</td>
</tr>
<tr>
<td>1.1.2 How Many Cores?</td>
<td>12</td>
</tr>
<tr>
<td>1.1.3 Which Operating System?</td>
<td>13</td>
</tr>
<tr>
<td>1.1.4 What Is a Compute Node?</td>
<td>13</td>
</tr>
<tr>
<td>1.1.5 Which File System?</td>
<td>14</td>
</tr>
<tr>
<td>1.1.6 Which Batch System?</td>
<td>14</td>
</tr>
<tr>
<td>1.1.7 What Is the <code>hostname</code>?</td>
<td>14</td>
</tr>
<tr>
<td>1.2 Logging In</td>
<td>14</td>
</tr>
<tr>
<td>1.2.1 UNIX or Linux Users</td>
<td>15</td>
</tr>
<tr>
<td>1.2.2 Windows, Macintosh, or Other Users</td>
<td>16</td>
</tr>
<tr>
<td>1.3 Navigating the File Systems</td>
<td>19</td>
</tr>
<tr>
<td><strong>Using Modules</strong> [2]</td>
<td>21</td>
</tr>
<tr>
<td>2.1 What Is Loaded Now?</td>
<td>21</td>
</tr>
<tr>
<td>2.2 What Is Available?</td>
<td>22</td>
</tr>
<tr>
<td>2.3 Loading and Unloading Modulefiles</td>
<td>22</td>
</tr>
<tr>
<td>2.4 Swapping Modulefiles</td>
<td>23</td>
</tr>
<tr>
<td>2.5 Release Notes and Module Help</td>
<td>24</td>
</tr>
<tr>
<td>2.6 For More Information</td>
<td>26</td>
</tr>
<tr>
<td><strong>Batch Systems and Program Execution</strong> [3]</td>
<td>27</td>
</tr>
<tr>
<td>3.1 Interactive Mode</td>
<td>28</td>
</tr>
<tr>
<td>3.1.1 Notes</td>
<td>28</td>
</tr>
<tr>
<td>3.2 Batch Mode</td>
<td>29</td>
</tr>
<tr>
<td>3.3 Using <code>aprun</code></td>
<td>31</td>
</tr>
<tr>
<td><strong>Using Compilers</strong> [4]</td>
<td>33</td>
</tr>
<tr>
<td>4.1 About Compiler Drivers</td>
<td>33</td>
</tr>
</tbody>
</table>

S–2396–50
4.2 About C/C++ Floating Point Data Types ........................................ 34
4.3 About PGI Compilers ..................................................................... 35
  4.3.1 Known Limitations ............................................................... 35
4.4 About the Cray Compiling Environment (CCE) ................................ 36
  4.4.1 Known Limitations ............................................................... 36
4.5 About PathScale Compilers ........................................................... 36
  4.5.1 Known Limitations ............................................................... 37
4.6 About Intel Compilers .................................................................. 37
  4.6.1 Known Limitations ............................................................... 37
4.7 About GNU Compilers .................................................................. 37
  4.7.1 Known Limitations ............................................................... 38
4.8 About the Chapel Parallel Programming Language ......................... 38
4.9 About Cross-compilers .................................................................. 38

5.1 Scientific Libraries ........................................................................ 41
  5.1.1 BLAS and LAPACK ............................................................... 42
    5.1.1.1 Notes ............................................................................. 43
  5.1.2 BLACS and ScaLAPACK ......................................................... 44
    5.1.2.1 Notes ............................................................................. 44
  5.1.3 IRT ....................................................................................... 45
  5.1.4 FFT ..................................................................................... 46
    5.1.4.1 CRAFFT ................................................................. 46
    5.1.4.2 FFTW ....................................................................... 46
    5.1.4.3 ACML ......................................................................... 47
5.2 About the Fast_mv Library ............................................................. 47
5.3 PETSc ......................................................................................... 49
  5.3.1 Notes ................................................................................. 50
5.4 Trilinos ....................................................................................... 50
5.5 MPT .......................................................................................... 51
  5.5.1 Static Libraries ..................................................................... 52
  5.5.2 Dynamic Libraries ............................................................... 52

6.1 Cray Debugger Support Tools ......................................................... 53
  6.1.1 Using lgdb ......................................................................... 54
  6.1.2 Using Abnormal Termination Processing (ATP) ......................... 55
    6.1.2.1 On CLE 3.0 or Later ...................................................... 55
6.1.2.2 On CLE 2.2 or Earlier .................................................. 56
6.1.3 Using MRNet (Deferred implementation) .......................... 56
6.1.4 Using STAT (Deferred implementation) ............................. 56
6.2 Using Fast-Track Debugging .............................................. 57
   6.2.1 Supported Compilers and Debuggers ............................... 58
6.3 About Core Files .......................................................... 58
6.4 Using TotalView .......................................................... 58
   6.4.1 Known Limitations .................................................. 60
6.5 Using DDT ............................................................... 60
   6.5.1 Known Limitations .................................................. 61
Performance Analysis [7] ...................................................... 63
   7.1 Using CrayPat .......................................................... 64
      7.1.1 Instrumenting the Program ...................................... 65
      7.1.2 Collecting Data .................................................. 65
      7.1.3 Analyzing Data .................................................. 65
      7.1.4 For more information ............................................ 66
   7.2 Using Cray Apprentice2 ................................................ 66
   7.3 Using PAPI ............................................................ 67
Glossary .............................................................................. 69
Figures .................................................................................. 16
   Figure 1. Selecting SSH Protocol ......................................... 16
   Figure 2. Enabling X11 Forwarding ...................................... 17
   Figure 3. Logging In ......................................................... 18
Tables .................................................................................... 32
   Table 1. aprun versus qsub Options ..................................... 32
   Table 2. C/C++ Data Type Sizes ......................................... 34
   Table 3. PGI Compiler Basics ............................................. 35
   Table 4. Cray Compiler Basics .......................................... 36
   Table 5. PathScale Compiler Basics .................................... 36
   Table 6. Intel Compiler Basics ......................................... 37
   Table 7. GNU Compiler Basics .......................................... 37
   Table 8. LibSci Basics ...................................................... 41
   Table 9. Fast_mv Basics .................................................. 47
   Table 10. PETSc Basics .................................................... 49
   Table 11. Trilinos Basics .................................................. 50
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>MPT Basics</td>
<td>51</td>
</tr>
<tr>
<td>13</td>
<td>lgdb Basics</td>
<td>54</td>
</tr>
<tr>
<td>14</td>
<td>atp Basics</td>
<td>55</td>
</tr>
<tr>
<td>15</td>
<td>TotalView Basics</td>
<td>58</td>
</tr>
<tr>
<td>16</td>
<td>DDT Basics</td>
<td>60</td>
</tr>
<tr>
<td>17</td>
<td>Performance Analysis</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Basics</td>
<td></td>
</tr>
</tbody>
</table>
Welcome to the Cray Application Developer’s Environment (CADE) user’s guide. This guide describes the software environment and tools used to develop, debug, and run applications on Cray XT and Cray XE systems. It is intended as a general overview and introduction to the Cray system for new users and application programmers.

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 and Cray XE systems are:

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

Supercomputers designed to scale to immense size and run applications requiring high-performance, large-scale processing, high network bandwidth, and complex inter-processor communications. Both series of 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 XE or Cray XT?

From the programmer's point of view, the most significant difference between the two systems is:

- Cray XE systems use Cray Gemini ASICs to manage inter-processor communications
- Cray XT systems use Cray SeaStar or Cray SeaStar2+ application-specific integrated circuits (ASICs) to manage inter-processor communications

Because of this hardware difference, the two systems use different versions of the MPI and SHMEM libraries, which offer different sets of optional controls to the application developer. Cray XE systems also 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 interconnect.

The differences between the Cray XT and Cray XE versions of MPI and SHMEM are discussed in more detail in MPT on page 51.

The Gemini GNI and DMAPP APIs are discussed in detail in Using the GNI and DMAPP APIs.

1.1.2 How Many Cores?

The next most 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") AMD Opteron CPUs 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.
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 XE5 and Cray XE6 systems run CLE release 3.1 or later. CLE 3.x is based on SLES 11.
- Cray XT6 and Cray XT6m systems run CLE release 3.0 or later. CLE 3.0 is based on SLES 11.
- All other systems may be running CLE release 2.1, 2.2, 3.0, or 3.1. If you are not certain which release your site is using, check the MOTD when you log in, or 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 21.) CLE 2.x is based on SLES 10.

Note: The Catamount compute node operating system is no longer supported by Cray.

1.1.4 What Is a Compute Node?

From the application developer's point of view, a modern 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.

- **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 or TORQUE Resource Manager), generally have access only to the high-performance parallel file system (typically Lustre), 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 a source of confusion. 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, you must 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 XT and Cray XE systems require the use of a high-performance parallel file system. Most sites currently use the Lustre File System, and 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 XT and Cray XE systems typically operate under the control of a batch system such as PBS, OpenPBS, PBS Professional, TORQUE Resource Manager, or the Moab Cluster Suite. All examples shown in this guide were developed using PBS Pro and/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 XT and Cray XE 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's 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 of the application development tools use GUI displays.

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

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

Or

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

In any case, after you SSH to the system, you may have to answer one or more RSA or password challenges, and then you are logged into the system. A series of system status and MOTD (message of the day) type 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 19.
1.2.2 Windows, Macintosh, or Other Users

If you use a Windows, Macintosh, or other 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 set the preferred SSH protocol version:

Figure 1. Selecting SSH Protocol
Click **X11** in the left pane and check the box to enable X11 forwarding:

**Figure 2. Enabling X11 Forwarding**
Then click **Session** in the left pane to return to the **Basic options** window.

**Figure 3. Logging In**

![PuTTY Configuration](image)

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) type 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.
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.

> cd $HOME

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@pt1:/narwhalnid8 8998913280 6946443260 1595348672 82% /lus/nid00008
```

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

```
users/userID> lfs df
```

```
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.
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/3.1.6.5          12) Base-opts/2.2.48B
2) xtpe-target-cnl          13) pgi/10.4.0
3) xt-service/2.2.48B       14) totalview-support/1.1.0
4) xt-os/2.2.48B            15) xt-totalview/8.8.0
5) xt-boot/2.2.48B          16) xt-libsci/10.4.4
6) xt-lustre-ss/2.2.48B_1.6.5 17) pmi/1.0-1.0000.7628.10.2.ss
7) cray/job/1.5.5-0.1.2.0202.19481.53.6 18) xt-mpt/4.1.1
8) cray/csa/3.0.0-1.2.0202.19602.75.1 19) xt-pe/2.2.48B
9) cray/account/1.0.0-2.0202.19482.49.3 20) xt-asyncpe/3.9.39
10) cray/projdb/1.0.0-1.0202.19483.52.1 21) PrgEnv-pgi/2.2.48B
11) pbs/default             22) cray/MySQL/5.0.64-1.0202.2899.21.1
This list is not as complicated as it may look. Essentially, it breaks down into three groups: operating system modules, programming environment modules, and support modules. For example, the xt-os module indicates that this system is running the CLE release 2.2 operating system, while the PrgEnv-pgi module indicates that PGI compiler suite is currently loaded.

### 2.2 What Is Available?

To see which module files are available on your system, use the `module avail` command.

**Note:** The `module avail` command produces an alphabetical listing of every module file in your module use path and has no option for "grepping." For this reason it is usually more useful to use the command with an argument. For example, if you are looking for a specific version of the PGI compiler suite, enter the following command.

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

```
---------------------------------- /opt/modulefiles ----------------------------------
PrgEnv-pgi/2.2.26 PrgEnv-pgi/2.2.31 PrgEnv-pgi/2.2.37
PrgEnv-pgi/2.2.27 PrgEnv-pgi/2.2.33 PrgEnv-pgi/2.2.38
PrgEnv-pgi/2.2.28 PrgEnv-pgi/2.2.34 PrgEnv-pgi/2.2.39
PrgEnv-pgi/2.2.29 PrgEnv-pgi/2.2.35 PrgEnv-pgi/2.2.41
PrgEnv-pgi/2.2.30 PrgEnv-pgi/2.2.36 PrgEnv-pgi/2.2.48B (default)
```

One module is generally 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.

### 2.3 Loading and Unloading Modulefiles

If a module file 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 22 are present on your system but PrgEnv-pgi is not already loaded, then:

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

loads PrgEnv-pgi/2.2.48B. To load a non-default version of the module—for example, release 2.2.39—you must specify the full module name.

```
users/yourname> module load PrgEnv-pgi/2.2.39
```
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 version 2.2.39, use the `module unload` command to remove the version currently loaded.

```bash
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 21:

```bash
users/yourname> module list
Currently Loaded Modulefiles:
1) modules/3.1.6.5
2) xtpe-target-cn1
3) xt-service/2.2.48B
4) xt-os/2.2.48B
5) xt-boot/2.2.48B
6) xt-lustre-ss/2.2.48B_1.6.5
7) cray/job/1.5.5-0.1_2.0202.19481.53.6
8) cray/csa/3.0.0-1_2.0202.19602.75.1
9) cray/account/1.0.0-2.0202.19482.49.3
10) cray/projdb/1.0.0-1.0202.19483.52.1
11) pbs/default
12) Base-opts/2.2.48B
13) cray/MySQL/5.0.64-1.0202.2899.21.1
```

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

```bash
users/yourname> module load PrgEnv-pgi/2.2.39
```

Then use the `module list` command again, you can see that the eight associated modules have been reloaded automatically, but this time using the versions that are appropriate for use with the 2.2.39 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:

```bash
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 21.

users/yourname> module list
Currently Loaded Modulefiles:
  1) modules/3.1.6.5
  2) xtpe-target-cn1
  3) xt-service/2.2.48B
  4) xt-os/2.2.48B
  5) xt-boot/2.2.48B
  6) xt-lustre-ss/2.2.48B_1.6.5
  7) cray/job/1.5.5-0.1_2.0202.19481.53.6
  8) cray/csa/3.0.0-1.2.0202.19602.75.1
  9) cray/account/1.0.0-2.0202.19482.49.3
 10) cray/projdb/1.0.0-1.0202.19483.52.1
 11) pbs/default
 12) Base-opts/2.2.48B
 13) cray/MySQL/5.0.64-1.0202.2899.21.1
 14) PrgEnv-cray/1.0.1
 15) xt-asyncpe/3.9.39
 16) xt-mpt/4.1.1
 17) pmi/1.0-1.0000.7628.10.2.ss
 18) xt-pe/2.2.48B
 19) xt-libsci/10.4.4
 20) acml/4.4.0

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

2.5 Release Notes and Module Help

Most module files on the Cray system include module help that is specific to the module file. The exact content of the module help varies from vendor to vendor and release to release, but it 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 that you want. For example, module help PrgEnv-pgi and module help pgi display different information.
The module help content is displayed.

pgi 10.5.0
---------

Release Date: May 20, 2010

Purpose:
--------

Features of PGI 10.5.0 are documented at:

The following bugs are fixed in the PGI 10.5.0 release.
751661  PGI C/C++ handles non-conforming pgm with broken perfectly nested loops
  - C (no msg) & C++ (signal 11) [TPR 16167]
755930  PGI OpenMP pgf90 gets incorrect output for REDUCTION with ALLOCATABLE array
  as variable. [16507]
757052  PGI OpenMP pgf90 incorrect output from threadprivate(A) when A is allocatable. 
  [16593]
757192  PGI OpenMP pgf90 !$omp do lastprivate(A) returns incorrect output. [16607]
758613  PGI OpenMP pgf90 incorrect output from sections lastprivate(a) where a is allocatable [16808]

Product and OS Dependencies:
----------------------------

pgi 10.5.0 is supported on Cray XT systems running the UNICOS/IC 2.0 
CNL and CLE 2.1 CNL or later operating systems.

NOTE: Licensing change:

PGI issued licensing policy has changed. The license will work with
existing PGI compilers and all new compilers released prior to a
specified date. A new license will be required to use any compilers
released after the expiration date.

Documentation:
---------------

PGI Tools Guide, /opt/pgi/10.5.0/linux86-64/10.5/doc/pgi10tools.pdf
PGI Workstation 10.5 Release Notes, /opt/pgi/10.5.0/linux86-64/10.5/doc/pgiwsrn105.pdf
PGI Fortran Reference, /opt/pgi/10.5.0/linux86-64/10.5/doc/pgifortref.pdf
PGI Workstation 10.5 Installation Guide, /opt/pgi/10.5.0/linux86-64/10.5/doc/pgiwsinstall105.pdf

Installation:
-------------

env CRAY_INSTALL_DEFAULT=1 rpm -ihv --oldpackage pgi-10.5.0-1.x86_64.rpm
2.6 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 3.1.6.5 (Copyright GNU GPL v2 1991):
Available Commands and Usage:
  + add|load   modulefile [modulefile ...]
  + rm|unload  modulefile [modulefile ...]
  + switch|swap modulefile1 modulefile2
  + display|show modulefile [modulefile ...]
  + avail      [modulefile [modulefile ...]]
  + use [-a|--append] dir [dir ...]
  + unused     dir [dir ...]
  + update
  + 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.
User applications are always launched on the 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*.

At most sites access to the compute node resources is managed by a batch control system, typically PBS Pro or Moab/TORQUE. Users run jobs either by using the `qsub` command to submit a job script, or else by using the `qsub` command 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 `qsub` and `aprun` command options, which are not necessarily the same. If running a batch job, modify your script accordingly.

4. Use the `qsub` command either to submit your job script or 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. To begin an interactive session, use the `qsub -I` command.

```
users/yourname> qsub -IV1 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*.

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 is typically 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 31 and described in detail in Workload Management and Application Placement for the Cray Linux Environment.

When you launch an interactive batch session, you must request the maximum number of resources you expect to use. Once a batch session begins, you can only use fewer resources than initially requested. You cannot use the aprun command to use more resources than you reserved using the qsub command.

### 3.2 Batch Mode

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

To run a batch script, use the qsub command.

```
quid/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:

```
1: #!/bin/bash
2: #PBS -A account
3: #PBS -N job_name
4: #PBS -j oe
5: #PBS -l walltime=1:00:00,size=192
6:
7: cd $PBS_O_WORKDIR
8: date
9: aprun -n 192 ./a.out > my_output_file 2>&1
```
Parsing this script line-by-line, it would be interpreted as follows:

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

   **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.

5. Reserve 192 cores 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 cores, 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 wallclock 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

<table>
<thead>
<tr>
<th>aprun Option</th>
<th>qsub -l Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n 4</td>
<td>-l mppwidth=4</td>
<td>Width (number of PEs)</td>
</tr>
<tr>
<td>-d 2</td>
<td>-l mppdepth=2</td>
<td>Depth (number of CPUs hosting OpenMP threads)</td>
</tr>
<tr>
<td>-N 1</td>
<td>-l mppnppn=1</td>
<td>Number of PEs per node</td>
</tr>
<tr>
<td>-L 5,6,7</td>
<td>-l mppnodes=&quot;5,6,7&quot;</td>
<td>Candidate node List</td>
</tr>
<tr>
<td>-m 1000</td>
<td>-l mppmem=1000</td>
<td>Memory per PE</td>
</tr>
</tbody>
</table>

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.
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.

- The Portland Group, Parallel Fortran, C, and C++
- Cray Inc., Cray Compiling Environment (CCE) (Fortran, C, and C++)
- PathScale Inc., PathScale Compiler Suite (Fortran and C++)
- Intel Inc., Intel 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. To date we have found that no one compiler is always the best for all possible applications under all possible circumstances, nor does any one compiler always produce faster executable code than any other.

### 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.
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 \texttt{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, \textbf{do not} use the native compiler commands. For example, if you are using the PGI compiler suite, do not use the \texttt{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 Cray compute nodes.

### 4.2 About C/C++ Floating Point Data Types

The C/C++ compilers differ on the size of the \textit{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.

\begin{table}[h]
\centering
\caption{C/C++ Data Type Sizes}
\begin{tabular}{ll}
\hline
Data Type & Size in Bytes \\
\hline
unsigned char & 1 \\
signed char & 1 \\
unsigned short & 2 \\
signed short & 2 \\
unsigned int & 4 \\
signed int & 4 \\
unsigned long & 8 \\
signed long & 8 \\
unsigned long long & 8 \\
signed long long & 8 \\
float & 4 \\
\hline
\end{tabular}
\end{table}
### 4.3 About PGI Compilers

#### Table 3. PGI Compiler Basics

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

### 4.3.1 Known Limitations

- 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 Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.
4.4 About the Cray Compiling Environment (CCE)

Table 4. Cray Compiler Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>PrgEnv-cray</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command:</td>
<td>ftn, cc, CC</td>
</tr>
<tr>
<td>Compiler-specific man pages:</td>
<td>crayftn(1), craycc(1), crayCC(1)</td>
</tr>
</tbody>
</table>

Note: Compiler-specific man pages are available only when the compiler module is loaded.

Online help: None provided


4.4.1 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.

4.5 About PathScale Compilers

Table 5. PathScale Compiler Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>PrgEnv-pathscale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command:</td>
<td>ftn, cc, CC</td>
</tr>
<tr>
<td>Compiler-specific man pages:</td>
<td>pathf95(1), pathf90(1), pathcc(1), pathCC(1), eko(7)</td>
</tr>
</tbody>
</table>

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/doc/version
4.5.1 Known Limitations

- The PathScale Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.

- The PETSc 3.1.0 library of parallel linear and nonlinear solvers does not support the PathScale Compiler Suite. Use PETSc 3.0.x instead.

4.6 About Intel Compilers

Table 6. Intel Compiler Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>PrgEnv-intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command:</td>
<td>ftn, cc, CC</td>
</tr>
<tr>
<td>Compiler-specific man pages:</td>
<td>ifort(1), fpp(1), icc(1), icpc(1)</td>
</tr>
</tbody>
</table>

**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 |

4.6.1 Known Limitations

- The Intel Compiler Suite 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 Compiler Suite does not support the UPC or Coarray Fortran parallel programming models.

4.7 About GNU Compilers

Table 7. GNU Compiler Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>PrgEnv-gnu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command:</td>
<td>ftn, cc, CC</td>
</tr>
<tr>
<td>Compiler-specific man pages:</td>
<td>gfortran(1), gcc(1), g++(1)</td>
</tr>
</tbody>
</table>

**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.

4.8 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 Supplement 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 2.1 or 2.2, the Linux system must be running SLES 10. If the Cray system is running CLE 3.0 or 3.1, 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 it will 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 Supplement Installation Guide*.

- You must install and configure modules on the Linux system.

After CADE, CADES, 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 or Cray XE6), 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)
- `xtpe-shanghai` (quad core)
- `xtpe-istanbul` (six cores)
- `xtpe-mc8` (eight cores)
- `xtpe-mc12` (twelve 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. 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.
Cray provides a large variety of libraries and functions 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 support all compilers currently supported on the Cray system, except where noted.

### 5.1 Scientific Libraries

**Table 8. LibSci Basics**

<table>
<thead>
<tr>
<th>Module:</th>
<th>xt-libsci</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man pages:</td>
<td>intro_libsci(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_blas1(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_blas2(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_blas3(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_blacs(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_lapack(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_scalapack(3s)</td>
</tr>
<tr>
<td></td>
<td>intro_irt(3),</td>
</tr>
<tr>
<td></td>
<td>intro_crafft(3s),</td>
</tr>
<tr>
<td></td>
<td>intro_fft(3),</td>
</tr>
<tr>
<td></td>
<td>intro_fft2(3),</td>
</tr>
<tr>
<td></td>
<td>intro_fft3(3)</td>
</tr>
</tbody>
</table>

*Note:* Library-specific man pages are available only when the associated module is loaded.

Cray LibSci is a collection of numerical routines optimized for best performance on Cray systems. All programming environment modules load `xt-libsci` by default. 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.

The Cray LibSci collection contains the following sub-libraries.

- **BLAS** (Basic Linear Algebra Subroutines)
- **BLACS** (Basic Linear Algebra Communication Subprograms)
- **LAPACK** (Linear Algebra Routines)
- **ScaLAPACK** (Scalable LAPACK)
- **IRT** (Iterative Refinement Toolkit)
5.1.1 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.
5.1.1.1 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 \texttt{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 \texttt{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:
  - \texttt{xtpe-barcelona} (quad core)
  - \texttt{xtpe-shanghai} (quad core)
  - \texttt{xtpe-istanbul} (six cores)
  - \texttt{xtpe-mc8} (eight cores)
  - \texttt{xtpe-mc12} (twelve cores)

  \textbf{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 \texttt{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 \texttt{OMP_NUM_THREADS}, as described in BLACS and ScaLAPACK on page 44.

- 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 \texttt{dgetrf()} function as follows:
  \begin{verbatim}
  dgetrf_(&uplo, &m, &n, a, &lda, ipiv, work, &lwork, &info);
  \end{verbatim}

- C programmers using the Fortran interface must order arrays in Fortran column-major order.
5.1.2 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.

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

5.1.2.1 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.
5.1.3 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 $\text{dgesv}()$ routine replaces either the LAPACK $\text{dgesv}()$ routine or the LAPACK $\text{dgetrf}()$ and $\text{dgetrs}()$ 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.
5.1.4 FFT

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.

5.1.4.1 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.

5.1.4.2 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:

```bash
% module swap fftw/3.2.1 fftw/2.1.5.1
```

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

### 5.1.4.3 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:

```bash
% 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 About the Fast_mv Library

#### Table 9. Fast_mv Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>libfast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man pages:</td>
<td>intro_fast_mv(3),</td>
</tr>
</tbody>
</table>

**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:

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

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

Table 10. PETSc Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>petsc, petsc-complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man pages:</td>
<td>intro_petsc(3s),</td>
</tr>
<tr>
<td>Note:</td>
<td>Library-specific man pages are available only when the associated module is loaded.</td>
</tr>
<tr>
<td>Web site:</td>
<td><a href="http://www.mcs.anl.gov/petsc/petsc-as/">http://www.mcs.anl.gov/petsc/petsc-as/</a></td>
</tr>
</tbody>
</table>

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.

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 3.1.

- 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 4.0. 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 interactive solvers. For further information, see http://crd.lbl.gov/~xiaoye/SuperLU/.
- SuperLU_dist 2.3. 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 3.1. 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 2.6.0b. 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/.

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. 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 11. Trilinos Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>trilinos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man pages:</td>
<td>intro_trilinos(1)</td>
</tr>
<tr>
<td>Web site:</td>
<td><a href="http://trilinos.sandia.gov/">http://trilinos.sandia.gov/</a></td>
</tr>
</tbody>
</table>

Note: Library-specific man pages are available only when the associated module is loaded.
Trilinos is a separate module that provides abstract, object-oriented interfaces to established libraries such as PETSc, 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 petsc, xt-libsci, and xt-asyncpe modules. Make certain these modules are loaded before using Trilinos.

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

```bash
% 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 12. MPT Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>xt-mpt, xt-mpich2, xt-shmem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man pages:</td>
<td>intro_mpi(3), intro_shmem(3)</td>
</tr>
</tbody>
</table>

**Note:** Library-specific man pages are available only when the associated module is loaded.

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 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 will 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 and Cray XT systems, and these differences are reflected in the functions that are available, and most visibly in the environment variables that are 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)` and `intro_shmem(3)` man pages.

Furthermore, there are differences in the ways that MPI and SHMEM are implemented in order to support code with static or dynamic libraries.

### 5.5.1 Static Libraries

If your code uses static libraries, load the `xt-mpt` module.

```bash
> module load xt-mpt
```

Regardless of whether you are working on an Cray XE and Cray XT system, always use the `xt-mpt` module with static libraries.

### 5.5.2 Dynamic Libraries

If your code uses dynamic, shared libraries, unload the `xt-mpt` module and then load either the `xt-mpich2` or `xt-shmem` module, depending on whether your code uses MPI or SHMEM message-passing.

```bash
> module unload xt-mpt
> module load [xt-mpich2|xt-shmem]
```

Regardless of whether you are working on an Cray XE and Cray XT system, always use either the `xt-mpich2` or `xt-shmem` module with dynamic libraries.
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**: an optional 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 (Deferred implementation)
- **STAT**: stack trace analysis tool (Deferred implementation)
6.1.1 Using lgdb

<table>
<thead>
<tr>
<th>Table 13. lgdb Basics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module: xt-lgdb</td>
</tr>
<tr>
<td>Command: lgdb</td>
</tr>
<tr>
<td>Man page: lgdb(1),</td>
</tr>
</tbody>
</table>

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 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)

#### Abnormal Termination Processing (ATP)

Abnormal Termination Processing (ATP) is an optional system that monitors user applications. When the `atp` module is loaded, 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) or alternatively with the file viewer `dotty`, which can be found on most Linux systems. The merged stack backtrace tree provides a concise yet comprehensive view of what the application was doing at the time of its termination.

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, if 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 (Deferred implementation)

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.

For more information, see: http://www.paradyn.org/mrnet/.

6.1.4 Using STAT (Deferred implementation)

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.

For more information, see: http://www.paradyn.org/STAT/STAT.html.
6.2 Using 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.

Example 5. 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 \texttt{l gdb} 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 \texttt{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, \texttt{atpMergedBT.dot}. The stack backtrace tree for the first process to die is sent to \texttt{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 55.

6.4 Using TotalView

Table 15. TotalView Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>xt-totalview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands:</td>
<td>totalview, totalviewcli</td>
</tr>
<tr>
<td>Man page:</td>
<td>totalview(1)</td>
</tr>
</tbody>
</table>

\textbf{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 6. 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 7. 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 8. 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 16. DDT Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>ddt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands:</td>
<td>ddt</td>
</tr>
<tr>
<td>Man page:</td>
<td>ddt(1),</td>
</tr>
</tbody>
</table>

**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 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.
Table 17. Performance Analysis Basics

<table>
<thead>
<tr>
<th>Module:</th>
<th>perftools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands:</td>
<td>pat_build, pat_report, app2</td>
</tr>
<tr>
<td>Man pages:</td>
<td>intro_craypat(1), pat_build(1), pat_report(1), pat_help(1), app2(1), intro_papi(3),</td>
</tr>
</tbody>
</table>

**Note:** Tool-specific man pages are available only when the associated module is loaded.

<table>
<thead>
<tr>
<th>Online help:</th>
<th>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 <code>pat_help</code> at the command line.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation:</td>
<td>Using Cray Performance Analysis Tools</td>
</tr>
</tbody>
</table>

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

Beginning with release 5.1.0, the Cray Performance Analysis Tools are bundled into a single module. 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.

### 7.1 Using CrayPat

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:

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

Alternatively:

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

Then link the object files to create the executable program:

```bash
> ftn -o executable sourcefile.o
```
7.1.1 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*.

7.1.2 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*.

7.1.3 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+PIDem-n.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.

### 7.1.4 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.

### 7.2 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*.

### 7.3 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 xt-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
xt-papi module.

```
> module unload perftools
> module load xt-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/](http://icl.cs.utk.edu/papi/).
blade
1) A field-replaceable physical entity. A service blade consists of AMD Opteron sockets, memory, Cray network application-specific integrated circuit (ASIC) chips, PCI cards, and a blade control processor. A compute blade consists of AMD Opteron sockets, memory, Cray network application-specific integrated circuit (ASIC) chips, and a blade control processor. 2) From a system management perspective, a logical grouping of nodes and blade control processor that monitors the nodes on that blade.

Catamount
The operating system kernel developed by Sandia National Laboratories and implemented to run on Cray XT single-core compute nodes. See also Catamount Virtual Node (CVN); compute node.

Catamount Virtual Node (CVN)
The Catamount kernel enhanced to run on dual-core Cray XT compute nodes.

CLE
The operating system for Cray XE and Cray XT systems.

CNL
The CLE compute node kernel. CNL provides a set of supported system calls. CNL provides many of the operating system functions available through the service nodes, although some functionality has been removed to improve performance and reduce memory usage by the system.

compute node
A node that runs application programs. A compute node performs only computation; system services cannot run on compute nodes. Compute nodes run a specified kernel to support either scalar or vector applications. See also node; service node.
**deferred implementation**
The label used to introduce information about a feature that will not be implemented until a later release.

**login node**
The service node that provides a user interface and services for compiling and running applications.

**module**
See *blade*.

**multicore**
A processor that combines multiple independent execution engines ("cores"), each with its own cache and cache controller.

**node**
For CLE systems, the logical group of processor(s), memory, and network components acting as a network end point on the system interconnection network.

**service node**
A node that performs support functions for applications and system services. Service nodes run a version of SLES and perform specialized functions. There are six types of predefined service nodes: login, IO, network, boot, database, and syslog.