<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.0     | December 2004  
Draft documentation to support Cray XT3 early-production systems. |
| 1.0     | March 2005  
Draft documentation to support Cray XT3 limited-availability systems. |
| 1.1     | June 2005  
Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.1 and UNICOS/lc 1.1 releases. |
# Contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Product Documentation</td>
<td>vii</td>
</tr>
<tr>
<td>Conventions</td>
<td>viii</td>
</tr>
<tr>
<td>Reader Comments</td>
<td>ix</td>
</tr>
</tbody>
</table>

## Introduction [1]
- The Cray XT3 Programming Environment | 1 |
- Documentation Included with This Release | 2 |

## Setting up the User Environment [2]
- Setting Up a Secure Shell | 5 |
  - RSA Authentication with a Passphrase | 5 |
  - RSA Authentication without a Passphrase | 6 |
  - Additional Information | 6 |
- Using Modules | 6 |
  - Modifying the PATH Environment Variable | 7 |
  - Software Locations | 7 |
  - Module Commands | 7 |

## Libraries and APIs [3]
- C Language Runtime Library | 11 |
- AMD Core Math Library | 11 |
- Cray XT3 LibSci Scientific Libraries | 11 |
- Cray MPICH2 Message Passing Library | 12 |
  - Cray MPICH2 Limitations | 12 |
  - Cray MPICH2 Programming Considerations | 13 |
  - MPI Environment Variables | 13 |
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample MPI Programs</td>
<td>13</td>
</tr>
<tr>
<td>Example 1: A Simple Work Distribution Program</td>
<td>13</td>
</tr>
<tr>
<td>Example 2: Combining Results from all Processors</td>
<td>15</td>
</tr>
<tr>
<td>Portals 3.3 Low-level Message-passing API</td>
<td>17</td>
</tr>
<tr>
<td>Cray Shared Memory Access (SHMEM) Library</td>
<td>17</td>
</tr>
<tr>
<td>Sample Cray SHMEM Programs</td>
<td>18</td>
</tr>
<tr>
<td>Example 3: Cray SHMEM put Function</td>
<td>18</td>
</tr>
<tr>
<td>Example 4: Cray SHMEM get() Function</td>
<td>20</td>
</tr>
<tr>
<td>Catamount Programming Considerations [4]</td>
<td>23</td>
</tr>
<tr>
<td>glibc Functionality</td>
<td>23</td>
</tr>
<tr>
<td>I/O Support in Catamount</td>
<td>24</td>
</tr>
<tr>
<td>Using the Lustre File System</td>
<td>25</td>
</tr>
<tr>
<td>Improving File I/O Bandwidth</td>
<td>25</td>
</tr>
<tr>
<td>Using Stride I/O functions</td>
<td>27</td>
</tr>
<tr>
<td>Timing Support in Catamount</td>
<td>27</td>
</tr>
<tr>
<td>Example 5: Using dcllock() to Calculate Elapsed Time</td>
<td>28</td>
</tr>
<tr>
<td>Signal Support in Catamount</td>
<td>28</td>
</tr>
<tr>
<td>Additional Programming Considerations</td>
<td>28</td>
</tr>
<tr>
<td>Compiler Overview [5]</td>
<td>31</td>
</tr>
<tr>
<td>Running an Application [6]</td>
<td>33</td>
</tr>
<tr>
<td>Monitoring the System</td>
<td>33</td>
</tr>
<tr>
<td>Using the yod Application Launcher</td>
<td>34</td>
</tr>
<tr>
<td>Controlling Node Allocation</td>
<td>34</td>
</tr>
<tr>
<td>Launching an MPMD Application</td>
<td>35</td>
</tr>
<tr>
<td>Example 6: Using a Loadfile</td>
<td>36</td>
</tr>
<tr>
<td>Managing Compute Node Processors from an MPI Program</td>
<td>36</td>
</tr>
<tr>
<td>Input and Output Modes under yod</td>
<td>36</td>
</tr>
<tr>
<td>Signal Handling under yod</td>
<td>36</td>
</tr>
</tbody>
</table>
## Contents

- **Associating a Project or Task with a Job Launch** ........................................... 37
- **Using PBS Pro** ........................................................................................................... 37
  - Submitting a PBS Pro Batch Job ............................................................................. 37
  - Using a Job Script ...................................................................................................... 38
    - Example 7: A Simple Job Script ............................................................................ 39
  - Getting Jobs Status .................................................................................................... 39
  - Removing a Job from the Queue .............................................................................. 40
  - Cray XT3 Specific PBS Pro Functions ........................................................................ 40
- **Running Applications in Parallel** ............................................................................ 40
  - Example 8: Running an MPI Program Interactively ................................................. 41
  - Example 9: Running an MPI Program under PBS Pro ........................................... 41
  - Example 10: Using a Script to Create and Run a Batch Job .................................... 42
- **Debugging an Application [7]** ................................................................................. 45
  - Troubleshooting Application Failures ....................................................................... 45
  - The TotalView Debugger ............................................................................................. 46
    - Overview of TotalView Features ............................................................................ 46
    - Differences in Functionality for Cray XT3 ............................................................ 46
    - Obtaining the TotalView Debugger ....................................................................... 47
    - Using The TotalView Debugger ............................................................................. 47
      - Example 11: Using TotalView ............................................................................. 47
- **Performance Analysis [8]** ..................................................................................... 49
  - Performance API (PAPI) ............................................................................................ 49
    - Using the High-level Interface ............................................................................... 49
      - Example 12: The High-level PAPI Interface ......................................................... 49
    - Using the Low-level Interface ............................................................................... 50
      - Example 13: The Low-level Interface .................................................................. 51
  - CrayPat Performance Analysis Tool ......................................................................... 52
    - Example 14: CrayPat Basics ................................................................................. 53
    - Example 15: Using Hardware Performance Counters ......................................... 58
The information in this preface is common to Cray documentation provided with this software release.

**Accessing Product Documentation**

With each software release, Cray provides books and man pages, and in some cases, third-party documentation. These documents are provided in the following ways:

- CrayDoc, the Cray documentation delivery system that allows you to quickly access and search Cray books, man pages, and in some cases, third-party documentation—Access this HTML and PDF documentation via CrayDoc at the following URLs:
  - The local network location defined by your system administrator
  - The CrayDoc public website: docs.cray.com

- Man pages—Access man pages by entering the `man` command followed by the name of the man page. For more information about man pages, see the `man(1)` man page by entering:
  
  ```
  % man man
  ```

- Third-party documentation not provided through CrayDoc—Access this documentation, if any, according to the information provided with that product.
# Conventions

These conventions are used throughout Cray documentation:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>This fixed-space font denotes literal items, such as file names, pathnames, man page names, command names, and programming language elements.</td>
</tr>
<tr>
<td>variable</td>
<td>Italic typeface indicates an element that you will replace with a specific value. For instance, you may replace <code>filename</code> with the name <code>datafile</code> in your program. It also denotes a word or concept being defined.</td>
</tr>
<tr>
<td>user input</td>
<td>This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Brackets enclose optional portions of a syntax representation for a command, library routine, system call, and so on.</td>
</tr>
<tr>
<td>...</td>
<td>Ellipses indicate that a preceding element can be repeated.</td>
</tr>
<tr>
<td>name(N)</td>
<td>Denotes man pages that provide system and programming reference information. Each man page is referred to by its name followed by a section number in parentheses.</td>
</tr>
</tbody>
</table>

Enter:

```bash
% man man
```

to see the meaning of each section number for your particular system.
Reader Comments

Contact us with any comments that will help us to improve the accuracy and usability of this document. Be sure to include the title and number of the document with your comments. We value your comments and will respond to them promptly. Contact us in any of the following ways:

E-mail:
docs@cray.com

Telephone (inside U.S., Canada):
1–800–950–2729  (Cray Customer Support Center)

Telephone (outside U.S., Canada):
+1–715–726–4993  (Cray Customer Support Center)

Mail:
Software Publications
Cray Inc.
1340 Mendota Heights Road
Mendota Heights, MN 55120–1128
USA
This guide is for application programmers and users of the Cray XT3 system. It describes the Cray XT3 programming environment products and related application development tools. In addition, it includes procedures and examples that show you how to set up your user environment and compile, launch, monitor, and optimize applications.

For an introduction to the Cray XT3 system, see the *Cray XT3 System Overview*; this user’s guide assumes you are familiar with the topics discussed in the system overview. System administrators should refer to *Cray XT3 System Management* for information on managing system resources.

**Note:** Functionality marked as deferred in this documentation is planned to be implemented in a later release.

### 1.1 The Cray XT3 Programming Environment

The Cray XT3 programming environment includes the following products and services:

- PGI compilers for Fortran, C, and C++ (see Chapter 5, page 31).
- Cray MPICH2, the Message-Passing Interface 2 (MPI-2) routines (see Section 3.4, page 12).
- Cray SHMEM logically shared, distributed memory access routines (see Section 3.5, page 17).
- AMD Core Math Library (ACML), which includes:
  - Level 1, 2, and 3 Basic Linear Algebra Subroutines (BLAS)
  - Linear Algebra (LAPACK) routines
  - Fast Fourier Transform (FFT) routines
    
    See Section 3.2, page 11 for further information.
- Cray XT3 LibSci scientific library, which includes:
  - ScaLAPACK, a set of LAPACK routines redesigned for use in MPI applications.
- BLACS, a set of communication routines used by ScaLAPACK and the user to set up a problem and handle the communications.

- SuperLU, a set of routines that solve large, sparse, nonsymmetric systems of linear equations.

See Section 3.3, page 11 for further information.

- A special port of the GNU C Library, *glibc* 2.3.2 routines for compute node applications (see Section 3.1, page 11).

- Performance API (PAPI) for measuring the efficiency of an application’s use of processor functions (see Section 8.1).

In addition to Programming Environment products, the Cray XT3 system provides these application development products and functions:

- The *yod* command (see Section 6.2, page 34) for running applications.

- Lustre parallel and UFS-like file systems (see Section 4.4, page 25).

- The *xtshowmesh* utility for determining the availability of batch and interactive compute nodes (see Section 6.1, page 33).

- Single-system view (SSV) commands (such as *xtps* and *xtkill*) for managing processes (see Appendix B, page 75).

- Portals, the low-level message-passing interface (see Section 3.4.5, page 17).

The following optional products are available for Cray XT3 systems:

- PBS Pro (see Section 6.3, page 37)

- CrayPat (see Section 8.2, page 52)

- Cray Apprentice² (see Section 8.3, page 64)

A special implementation of TotalView is available from Etnus, LLC (http://www.etnus.com). For more information, see Section 7.2, page 46.

### 1.2 Documentation Included with This Release

Table 1 lists the manuals and man pages that are provided with this release. All manuals are provided as PDF files and some are available as HTML files. You can view the manuals and man pages through the CrayDoc interface or move the files to another location, such as your desktop.
**Note:** You can use the Cray XT3 System Documentation Site Map on CrayDoc to link to all manuals and man pages included with this release.

<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray XT3 System Overview</td>
</tr>
<tr>
<td>Cray XT3 Systems Software Release Overview</td>
</tr>
<tr>
<td>Glossary of Cray XT3 Terms</td>
</tr>
<tr>
<td>PGI User’s Guide</td>
</tr>
<tr>
<td>PGI Fortran Reference</td>
</tr>
<tr>
<td>PGI Tools Guide</td>
</tr>
<tr>
<td>Cray MPICH2 man pages (read the Cray intro_mpi(1) man page first)</td>
</tr>
<tr>
<td>Cray SHMEM man pages (read intro_shmem(1) first)</td>
</tr>
<tr>
<td>AMD Core Math Library (ACML)</td>
</tr>
<tr>
<td>Cray XT3 LibSci man pages</td>
</tr>
<tr>
<td>SuperLU Users’ Guide</td>
</tr>
<tr>
<td>PBS Pro 5.3 Quick Start Guide, PBS-3BQ01</td>
</tr>
<tr>
<td>PBS Pro 5.3 User Guide, PBS-3BU01</td>
</tr>
<tr>
<td>PBS Pro 5.3 External Reference Specification, PBS-3BE01</td>
</tr>
<tr>
<td>PAPI User’s Guide</td>
</tr>
<tr>
<td>PAPI Programmer’s Reference</td>
</tr>
<tr>
<td>PAPI Software Specification</td>
</tr>
<tr>
<td>PAPI man pages</td>
</tr>
<tr>
<td>SUSE Linux man pages</td>
</tr>
<tr>
<td>UNICOS/lc man pages (start with intro_xt3(1))</td>
</tr>
</tbody>
</table>

---

1 PBS Pro is an optional product available from Cray Inc.
Additional sources of information are available at:

- For more information about using the PGI compilers, see The Portland Group website at http://www.pgroup.com, which answers FAQs and provides access to developer forums.


- The ScalAPACK User’s Guide and ScaLAPACK tutorial are available in HTML format at http://www.netlib.org/scalapack/slug/.

- Additional SuperLU documentation is available at http://crd.lbl.gov.

- For additional information about PAPI, see http://icl.cs.utk.edu/papi.
Setting up the User Environment

Configuring your user environment on a Cray XT3 system is similar to what you would do on a typical Linux workstation. However, there are Cray XT3 specific steps that you need to take before you begin developing applications.

2.1 Setting Up a Secure Shell

Cray XT3 systems use `ssh` and `ssh`-enabled applications such as `scp` for secure, password-free remote access to the login nodes.

Before you can use the `ssh` commands, you will need to generate an RSA authentication key. There are two methods of passwordless authentication—with or without a passphrase. Although both methods are described here, you will need to use the latter method to access the compute nodes through a script or when using a single system view (SSV) command. For information about single-system view commands, see Appendix B, page 75.

2.1.1 RSA Authentication with a Passphrase

To enable `ssh` with a passphrase, complete the following steps.

1. Generate the RSA keys by entering the following command:

   % ssh-keygen -t rsa

   and follow the prompts. You will be asked to supply a passphrase.

2. The public key is stored in your `$HOME/.ssh` directory. Enter the following command to copy the key to your home directory on the remote host(s):

   % scp $HOME/.ssh/id_rsa.pub
   
   `username@system_name:/home/users/username/.ssh/`

3. Connect to the remote host by typing the following commands.

   If you are using a C shell, enter:

   % eval ‘ssh-agent’

   % ssh-add
If you are using a Bourne shell, enter:

```
$ eval 'ssh-agent -s'
$ ssh-add
```

Enter your passphrase when prompted, followed by:

```
% ssh remote_host_name
```

### 2.1.2 RSA Authentication without a Passphrase

To enable `ssh` without a passphrase, complete the following steps.

1. Generate the RSA keys by typing the following command:

```
% ssh-keygen -t rsa -N ""
```

and following the prompts.

2. The public key is stored in your `$HOME/.ssh` directory. Type the following command to copy the key to your home directory on the remote host(s):

```
% scp $HOME/.ssh/id_rsa.pub

username@system_name:/home/users/username/.ssh/
```

**Note:** This step is not required if your home directory is shared.

3. Connect to the remote host by typing the following command:

```
% ssh remote_host_name
```

### 2.1.3 Additional Information

For more information about setting up and using a secure shell, see the `ssh(1)`, `ssh-keygen(1)`, `ssh-agent(1)`, `ssh-add(1)`, and `scp(1)` man pages.

### 2.2 Using Modules

Cray XT3 systems use modules in the user environment to support multiple versions of software, such as compilers, and to create integrated software packages. As new versions of the supported software and associated man pages become available, they are added automatically to the programming environment, while earlier versions are retained to support legacy applications. By specifying the module to load, you can choose the default version of an application or another version.
Modules also provide a simple mechanism for updating certain environment variables, such as `PATH`, `MANPATH`, and `LD_LIBRARY_PATH`. In general, you should make use of the modules system rather than embedding specific directory paths into your startup files, makefiles, and scripts.

The following paragraphs describe the processes you follow to manage your user environment.

### 2.2.1 Modifying the PATH Environment Variable

Do not reinitialize the system-defined `PATH`. The following example shows how to modify it for a specific purpose—in this case to add `$HOME/bin` to the path.

If you are using `csh`, enter:

```
% set path = ($path $HOME/bin)
```

If you are using `bash`:

```
$ export PATH=$PATH:$HOME/bin
```

### 2.2.2 Software Locations

On a typical Linux system, compilers and other software packages are located in the `/bin` or `/usr/bin` directories. However, on a Cray XT3 system these files are in versioned locations under the `/opt` directory.

Cray software is self-contained and is installed as follows:

- Base prefix: `/opt/pkgname/pkgversion/`, such as `/opt/xt-pe/1.1`
- Package environment variables: `/opt/pkgname/pkgversion/var`
- Package configurations: `/opt/pkgname/pkgversion/etc`

**Note:** To run a Programming Environment product, specify the command name (and arguments) only; do not enter an explicit path to the Programming Environment product. Likewise, job files and makefiles should not have explicit paths to Programming Environment products embedded in them.

### 2.2.3 Module Commands

To run the module utility, type:

```
% module option args
```
The module command accepts the following options:

- **avail**
  Displays all the modules that are available on the system. When there is more than one version of a module, the default version is denoted by *(default)*.

- **list**
  Displays all the modules that are currently loaded into your user environment.

- **add|load modulename**
  Loads the specified module into your user environment.

- **rm modulename|unload modulename**
  Unloads the specified module from your user environment. Before loading a module that replaces another version of the same package, you should always unload the module that is being replaced.

- **clear modulename**
  Unloads all loaded modules.

- **display modulename|show modulename**
  Shows the changes that the specified module will make in your environment (for example, what will be added to the PATH and MANPATH environment variables).

- **switch modulename/current_version modulename/new_version**
  Replaces the currently loaded version of a module with a different version. When the new version is loaded, the man pages for the specified software will also be updated.

- **help**
  Displays online help for the module command. The following command will provide more detailed information on the specified module:

  ```
  % module help modulename
  ```

When you log in, the PrgEnv module you will need to build and run applications on the system may be automatically initialized and loaded. To check if the module has been loaded, enter:

```
% module list
```

If PrgEnv has not been loaded enter:

```
% module load PrgEnv
```
If the correct modules are loaded, you need not use the `module` command until new versions become available.

For further information on the module utility, see the `module(1)` and `modulefile(4)` man pages.
This chapter describes the libraries and APIs that are available to application developers.

3.1 C Language Runtime Library

A subset of the GNU C runtime library, glibc version 2.3.2, is implemented on Catamount (see Section 4.2, page 23 and Appendix A, page 69 for more information).

Note: The Cray XT3 implementation of glibc for compute nodes includes a simple implementation of malloc(), optimized for the lightweight microkernel and large memory allocations. This version of malloc() is the default. To override the default and use the standard glibc implementation, include -lgmalloc on the compiler command line (see Section 4.7, page 28).

3.2 AMD Core Math Library

The Cray XT3 programming environment includes the 64-bit AMD Core Math Library (ACML). The ACML includes:

- Level 1, 2, and 3 Basic Linear Algebra Subroutines (BLAS)
- A full suite of Linear Algebra (LAPACK) routines
- A suite of Fast Fourier Transform (FFT) routines in single-, double-, single-complex, and double-complex data types

Note: The compiler drivers automatically load and link to the ACML library, libacml.a. It is not necessary to load and link manually as described in the ACML documentation.

3.3 Cray XT3 LibSci Scientific Libraries

The Cray XT3 programming environment includes a scientific libraries package, Cray XT3 LibSci. Cray XT3 LibSci provides ScaLAPACK, BLACS, and SuperLU routines.

The ScaLAPACK library contains parallel versions of a set of LAPACK routines. The BLACS package is a set of communication routines used by ScaLAPACK and
the user to set up a problem and handle the communications. Both packages are designed to be used in MPI applications.

The SuperLU library is designed to solve large, sparse nonsymmetric systems of linear equations. The Cray XT3 LibSci package contains only the distributed-memory parallel version of SuperLU. The library is written in C but can be called from programs written in either C or Fortran.

3.4 Cray MPICH2 Message Passing Library

This release of MPI-2 derives from MPICH2 and implements the MPI-2 standard, except for support of spawn functions. It also implements the MPI 1.2 standard, as documented by the MPI Forum in the spring 1997 release of MPI: A Message Passing Interface Standard.

The Cray MPICH2 message-passing library is implemented on top of the Portals low-level message-passing engine. For more information about using Cray MPICH2 functions, see the MPI man pages, starting with intro_mpi(1).

3.4.1 Cray MPICH2 Limitations

There is a name conflict between stdio.h and the MPI C++ binding in relation to the names SEEK_SET, SEEK_CUR, and SEEK_END. If your application does not reference these names, you can work around this conflict by using the compiler flag -DMPICH_IGNORE_CXX_SEEK. If your application does require these names, as defined by MPI, undefine the names (#undef SEEK_SET, for example) prior to including mpi.h. Alternatively, if the application requires the stdio.h naming, your application should include mpi.h before stdio.h or the iostream routine.

The following process-creation functions are not supported and if used will generate aborts at runtime:

- MPI_Close_port and MPI_Open_port
- MPI_Comm_accept
- MPI_Comm_connect and MPI_Comm_disconnect
- MPI_Comm_spawn and MPI_Comm_spawn_multiple
- MPI_Comm_get_attr - with attribute MPI_UNIVERSE_SIZE
- MPI_Comm_get_parent
• MPI_Lookup_name
• MPI_Publish_name and MPI_Unpublish_name

The MPI_LONG_DOUBLE data type is not supported.

3.4.2 Cray MPICH2 Programming Considerations

There is a name conflict between stdio.h and the MPI C++ binding in relation to the names SEEK_SET, SEEK_CUR, and SEEK_END. If your application does not reference these names, you can work around this conflict by using the compiler flag -DMPICH_IGNORE_CXX_SEEK. If your application does require these names, as defined by MPI, #undef the names (#undef SEEK_SET, for example) prior to including mpi.h. Alternatively, if the application requires the stdio.h naming, your application should include mpi.h before stdio.h or iostream.

3.4.3 MPI Environment Variables

For information about MPI environment variables, refer to the intro_mpi(1) man page.

3.4.4 Sample MPI Programs

The following sample applications demonstrate basic MPI functionality in a program built for both Fortran and C components. See Chapter 5, page 31 for a description of the commands used to invoke the compilers.

Example 1: A Simple Work Distribution Program

This example uses MPI solely to identify the processor associated with each process, then selects the work to be done by each processor. Each processor writes its output directly to stdout.

Source code of Fortran main program (prog.f90):

program main
include ’mpif.h’ ! Required

 call MPI_Init(ierr) ! Required
 call MPI_Comm_rank(MPI_COMM_WORLD,mype,ierr)
 call MPI_Comm_size(MPI_COMM_WORLD,npes,ierr)

 print *,’hello from pe’,mype,’ of’,npes
do i=1+mype,1000,npes ! Distribute the work
call work(i,mype)
enddo

call MPI_Finalize(ierr) ! Required
end

The C function work.c processes a single item of work.

Source code of work.c:

```c
void work_(int *N, int *MYPE)
{
    int n=*N, mype=*MYPE;

    if (n == 42) {
        printf("PE %d: sizeof(long) = %d\n",mype,sizeof(long));
        printf("PE %d: The answer is: %d\n",mype,n);
    }
}
```

Compile work.c:

```
% cc -c work.c
```

Compile prog.f90, load work.o, and create executable program1:

```
% ftn -o program1 prog.f90 work.o
```

Run program1 on 2 nodes:

```
% yod -np 2 program1
```

Output from program1:

```
hello from pe 0 of 2
hello from pe 1 of 2

PE 1: sizeof(long) = 8
PE 1: The answer is: 42
```

Note: The output refers to a node as a "pe" or "PE" (processing element).

The C equivalent of prog.f90 is:

```c
#include <stdio.h>
#include <mpi.h>
```
main(int argc, char **argv)
{
    int i, mype, npes;

    MPI_Init(&argc, &argv);  // Required
    MPI_Comm_rank(MPI_COMM_WORLD, &mype);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);

    printf("hello from pe %d of %d\n", mype, npes);

    for (i = 1 + mype; i <= 1000; i += npes)  // distribute the work
        work_(&i, &mype);

    MPI_Finalize();  // Required
}

**Example 2: Combining Results from all Processors**

In this example, MPI is also used to combine the results from each processor; only processor 0 writes the output to stdout.

**Source code of Fortran main program (prog.f90):**

```fortran
program main
    include 'mpif.h'
    integer work

    call MPI_Init(ierr)
    call MPI_Comp_rank(MPI_COMM_WORLD, mype, ierr)
    call MPI_Comm_size(MPI_COMM_WORLD, npes, ierr)

    n = 0
    do i = 1 + mype, 1000, npes
        n = n + work(i, mype)
    enddo

    call MPI_Reduce(n, nres, 1, MPI_INTEGER, MPI_SUM, 0, MPI_COMM_WORLD, ierr)
    if (mype.eq.0) print *, 'PE', mype, ': The answer is:', nres

    call MPI_Finalize(ierr)
end
```
The C function `work.c` processes a single item of work.

Source code of `work.c`:

```c
int work_(int *N, int *MYPE)
{
    int n=*N, mype=*MYPE;
    int mysum=0;

    if (n == 12) mysum+=n;
    if (n == 68) mysum+=n;
    if (n == 94) mysum+=n;
    if (n == 120) mysum+=n;
    if (n == 19) mysum-=n;
    if (n == 103) mysum-=n;
    if (n == 53) mysum-=n;
    if (n == 77) mysum-=n;
    return mysum;
}
```

Compile `work.c`, compile `prog.f90`, and run executable `program1`:

```
% cc -c work.c
% ftn -o program1 prog.f90 work.o
% yod -np 3 program1
```

The output will be similar to this:

```
PE 2: sizeof(long) = 8
PE 2: The answer is: 42
PE 0 : The answer is: -989531532
```

If you want to use a C main program instead of the Fortran main program, compile `prog.c`:

```c
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
{
    int i,mype,npes,n=0,res;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&mype);
```
MPI_Comm_size(MPI_COMM_WORLD,&npes);

for (i=mype; i<1000; i+=npes) {
    n += work_(&i, &mype);
}

MPI_Reduce(&n,&res,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
if (!mype) {
    printf("PE %d: The answer is: %d\n",mype,res);
}
MPI_Finalize();

and link it with work.o:

% cc -o program1 prog.c work.o

To run executable program1 on 6 nodes, enter:

% yod -np 6 program1

### 3.4.5 Portals 3.3 Low-level Message-passing API

The Portals message-passing API is split between user-level and microkernel-level functions on the Opteron processor and firmware on the SeaStar chip. Applications communicating at the user level link to the Cray MPICH2 or Cray SHMEM library. The Portals interface is transparent to the application programmer.

### 3.5 Cray Shared Memory Access (SHMEM) Library

The Cray SHMEM library is a set of logically shared, distributed memory access routines.\(^1\) Cray SHMEM routines are similar to MPI routines; they pass data between cooperating parallel processes.

Cray SHMEM routines can be used in programs that perform computations in separate address spaces and that explicitly pass data via puts and gets to and from different processing elements in the program. Cray SHMEM routines can be called from Fortran and C programs and can be used either by themselves or in conjunction with MPI functions.

---

\(^1\) (Deferred implementation) Support of Cray SHMEM atomic operations is deferred.
To build, compile, and run Cray SHMEM applications, you need to:

- Call `start_pes(int npes)` as the first Cray SHMEM call and `shmem_finalize()` as the last Cray SHMEM call.
- Include `-lsma` on the compiler command line to link the Cray SHMEM library routines:

  ```
  % cc -o shmem1 -lsma shmem1.c
  % ftn -o shmem2 -lsma shmem2.f90
  ```

See the `intro_shmem(1)` man page for a list of supported Cray SHMEM functions.

### 3.5.1 Sample Cray SHMEM Programs

The following examples demonstrate basic Cray SHMEM functions. See Chapter 5, page 31 for a description of the commands used to invoke the compilers.

**Example 3: Cray SHMEM put Function**

Source code of C program (`shmem1.c`):

```c
/*
 * simple put test
 */

#include <stdio.h>
#include <stdlib.h>
#include <mpp/shmem.h>

/* Dimension of source and target of put operations */
#define DIM 1000000

long target[DIM];
long local[DIM];

main(int argc, char **argv)
{
    register int i;
    int my_partner, my_pe;

    /* Prepare resources required for correct functionality of SHMEM on XT3. Alternatively, shmem_init() could */
be called. */
start_pes(0);

for (i=0; i<DIM; i++) {
    target[i] = 0L;
    local[i] = shmem_my_pe() + (i * 10);
}

my_pe = shmem_my_pe();

if(shmem_n_pes()%2) {
    if(my_pe == 0) printf("Test needs even number of processes\n");
    /* Clean up resources before exit. */
    shmem_finalize();
    exit(0);
}

shmem_barrier_all();

/* Test has to be run on two procs. */
my_partner = my_pe % 2 ? my_pe - 1 : my_pe + 1;

shmem_put64(target,local,DIM,my_partner);

/* Synchronize before verifying results. */
shmem_barrier_all();

/* Check results of put */
for(i=0; i<DIM; i++) {
    if(target[i] != (my_partner + (i * 10))) {
        fprintf(stderr,"FAIL (1) on PE %d target[%d] = %d (%d)\n",
                shmem_my_pe(), i, target[i],my_partner+(i*10));
        shmem_finalize();
        exit(-1);
    }
}

printf(" PE %d: Test passed.\n",my_pe);

/* Clean up resources. */
shmem_finalize();}
Compile `shmem1.c` and create executable `shmem1`:

```
% cc -o shmem1 -lsma shmem1.c
```

Run the executable:

```
% qsub -I -l size=4
% yod -np 4 shmem1
```

The output will be similar to this:

```
PE 0: Test passed.
PE 3: Test passed.
PE 2: Test passed.
PE 1: Test passed.
```

**Example 4: Cray SHMEM get() Function**

*Note:* The Fortran module for Cray SHMEM is not supported. Use the `INCLUDE 'mpp/shmem.fh'` statement instead.

Source code of Fortran program (`shmem2.f90`):

```
PROGRAM REDUCTION

INCLUDE 'mpp/shmem.fh'

REAL VALUES, SUM
COMMON /C/ VALUES
REAL WORK

CALL START_PES(0)
VALUES = MY_PE()
CALL SHMEM_BARRIER_ALL! Synchronize all PEs
SUM = 0.0
DO I = 0, NUM_PES()-1
   CALL SHMEM_GET(WORK, VALUES, 1, I) ! Get next value
   SUM = SUM + WORK ! Sum it
ENDDO

PRINT*, 'PE ', MY_PE(),' COMPUTEDSUM=',SUM

CALL SHMEM_BARRIER_ALL
CALL SHMEM_FINALIZE

END
```
Compile `shmem2.f90` and create executable `shmem2`:

```
% ftn -o shmem2 -lsma shmem2.f90
```

Run the executable:

```
% yod -np 2 shmem1
```

The output will be similar to this:

```
PE      1  COMPUTEDSUM=  1.000000
PE      0  COMPUTEDSUM=  1.000000
```
This chapter provides information you need to take into consideration when writing programs that will run on compute nodes.

4.1 PGI 6.0 Compilers

Object and module files created using PGI 6.0 compilers are incompatible with object files from previous releases.

The `-i8` option can make programs incompatible with MPI and the ACML math library. Typically, use of any INTEGER*8 array size argument can cause failures with these libraries.

4.2 glibc Functionality

Because the Catamount microkernel is designed specifically to provide critical support to high-speed computational applications, its functionality is limited in certain areas where the service nodes are expected to take over. In particular, glibc on Catamount does not support:

- Pipes, sockets, remote procedure calls, or other TCP/IP communication. The Cray MPICH2 and Cray SHMEM message passing interfaces and the underlying Portals interface are the only communication mechanisms.
- Dynamic process control (such as `exec`, `popen`, `fork`, or `system` library calls).
- Dynamic loading of executable code.
- Threading.
- The `/proc` files such as `cpuinfo` and `meminfo`. (These files contain information about your login node.)
- The `ptrace` system call.
- The `mmap` function. (If `mmap` is called, a skeleton function returns -1.) You should use `malloc` instead of `mmap` if the `mmap` call is using the `MAP_ANONYMOUS` flag (`malloc` is not an appropriate replacement for `mmap` calls that use the `MAP_FIXED` or `MAP_FILE` flag). If you do use `malloc`, be
aware that you may have to resolve data alignment issues. See the malloc(3)
man page for details.

- The profil function.
- Any of the getpwd family of library calls.
- Terminal control.
- Any functions that require a daemon.
- Any functions that require a database, such as ndb. For example, there is no
  support for the uid and gid family of queries that are based on the ndb.
- There is limited support for signals and ioctl().

Appendix A, page 69 lists the glibc functions that Catamount supports. The
glibc functions that Catamount does not support are so noted in their man
pages.

### 4.3 I/O Support in Catamount

I/O support for compute node applications is limited. The only operations
allowed are Fortran, C, and C++ I/O calls, Cray MPICH2 and Cray SHMEM
I/O functions, and the underlying Catamount (libsysio) and Lustre (liblustre)
functions.

Application programmers should keep in mind the following behavior:

- I/O is offloaded to the service I/O nodes. The yod application launcher
  handles stdin, stderr, and stdout. For more information, see Section
  6.2.4, page 36.

- Calling an I/O function such as open with a bad address will cause the
  application to fail with a page fault. On the service nodes, a bad address will
  cause the function to set errno = EFAULT and return −1.

- Catamount does not support I/O on named pipes.

- By default stdio is unbuffered. Under Catamount, this imposes a bandwidth
  limitation of approximately 10 bytes per second because read and write calls
  are being offloaded to yod. To improve performance, call setvbuf() to
  buffer stdin input or stdout/stderr output.
The following program improves the performance of the `printf()` loop by using `setvbuf()` with the mode of `_IOFBF` (fully buffered) and a buffer size of 1024:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int i, bsize, count;
    char *buf;
    i = 1;
    bsize = (i < argc) ? atoi(argv[i++]) : 1024;
    count = (i < argc) ? atoi(argv[i++]) : 1024;

    if(bsize > 0) {
        buf = malloc(bsize);
        setvbuf(stdout, buf, _IOFBF, bsize);
    }

    for(i=0; i < count; i++) {
        printf("this is line %5d\n", i);
    }

    exit(0);
}
```

### 4.4 Using the Lustre File System

If your application uses the Lustre parallel file system, there are some actions you need to perform and some options you can use to improve performance.

#### 4.4.1 Improving File I/O Bandwidth

You can realize high bandwidth file I/O by directing file operations to paths within a Lustre mount point. To do this, complete the following steps:

1. Link your application to the Lustre library. There are two options.
   a. Load the Lustre module:

```
% module load xt-lustre-ss
```
b. Or include `-llustre` on the compiler command line:

```
% cc -o my_lustre_app -llustre my_lustre_app.c
```

2. Send I/O through the Lustre library directly to a Lustre file system. To do this, your application must direct file operations to paths within a Lustre mount point. To determine the Lustre mount points as seen by Lustre applications, search the `/etc/sysio_init` file for the string `llite`:

For example, enter:

```
% grep llite /etc/sysio_init
```

Your output will be similar to this:

```
{creat, ft=file,nm="/lus/nid00007/.mount",pm=0644, str="llite:7:/nid00007-mds/client"}
{creat, ft=file,nm="/lus/nid00135_a/.mount",pm=0644, str="llite:135:/nid00135_mdsa/client"}
```

In is example, the mount points are:

- `/lus/nid00007`  
- `/lus/nid00135_a`

3. Verify that your application is properly linked to the Lustre library. Search for symbols prefixed with the string `llu_`:

For example, enter:

```
% nm my_lustre_app | grep llu
```

Your output will be similar to this:

```
000000000021acb0 t llu_ap_completion
000000000021ab26 t llu_ap_fill_obdo
000000000406a60 d llu_async_page_ops
```

4. Verify that a Lustre file system is mounted on a Linux node:

For example, enter:

```
% df -t lustre
```

Your output will be similar to this:

```
Filesystem  1K-blocks Used Available Use% Mounted on
7:/nid00007-mds/client  846749008 93780876  709955704 12% /lus/nid00007
135:/nid00135_mdsa/client 3289403772 2651208 3119660552  1% /lus/nid00135
```
4.4.2 Using Stride I/O functions

You can improve file I/O performance of C and C++ programs by using the readx(), writex(), ireadx(), and iwritex() stride I/O functions. See the readx(2), writex(2), ireadx(2), and iwritex(2) man pages for details.

4.5 Timing Support in Catamount

Catamount supports the following timing functions:

- Interval timer. Catamount supports the setitimer ITIMER_REAL function. It does not support the ITIMER_VIRTUAL or the ITIMER_PROF function. Also, Catamount does not support the getitimer() function.

- CPU timers. Catamount supports the getrusage() and cpu_time() functions. For C and C++ programs, getrusage() returns the current resource usages of either RUSAGE_SELF or RUSAGE_CHILDREN. The Fortran cpu_time(secs) intrinsic subroutine returns the processor time, where secs is real4 or real8. The magnitude of the value returned by cpu_time() is not necessarily meaningful. You call cpu_time() before and after a section of code; the difference between the two times is the CPU time used in seconds.

- Elapsed time counter. Use the dclock() function or MPI_Wtime() functions to calculate elapsed time. The etime() function is not supported.

The dclock() value rolls over approximately every 14 years and is expected to have an accuracy of 100 nanoseconds on each node.

Note: The dclock() function is based on the configured processor frequency, which may vary slightly from the actual frequency. Currently, the clock frequency is not calibrated. Further, the difference between configured and actual frequency may vary slightly from processor to processor. Because of these two factors, accuracy of the dclock() function may be off by as much as +/-50 microseconds/second or 4 seconds/day.

The MPI_Wtime() function returns the elapsed time. The MPI_Wtick() function returns the resolution of MPI_Wtime() in seconds.
Example 5: Using dclock() to Calculate Elapsed Time

The following example uses the dclock() function to calculate the elapsed time of a program segment.

Source code of dclock.c:

```c
#include <catamount/dclock.h>

main()
{
    double start_time, end_time, elapsed_time;
    start_time = dclock();
    sleep(5);
    end_time = dclock();
    elapsed_time = end_time - start_time;
    printf("Elapsed time = %f\n", elapsed_time);
}
```

Compile dclock.c and create executable dclock:

```bash
% cc -o dclock dclock.c
```

Run the program:

```bash
% yod dclock
```

Program output:

```
Elapsed time = 5.000008
```

4.6 Signal Support in Catamount

See Section 6.2.5, page 36 for a description of how yod propagates signals to running applications.

4.7 Additional Programming Considerations

- By default, when an application fails on Catamount, only one core file, that of the first failing processing, is generated. See the core(5) man page for information about overriding the defaults. Use caution with the overrides because dropping core files from all processes is not scalable.

- The Catamount getpagesize() function returns 4 KB. Although the system uses 2 MB pages in many of its memory sections, always assuming a 4 KB page size is a more robust approach.
• Because a Catamount application has dedicated use of the processor and physical memory on a compute node, many resource limits return RLIM_INFINITY. Keep in mind that while Catamount itself has no limitation on file size or the number of open files, the specific file systems on the Linux service partition may have limits that are unknown to Catamount.

• Catamount provides a custom implementation of the malloc() function. This implementation is tuned to Catamount’s non-virtual-memory operating system and favors applications allocating large, contiguous data arrays. The function uses a first-fit, last-in-first-out (LIFO) linked list algorithm. See the heap_info(3) man page for gathering statistics on memory usage. An application can use the glibc malloc() function instead of the custom malloc() by adding -lgmalloc to the compiler command line.

• On Catamount, the setrlimit() function always returns success when given a valid resource name and a non-NULL pointer to an rlimit structure. The rlimit value is never used because Catamount gives the application dedicated use of the processor and physical memory.
The Cray XT3 programming environment includes PGI Fortran, C and C++ compilers from STMicroelectronics for developing applications.

You access the PGI compilers through compiler drivers. The Cray XT3 compiler drivers perform the necessary initializations and load operations, such as linking in the header files and system libraries (such as libc.a and libmpich.a) before invoking the PGI compilers.

The Cray XT3 programming environment provides Fortran, C, and C++ compilers for building applications. The syntax for invoking the compiler drivers is:

```bash
% compiler_command options filename,...
```

For example, to compile `prog1.f90` and create default executable `a.out`, enter:

```bash
% ftn prog1.f90
```

The commands for invoking the compilers and the source file extensions are:

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Command</th>
<th>Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>C compiler</td>
<td>cc</td>
<td><code>filename.c</code></td>
</tr>
<tr>
<td>C++ compiler</td>
<td>CC</td>
<td><code>filename.c</code></td>
</tr>
<tr>
<td>Fortran compiler for Fortran 90 and Fortran 95</td>
<td>ftn</td>
<td><code>filename.f</code> (fixed source)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>filename.F90, filename.F95</code> (free source)</td>
</tr>
<tr>
<td>FORTRAN 77 compiler</td>
<td>f77</td>
<td><code>filename.f77</code></td>
</tr>
</tbody>
</table>

**Note:** To invoke the PGI compiler for all applications, including MPI applications, use either the `cc`, `CC`, `ftn`, or `f77` command. If you invoke a compiler directly by using a command such as `mpicc`, the resulting executable will not run on a Cray XT3 system.

Examples of compiler commands:

```bash
% cc -c myCprog.c
```
% CC -o my_executable myprogl.o myCCprog.C

% ftn -o sample1 sample1.f90

For examples of compiler command usage, see Section 3.4.4, page 13.

For more information on using the compiler commands, see the following man pages: cc(1), CC(1), ftn(1), and f77(1) and the PGI manuals (see Section 1.2, page 2).

To verify that you are using the correct version of a compiler, enter the cc -V, CC -V, or ftn -V.
This chapter describes the ways to launch an application on a Cray XT3 system, how to request compute nodes, and how to monitor the system.

The Cray XT3 system has been configured with a given number of interactive job processors and a given number of batch processors. A job that is launched from the command line will be sent to the interactive processors. If there are not enough processors available to handle the job, the command fails and an error message is displayed. Similarly, a job that is submitted as a batch process can use only the processors that have been allocated to the batch subsystem. If a job requires more processors than have been allocated for batch processing, it will never exit the batch queue.

Note: At any time, the system administrator can change the designation of any node from interactive to batch or vice versa. However, this will not affect jobs already running on those nodes; it will apply only to jobs that are in the queue and to subsequent jobs.

### 6.1 Monitoring the System

Before launching a job, enter the `xtshowmesh` command. The `xtshowmesh` utility displays the status of the compute and service processors—whether they are up or down, allocated to interactive or batch processing, and if they are free or in use. Each character in the display represents a single node.

Note: If `xtshowmesh` indicates that no compute nodes have been allocated for interactive processing, you can still run your job interactively by entering the PBS Pro `qsub -I` command and then, when your job has been queued, `yod` commands.

The following example shows a segment of output from `xtshowmesh`:

```bash
% xtshowmesh

Compute Processor Allocation Status as of Fri May 13 14:36:10 2005

<table>
<thead>
<tr>
<th>Cabinet 0</th>
<th>Cabinet 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1 2</td>
</tr>
</tbody>
</table>

Node-> 012345678901234567890123 012345678901234567890123
Row 0 LLLLaaaaaaaaaaaaaaaa||| LLLL|||A|A|
Row 1  aaaaXaaaaaaaaaaaaaaa||| A|A|
```
Legend:
- nonexistent node
- L unallocated Linux node
- : free interactive compute node
- A allocated, but idle compute node
- | free PBS compute node
- ? suspect compute node
- X failed compute node

Available compute nodes: 0 interactive, 90 batch

YODS LAUNCHED ON CATAMOUNT NODES

<table>
<thead>
<tr>
<th>Job ID</th>
<th>User</th>
<th>Size</th>
<th>Start</th>
<th>yod command line and arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>a 21251</td>
<td>abc1234</td>
<td>64</td>
<td>May 13 14:10:13</td>
<td>yod -sz 64 ./app999</td>
</tr>
<tr>
<td>b 21253</td>
<td>xyz777</td>
<td>1</td>
<td>May 13 14:20:32</td>
<td>yod -D all -sz 1./app777</td>
</tr>
</tbody>
</table>

Note: For systems running a large number of jobs, more than one character may be used to designate jobs.

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

### 6.2 Using the **yod** Application Launcher

The **yod** utility launches applications on compute nodes. When you start a **yod** process, the application launcher coordinates first with the Compute Processor Allocator (CPA) to allocate nodes for the application, then uses Process Control Threads (PCTs) to transfer the executable across the system interconnection network to the compute nodes. While the application is running, **yod** provides I/O services for the application, propagates signals, and participates in cleanup when the application terminates.

The topics in this section describe some of the commonly used **yod** options. For more information, see the **yod**(1) man page.

#### 6.2.1 Controlling Node Allocation

When launching an application with **yod**, you can specify the number of nodes to allocate to an application and allow applications to share nodes.
Use the following command to specify the number of processors to allocate:

```
% yod -size n [other arguments] program_name
```

where \( n \) is the number of processors.

**Note:** The `-size`, `-sz`, and `-np` options are synonymous.

The default behavior of `yod` is to allow only one process per node. The `-share` option allows more than one `yod` instance with the same owner to share a set of nodes. Use the following command with each invocation of `yod` to allow subsequent instances of `yod` (run by the same user) to share nodes:

```
% yod -share other_arguments program_name
```

Be aware that each node does its own scheduling and jobs are not gang scheduled across nodes. Also, when there are multiple processes on a node, you cannot monitor their performance because the performance registers are not saved and restored for each process.

### 6.2.2 Launching an MPMD Application

The `yod` utility supports multiple-program, multiple-data (MPMD) applications of up to 32 separate executable images. To run an MPMD application under `yod`, first create a `loadfile` where each line in the file is the `yod` command for one executable image. To communicate with each other, all of the executable images launched in a `loadfile` share the same `MPI_COMM_WORLD` process communicator.

The following `yod` options are valid within a `loadfile`:

- `-heap size` Specifies the number of bytes to reserve for the heap.
- `-list processor-list` Lists the specific compute nodes on which to run the application, such as `-list 42,58,64..100,150..200`. This option should be used only for testing and diagnostic purposes.
- `-Priority priority` Sets the process priority.
- `-size|-sz|-np n` Specifies the number of processors (compute nodes) on which to run the application.
- `-stack size` Specifies the number of bytes to reserve for the stack.
Example 6: Using a Loadfile

This simple loadfile script launches program1 on 128 nodes and program2 on 256 nodes:

```
#loadfile
yod -sz 128 program1
yod -sz 256 program2
```

To launch the application, enter:

```
% yod -F loadfile
```

6.2.3 Managing Compute Node Processors from an MPI Program

Programs that use MPI library routines for parallel control and communication should call the `MPI_Finalize()` routine at the conclusion of the program. This call waits for all processing elements to complete before exiting. However, if one of the processes fails to start or stop for any reason, the program will never complete and `yod` will hang. To prevent this behavior, use the `-tlimit` argument to `yod`, to terminate the application after a specified number of seconds. For example,

```
% yod -tlimit 30K myprog1
```

will terminate all processes remaining after 30K seconds so that `MPI_Finalize()` can complete. You can also use the environment variable `YOD_TIME_LIMIT` to specify the time limit. The time limit specified on the command line will override the value specified by the environment variable. The PBS Pro time limit also will terminate remaining processes that have not executed `MPI_Finalize()`.

6.2.4 Input and Output Modes under `yod`

All standard I/O requests are funneled through `yod`. The `yod` utility handles standard input (`stdin`) on behalf of the user and handles standard output (`stdout`) and standard error messages (`stderr`) for user applications.

See Section 4.3, page 24 for other I/O considerations.

6.2.5 Signal Handling under `yod`

The `yod` utility uses two signal handlers, one for the load sequence and a second for application execution. During the load operation, all signals sent to `yod`
Running an Application [6]

will terminate the operation. Once the load is completed and all nodes of the application have signed in with yod, the second signal handler takes over.

During the execution of a program, yod takes most signals as being intended for itself rather than the application. The only signals propagated to the application are SIGUSR1, SIGUSR2, and SIGTERM. All other signals will effectively terminate the running application. Note that the application can ignore the signals that yod passes along to it; SIGTERM, for example, will not necessarily terminate an application. However, a SIGINT delivered to yod will initiate a forced termination of the application.

6.2.6 Associating a Project or Task with a Job Launch

Use the –Account "project task" or –A "project task" yod option or the –A "project task" qsub option to associate a job launch with a particular project and task. Use double quotes around the string that specifies the project and, optionally, task values. For example:

% yod –Account "grid_test_1234 task1" -np 16 myapp123

You can also use the environment variable XT_ACCOUNT="project task" to specify account information. The –Account or –A command line option overrides the environment variable.

If yod is invoked from a batch job, the –A qsub account information takes precedence; yod writes a warning message to stderr in this case.

6.3 Using PBS Pro

Your Cray XT3 programming environment may include the optional PBS Pro batch scheduling software package from Altair Grid Technologies. This section provides an overview of job launching under PBS Pro.

For a list of PBS Pro documentation, see Section 1.2, page 2.

6.3.1 Submitting a PBS Pro Batch Job

To submit a job to the batch scheduler, use the following command:

% qsub [-l size=n] jobscript

where n is the number of processors to allocate to the job, and jobscript is the name of a job script that includes a yod command to launch the job. When the size=n option is not specified, qsub defaults to scheduling a single processor.
If you are running multiple sequential jobs, the number of processors you specify as an argument to `qsub` is the largest number of processors required by an invocation of `yod` in your script. For example, if your job script `job123` includes these calls to `yod`:

```
yod -sz 4 a.out
yod -sz 8 b.out
yod -sz 16 c.out
```

you would specify `size=16` in the `qsub` command line:

```
% qsub -l size=16 job123
```

However, if you are running multiple parallel jobs, the number of processors is the total number of processors specified by calls to `yod`. For example, if your job script includes these calls to `yod`:

```
yod -sz 4 a.out &
yod -sz 8 b.out &
yod -sz 16 c.out &
```

you would specify `size=28` in the `qsub` command line.

In either case, `yod` commands invoked from a script will use only those processors that were allocated to the batch job. See the `qsub(1B)` man page for details.

### 6.3.2 Using a Job Script

A job script may consist of PBS Pro directives, comments, and executable statements. A PBS Pro directive provides a way to specify job attributes apart from the command line options:

```
#PBS -N job_name
#PBS -l size=num_processors
#
command
command
...
```
The `qsub` command scans the lines of the script file for directives. An initial line in the script that begins with the characters `#!` or the character `:` will be ignored and scanning will start with the next line. Scanning will continue until the first executable line (that is, a line that is not blank, not a directive line, nor a line whose first non-white-space character is `#`). If directives occur on subsequent lines, they will be ignored.

If a `qsub` option is present in both a directive and on the command line, the command line takes precedence. If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

**Example 7: A Simple Job Script**

This example of a simple job script requests 16 processors to run the application `myprog`:

```bash
#!/bin/bash
#
# Define the destination of this job
# as the queue named "workq":
#PBS -q workq
#PBS -l size=16
# Tell PBS Pro to keep both standard output and
# standard error on the execution host:
#PBS -k eo
yod -sz 16 myprog
exit 0
```

### 6.3.3 Getting Jobs Status

The `qstat` command displays the following information about all jobs currently running under PBS Pro:

- The job identifier (Job id) assigned by PBS Pro
- The job name (Name) given by the submitter
- The job owner (User)
- CPU time used (Time Use)
- The job state (S): whether job is exiting (E), held (H), in the queue (Q), running (R), suspended (S, being moved to a new location (T), or waiting for its execution time (W)
• The queue (Queue) in which the job resides

For example:

```
% qstat
```

```
Job id   Name   User   Time    Use  S  Queue
------   ------- ------ ---------- -------- - -----
2983.la3db1 STDIN  alw  47:33:12 H workq
```

If the `-a` option is used, queue information is displayed in the alternative format.

```
% qstat -a
```

```
Job ID   Username  Queue  Jobname  SessID  Queue  Nodes  Time    S  Time
------   --------   -----   ------   ------   ------   -----    ------ ----
2983     cat       workq  STDIN   15951    536:53  10      R 47:25
```

Total compute nodes allocated: 10

See the `qstat(1B)` man page for details.

### 6.3.4 Removing a Job from the Queue

The `qdel` command removes a PBS Pro batch job from the queue. As a user, you can remove any batch job for which you are the owner. Jobs are removed from the queue in the order they are presented to `qdel`. See the `qdel(1B)` man page and the `PBS Pro 5.3 User Guide, PBS-3BU01` for more information.

### 6.3.5 Cray XT3 Specific PBS Pro Functions

The `pbs_resources_xt3(7B)` man page describes the resources that PBS Pro supports on Cray XT3 systems. You specify these resources by including them in the `-l` option argument on the `qsub` or `qalter` command or in a PBS Pro job script. See the description of the `-l` option in the `qsub(1B)` man page for more.

### 6.4 Running Applications in Parallel

Single-CPU programs as well as MPI and SHMEM programs can be run in parallel under `yod`. Although the following programming examples given are for MPI programs, most of this information is applicable to single-CPU and SHMEM programs as well.
**Example 8: Running an MPI Program Interactively**

This example shows how to create, compile, and run a simple MPI program.

Create a C program, `simple.c`:

```c
#include "mpi.h"

int main(int argc, char *argv[]) 
{
    int rank;
    int numprocs;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    printf("hello from pe %d of %d\n",rank,numprocs);
    MPI_Finalize();
}
```

Compile the program:

```bash
% cc -o simple simple.c
```

Run the program in interactive mode on 6 processors.

```bash
% yod -sz 6 simple
```

The output to stdout will look like this:

```
hello from pe 3 of 6
hello from pe 5 of 6
hello from pe 2 of 6
hello from pe 0 of 6
hello from pe 4 of 6
hello from pe 1 of 6
```

**Example 9: Running an MPI Program under PBS Pro**

This example shows a simple batch script that runs the program `simple.c` from the previous example.

Create a simple batch script, `my_jobscript`:

```bash
% cat my_jobscript

#PBS -N s_job       # Optional - specify name of job
```

**Example 9: Running an MPI Program under PBS Pro**

This example shows a simple batch script that runs the program `simple.c` from the previous example.

Create a simple batch script, `my_jobscript`:

```bash
% cat my_jobscript

#PBS -N s_job       # Optional - specify name of job
```
Submit the script to the PBS Pro batch system:

```bash
% qsub my_jobscript
```

The `qsub` command produces a batch job log file, `s_job.0nnnnn`. To view the output enter:

```bash
% cat s_job.0nnnnn
```

Ignore this warning message, if present:

```
Warning: no access to tty (Bad file descriptor).
Thus no job control in this shell.
```

The output will be similar to this:

```
hello from pe 3 of 6
hello from pe 5 of 6
hello from pe 2 of 6
hello from pe 0 of 6
hello from pe 4 of 6
hello from pe 1 of 6
```

**Example 10: Using a Script to Create and Run a Batch Job**

This example script takes two arguments—the name of a program and the number of processors on which to run the program. The script, called `run123`, performs the following actions:

1. Creates a temporary file that contains a PBS Pro batch job script
2. Submits the file to PBS Pro
3. Deletes the temporary file

Create script `run123`:

```bash
% cat run123
```

```bash
#!/bin/csh
if ( "$1" == "" ) then
    echo "Usage: run [executable|script] [ncpus]"
```
exit
endif
set n=1 # set default number of CPUs
if ( "$2" != "" ) set n=$2
cat > job.$$ <<EOT #creates the batch jobs script
#!/bin/csh
#PBS -N $1
#PBS -l size=$n
#PBS -joe
module load PrgEnv
cd \$PBS_O_WORKDIR
yod -sz $n -tlimit 30 $1
EOT
qsub job.$$ # submit batch job
rm job.$$

Run the job script:

% run123 shmem2 4

List the job output:

% cat shmem2.o52065

<table>
<thead>
<tr>
<th>PE</th>
<th>COMPUTEDSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.000000</td>
</tr>
<tr>
<td>3</td>
<td>6.000000</td>
</tr>
<tr>
<td>0</td>
<td>6.000000</td>
</tr>
<tr>
<td>2</td>
<td>6.000000</td>
</tr>
</tbody>
</table>
This chapter describes some of the debugging options that are native to the Cray XT3 programming environment, as well as the optional TotalView debugging software package from Etnus.

### 7.1 Troubleshooting Application Failures

The `yod` utility can provide rudimentary diagnostics for a subset of compute node operating system calls. The subset consists of the following system calls which perform remote procedure calls (RPCs) to `yod`:

<table>
<thead>
<tr>
<th>RPCs to <code>yod</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>chmod</td>
</tr>
<tr>
<td>fstatfs</td>
</tr>
<tr>
<td>mkdir</td>
</tr>
<tr>
<td>rmdir</td>
</tr>
<tr>
<td>symlink</td>
</tr>
<tr>
<td>chown</td>
</tr>
<tr>
<td>fsync</td>
</tr>
<tr>
<td>open</td>
</tr>
<tr>
<td>setegid</td>
</tr>
<tr>
<td>sync</td>
</tr>
<tr>
<td>close</td>
</tr>
<tr>
<td>ftruncate</td>
</tr>
<tr>
<td>pread</td>
</tr>
<tr>
<td>seteuid</td>
</tr>
<tr>
<td>truncate</td>
</tr>
<tr>
<td>exit</td>
</tr>
<tr>
<td>getdirent</td>
</tr>
<tr>
<td>pwrite</td>
</tr>
<tr>
<td>setgid</td>
</tr>
<tr>
<td>umask</td>
</tr>
<tr>
<td>fchmod</td>
</tr>
<tr>
<td>link</td>
</tr>
<tr>
<td>read</td>
</tr>
<tr>
<td>setuid</td>
</tr>
<tr>
<td>unlink</td>
</tr>
<tr>
<td>fchown</td>
</tr>
<tr>
<td>lseek</td>
</tr>
<tr>
<td>readlink</td>
</tr>
<tr>
<td>stat</td>
</tr>
<tr>
<td>utimes</td>
</tr>
<tr>
<td>fstat</td>
</tr>
<tr>
<td>lstat</td>
</tr>
<tr>
<td>rename</td>
</tr>
<tr>
<td>statfs</td>
</tr>
<tr>
<td>write</td>
</tr>
</tbody>
</table>

Any system calls that are performed solely by Catamount will not show up in the diagnostic output.

There are two ways to enable this feature:

- **Invoke `yod` with the `-strace` option.**
- **Set `YOD_STRACE=1` in your shell environment.**

Note that in this context the term `strace` is a misnomer. The `yod` utility does not provide the UNIX like `strace()` function. Enabling `strace` simply turns on diagnostic output generated by the RPC library, which `yod` uses to service the above-listed system calls. Also note that the I/O-related system calls are for the non-parallel file system.
7.2 The TotalView Debugger

Cray XT3 supports a special implementation of the Etnus TotalView debugger. TotalView provides source-level debugging of MPI applications and is compatible with the PGI Fortran 90, C, and C++ compilers.

7.2.1 Overview of TotalView Features

- Provides both a command line interface (with command line help) and a Motif-based graphical user interface
- Supports C, C++, Fortran 90, and the x86-64 Assembler
- Supports programs written in mixed languages
- Supports PGI compilers for compute node executables
- Supports debugging of up to 1024 compute node processes
- Supports MPI message queue display
- Supports watchpoints

7.2.2 Differences in Functionality for Cray XT3

The TotalView debugging suite for the Cray XT3 system differs in functionality from the standard TotalView implementation in the following ways:

- The TotalView Visualizer is not included
- The TotalView HyperHelp browser is not included
- Debugging multiple threads on compute nodes is not supported
- Debugging MPI_Spawn, OpenMP, Cray SHMEM, or PVM programs is not supported
- Compiled \texttt{EVAL} points and expressions are not supported
- Type transformations for the PGI C++ compiler standard template library collection classes are not supported
- Exception handling for the PGI C++ compiler runtime library is not supported
- Spawning a process onto the compute processors is not supported
• Machine partitioning schemes, gang scheduling or batch systems are not supported

In some cases, the functionality is limited because Catamount does not support the feature in the user program.

7.2.3 Obtaining the TotalView Debugger

The TotalView debugging suite is not included with the Cray XT3 software package. For information on purchasing TotalView for Cray XT3, contact Etnus directly through http://www.etnus.com.

7.2.4 Using The TotalView Debugger

As part of launching an application on a compute node, TotalView will launch a server program on your login node using ssh. As with any ssh session, authentication will be required. It is recommended that users enable ssh without a passphrase, as explained in Section 2.1.2, page 6.

TotalView typically is run interactively. If your site has not designated any compute nodes for interactive processing, use the PBS Pro qsub -I interactive mode described in Section 6.1, page 33.

If the TotalView debugging suite is installed on your system, the following command loads the debugger into your user environment:

```
% module load totalview
```

Use the following command to start the TotalView command line interface on an application:

```
% tv6cli yod [-a argument_list] application_name
```

Use the following command to start the TotalView graphical user interface on an application:

```
% tv6 yod [-a argument_list] application_name
```

**Example 11: Using TotalView**

This example shows how to invoke TotalView to debug application a.out.

```
% qsub -I -l size=4
```

qsub: waiting for job 14448.nid00003 to start
qsub: job 14448.nid00003 ready
DISPLAY is user1:0.0
Linux perch 2.4.21-0-sles9-ss-lustre #2 Fri Apr 29
17:14:15 PDT 2005 x86_64 x86_64 x86_64 GNU/Linux
/ufs/home/users/user1

% module load totalview

% cd working_directory

% totalview yod -a -sz 4 a.out
This chapter describes the Cray XT3 performance analysis tools.

8.1 Performance API (PAPI)

The Performance API (PAPI) is a standard API for accessing the registers on the microprocessor board that count events or occurrences of specific signals related to the processor’s function. By monitoring these events, you can determine the extent to which your code efficiently maps to the underlying architecture.

PAPI provides two interfaces to the counter hardware:

- A simple high-level interface for basic measurements
- A fully programmable, low-level interface for users with more sophisticated needs

To use PAPI, you need to load the PAPI module:

```bash
% module load papi
```

8.1.1 Using the High-level Interface

The high-level interface provides the ability to start, stop and read specific events, one at a time.

**Example 12: The High-level PAPI Interface**

Create sample program `example1.c`:

```c
#include <papi.h>
void main()
{
  int retval, Events[2]= {PAPI_TOT_CYC, PAPI_TOT_INS};
  long_long values[2];

  if (PAPI_start_counters (Events, 2) != PAPI_OK) {
    printf("Error starting counters\n");
    exit(1);
  }

  /* Do some computation here... */
```
if (PAPI_stop_counters (values, 2) != PAPI_OK) {
    printf("Error stopping counters\n");
    exit(1);
}

printf("PAPI_TOT_CYC = %lld\n", values[0]);
printf("PAPI_TOT_INS = %lld\n", values[1]);
}

To compile example1.c, enter:
% module load papi
% cc -c example1.c
% cc -o example1 example1.o

To run the program, enter:
% yod example1

Output from this example:
PAPI_TOT_CYC = 2314
PAPI_TOT_INS = 256

8.1.2 Using the Low-level Interface

The low-level PAPI interface deals with hardware events in groups called event sets. An event set is based on mapping the hardware counters available on the system to a set of predefined events, called presets. The event set reflects how the counters are most frequently used, such as taking simultaneous measurements of different hardware events and relating them to one another. For example, relating cycles to memory references or flops to level 1 cache misses can reveal poor locality and memory management. Event sets are fully programmable and have features such as guaranteed thread safety, writing of counter values, multiplexing, and notification on threshold crossing as well as processor-specific features. For the list of predefined event sets, see the hwpc(3) man page. For information on constructing an event set, see the PAPI User Guide and the PAPI Programmer’s Reference.

See Appendix C, page 79 for a list of supported hardware counter presets from which to construct an event set.
Example 13: The Low-level Interface

This example creates an event set and counts events as they occur:

```c
#include <papi.h>
void main()
{
    int EventSet = PAPI_NULL;
    long_long values[1];

    /* Initialize PAPI library */
    if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT) {
        printf("Error initializing PAPI library\n");
        exit(1);
    }

    /* Create Event Set */
    if (PAPI_create_eventset(&EventSet) != PAPI_OK) {
        printf("Error creating eventset\n");
        exit(1);
    }

    /* Add Total Instructions Executed to eventset */
    if (PAPI_add_event (EventSet, PAPI_TOT_INS) != PAPI_OK) {
        printf("Error adding event\n");
        exit(1);
    }

    /* Start counting ... */
    if (PAPI_start (EventSet) != PAPI_OK) {
        printf("Error starting counts\n");
        exit(1);
    }

    /* Do some computation here...*/

    if (PAPI_read (EventSet, values) != PAPI_OK) {
        printf("Error stopping counts\n");
        exit(1);
    }

    printf("PAPI_TOT_INS = %lld\n", values[0]);
}
```
To compile and run the program, enter:

% module load papi
% cc -c example2.c
% cc -o example2 example2.o
% yod example2

Output from this example:
PAPI_TOT_INS = 208

8.2 CrayPat Performance Analysis Tool

The Cray Performance Analysis Tool (CrayPat) helps you analyze the performance of programs running on Cray XT3 systems. Here is an overview of how to use it:

1. Load the craypat module:

   % module load craypat

   Note: You need to load the craypat module before building even the uninstrumented version of the application or you will get an error from pat_build.

2. Compile and link your application.

3. Use pat_build to create an instrumented version of the application, specifying the functions to be traced via options such as -u and -g mpi.

4. Set any relevant environment variables, such as:

   - PAT_RT_HWPC=1, which specifies the first of the 9 predefined sets of hardware counter events.

   - PAT_RT_SUMMARY=0, which specifies a full trace data file rather than a profile version. Such a file can be very large but is needed to view behavior over time with Cray Apprentice².

5. Execute the instrumented program.
6. Use `pat_report` on the resulting data file to generate a report. The default report is a profile by function, but alternative views can be specified via options such as:

- `-b calltree,pe=HIDE` (omit `=HIDE` to see per-pe data)
- `-b functions,callers,pe=HIDE`
- `-b functions,pe` (shows per-pe data)

These steps are illustrated in the following examples. For more details, see the man pages and the interactive `pat_help` utility.

CrayPat on Cray XT3 systems supports one type of experiment: tracing. Tracing counts an event such as the number of times an MPI call is executed. Profiling and sampling experiments are not supported. Therefore, setting the runtime environment variable `PAT_RT_EXPERIMENT` to any value other than `trace` will result in a runtime error from the CrayPat runtime library.

CrayPat provides profile information by collecting and reporting trace-based information about total user time and system time consumed by a program and its functions. For an example of profile information, see the summary table at the end of `program1.rpt1` in Example 14, page 53.

**Example 14: CrayPat Basics**

This example shows how to instrument a program, run the instrumented program, and generate CrayPat reports.

Load the `craypat` module:

```
% module load craypat
```

Then compile the sample program `prog.f90` and the routine it calls, `work.c`

Source code of `prog.f90`:

```fortran
program main
include 'mpif.h' ! Required

call MPI_Init(ierr) ! Required
call MPI_Comm_rank(MPI_COMM_WORLD,mype,ierr)
call MPI_Comm_size(MPI_COMM_WORLD,npes,ierr)

print *,’hello from pe’,mype,’ of’,npes

do i=1+mype,1000,npes ! Distribute the work
```

```
call work(i,mype)
enddo

call MPI_Finalize(ierr) ! Required
end

Source code of routine work.c:
void work_(int *N, int *MYPE)
{
    int n=*N, mype=*MYPE;

    if (n == 42) {
        printf("PE %d: sizeof(long) = %d\n",mype,sizeof(long));
        printf("PE %d: The answer is: %d\n",mype,n);
    }
}

Compile prog.f90 and work.c:
% ftn -c prog.f90
% cc -c work.c

Create executable program1:
% ftn -o program1 prog.o work.o

Run pat_build to generate instrumented program program1+pat:
% pat_build -u -g mpi program1 program1+pat

Note that the tracegroup (-g option) is mpi.

Run instrumented program program1+pat:
% qsub -I -l size=4
% yod -sz 4 program1+pat

CrayPat/X: Version 10.224  06/13/05 15:06:17
CrayPat/X:  Runtime summarization enabled. Set PAT_RT_SUMMARY=0 to disable.
hello from pe 0  of  4
hello from pe 1  of  4
hello from pe 2  of  4
hello from pe 3  of  4
PE 1: sizeof(long) = 8
PE 1: The answer is: 42

Find the pat_build experiment data file *.xf:

% ls *.xf

program1+pat+74td.xf

Load the craypat module:

% module load craypat

Run pat_report to generate reports program1.rpt1 (using default pat_report options) and program1.rpt2 (using the -b calltree option).

% pat_report program1+pat+74td.xf > program1.rpt1
% pat_report -b calltree,pe=HIDE program1+pat+74td.xf > program1.rpt2

List program1.rpt1:

% more program1.rpt1

CrayPat/X: Version 10.224 (xf 10.5) 06/13/05 15:06:17

Experiment: trace

Experiment data file:
/ufs/home/users/userw/userx/program1+pat+74td.xf (RTS)

Original program: /ufs/home/users/userw/program1

Instrumented program: /ufs/home/users/userw/userx/./program1+pat

Program invocation: ./program1+pat

Runtime environment variables:
PAT_ROOT=/opt/xt-tools/craypat/1.0.2/cpatx

Report time environment variables:
PAT_ROOT=/opt/xt-tools/craypat/1.0.2/cpatx

Report command line options: <none>
Host name and type:  bass x86_64 2000 MHz

Operating system:  catamount 1.0 2.0

Traced functions:
- MPI_Allreduce  .../src/mpi/coll/allreduce.c
- MPI_Barrier  .../src/mpi/coll/barrier.c
- MPI_Bcast  .../src/mpi/coll/bcast.c
- MPI_Comm_rank  .../src/mpi/comm/comm_rank.c
- MPI_Comm_size  .../src/mpi/comm/comm_size.c
- MPI_Finalize  .../src/mpi/init/finalize.c
- MPI_Get_count  .../src/mpi/datatype/get_count.c
- MPI_Init  .../src/mpi/init/init.c
- MPI_Init_thread  .../src/mpi/init/initthread.c
- MPI_Op_create  .../src/mpi/coll/op_create.c
- MPI_Pack  .../src/mpi/datatype/pack.c
- MPI_Pack_size  .../src/mpi/datatype/pack_size.c
- MPI_Reduce  .../src/mpi/coll/reduce.c
- MPI_Type_get_extent  .../src/mpi/datatype/type_get_extent.c
- MPI_Type_get_true_extent  .../src/mpi/datatype/type_get_true_extent.c
- MPI_Type_size  .../src/mpi/datatype/type_size.c
- MPI_Unpack  .../src/mpi/datatype/unpack.c
- _Exit  .../..sysdeps/catamount/_exit.c
- longjmp  .../..sysdeps/generic/longjmp.c
- main  ==NA==
- mpi_comm_rank_  .../src/binding/f77/comm_rankf.c
- mpi_comm_size_  .../src/binding/f77/comm_sizef.c
- mpi_wtick_  .../src/binding/f77/wtickf.c
- mpi_wtime_  .../src/binding/f77/wtimef.c
- work_  .../home/users/userx/work.c

Table 1:  -d time%,cum_time%,time,traces,P
         -b exp,function,pe=HIDE

<table>
<thead>
<tr>
<th>Time%</th>
<th>Cum.Time%</th>
<th>Time</th>
<th>Calls</th>
<th>Experiment=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>100.0%</td>
<td>0.018196</td>
<td>1032</td>
<td>Total</td>
</tr>
<tr>
<td>98.6%</td>
<td>98.6%</td>
<td>0.017939</td>
<td>4</td>
<td>main</td>
</tr>
<tr>
<td>1.4%</td>
<td>99.9%</td>
<td>0.000248</td>
<td>1000</td>
<td>work_</td>
</tr>
</tbody>
</table>
Exit status and elapsed time by process:

<table>
<thead>
<tr>
<th>PE</th>
<th>Status</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.040101</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.020731</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.020067</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.020026</td>
</tr>
</tbody>
</table>

List program1.rpt2 (snippet):

```bash
% more program1.rpt2

... Report command line options: -b calltree,pe=HIDE

... Table 1: -d time%,cum_time%,time,traces,P
      -b calltree,pe=HIDE

<table>
<thead>
<tr>
<th>Time%</th>
<th>Cum.Time%</th>
<th>Time</th>
<th>Calls</th>
<th>Calltree</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>100.0%</td>
<td>0.018196</td>
<td>1032</td>
<td>Total</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>100.0%</td>
<td>100.0%</td>
<td>0.018195</td>
<td>1028</td>
<td>main</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>98.6%</td>
<td>98.6%</td>
<td>0.017939</td>
<td>4</td>
<td>main(exclusive)</td>
</tr>
<tr>
<td>1.4%</td>
<td>100.0%</td>
<td>0.000254</td>
<td>1016</td>
<td>MAIN_</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-----------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>1.4%</td>
<td>99.9%</td>
<td>0.000248</td>
<td>1000</td>
<td>work_</td>
</tr>
<tr>
<td>0.0%</td>
<td>100.0%</td>
<td>0.000003</td>
<td>8</td>
<td>mpi_comm_rank_</td>
</tr>
</tbody>
</table>
Example 15: Using Hardware Performance Counters

This example uses the same instrumented program as Example 14, page 53 and generates reports showing hardware performance counter information.

Collect hardware performance counter event set 1 information and generate report program1.rpt3 (see the hwpc(3) man page for a list of predefined event sets):

```bash
% setenv PAT_RT_HWPC 1
% yod -sz 4 program1+pat
```

Exit status and elapsed time by process:

<table>
<thead>
<tr>
<th>PE</th>
<th>Status</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.040101</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.020731</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.020067</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.020026</td>
</tr>
</tbody>
</table>

Example 15: Using Hardware Performance Counters

This example uses the same instrumented program as Example 14, page 53 and generates reports showing hardware performance counter information.

Collect hardware performance counter event set 1 information and generate report program1.rpt3 (see the hwpc(3) man page for a list of predefined event sets):

```bash
% setenv PAT_RT_HWPC 1
% yod -sz 4 program1+pat
```

Exit status and elapsed time by process:

<table>
<thead>
<tr>
<th>PE</th>
<th>Status</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.040101</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.020731</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.020067</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.020026</td>
</tr>
</tbody>
</table>
hello from pe 2 of 4
hello from pe 3 of 4
PE 1: sizeof(long) = 8
PE 1: The answer is: 42

% pat_report program1+program1+pat+101td.xf > program1.rpt3

List program1.rpt3 (snippet):

% more program1.rpt3

CrayPat/X:  Version 10.224 (xf 10.5) 06/13/05 15:06:17

Experiment: trace

Experiment data file:
   /ufs/home/users/userw/userx/program1+pat+101td.xf  (RTS)

Original program: /ufs/home/users/userx/program1

Instrumented program: /ufs/home/users/userw/userx/program1+pat

Program invocation: program1+pat

Runtime environment variables:
   PAT_ROOT=/opt/xt-tools/craypat/1.0.2/cpatx
   PAT_RT_HWPC=1

Report time environment variables:
   PAT_ROOT=/opt/xt-tools/craypat/1.0.2/cpatx

Report command line options: <none>

Host name and type: bass x86_64 2000 MHz

Operating system: catamount 1.0 2.0

Hardware performance counter events:
   PAPI_TLB_DM  Data translation lookaside buffer misses
   PAPI_L1_DCA  Level 1 data cache accesses
   PAPI_FP_OPS  Floating point operations
   DC_MISS     Data Cache Miss
   User_Cycles Virtual Cycles
Traced functions:

- MPI_Allreduce  
  .../src/mpi/coll/allreduce.c
- MPI_Barrier  
  .../src/mpi/coll/barrier.c
- MPI_Bcast  
  .../src/mpi/coll/bcast.c
- MPI_Comm_rank  
  .../src/mpi/comm/comm_rank.c
- MPI_Comm_size  
  .../src/mpi/comm/comm_size.c
- MPI_Finalize  
  .../src/mpi/init/finalize.c
- MPI_Get_count  
  .../src/mpi/datatype/get_count.c
- MPI_Init  
  .../src/mpi/init/init.c
- MPI_Init_thread  
  .../src/mpi/init/initthread.c
- MPI_Op_create  
  .../src/mpi/coll/op_create.c
- MPI_Pack  
  .../src/mpi/datatype/pack.c
- MPI_Pack_size  
  .../src/mpi/datatype/pack_size.c
- MPI_Reduce  
  .../src/mpi/coll/reduce.c
- MPI_Type_get_extent  
  .../src/mpi/datatype/type_get_extent.c
- MPI_Type_get_true_extent  
  .../src/mpi/datatype/type_get_true_extent.c
- MPI_Type_size  
  .../src/mpi/datatype/type_size.c
- MPI_Unpack  
  .../src/mpi/datatype/unpack.c
- _Exit  
  .../..sysdeps/catamount/_exit.c
- longjmp  
  .../..sysdeps/generic/longjmp.c
- main  
  ==NA==
- mpi_comm_rank_  
  .../src/binding/f77/comm_rankf.c
- mpi_comm_size_  
  .../src/binding/f77/comm_sizef.c
- mpi_wtick_  
  .../src/binding/f77/wtickf.c
- mpi_wtime_  
  .../src/binding/f77/wtimef.c
- work_  
  .../home/users/userx/work.c

Table 1: -d time%,cum_time%,time,traces,P

-b exp,function,pe=HIDE

| Experiment=1 | Function | PE=‘HIDE’ |

Totals for program

<table>
<thead>
<tr>
<th>Time%</th>
<th>100.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cum.Time%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Time</td>
<td>0.018211</td>
</tr>
<tr>
<td>Calls</td>
<td>1032</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.025M/sec</td>
</tr>
<tr>
<td>Metric</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>5735.205M/sec</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>0.023M/sec</td>
</tr>
<tr>
<td>DC_MISS</td>
<td>0.106M/sec</td>
</tr>
<tr>
<td>User time</td>
<td>0.018 secs</td>
</tr>
<tr>
<td>Utilization rate</td>
<td>100.0%</td>
</tr>
<tr>
<td>HW FP Ops / Cycles</td>
<td>0.00 ops/sec</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>0.023M/sec</td>
</tr>
<tr>
<td>Computation intensity</td>
<td>0.00 ops/ref</td>
</tr>
<tr>
<td>LD &amp; ST per TLB miss</td>
<td>230040.84 ops/miss</td>
</tr>
<tr>
<td>LD &amp; ST per D1 miss</td>
<td>54141.29 ops/miss</td>
</tr>
<tr>
<td>D1 cache hit ratio</td>
<td>100.0%</td>
</tr>
<tr>
<td>% TLB misses / cycle</td>
<td>0.0%</td>
</tr>
<tr>
<td><strong>main</strong></td>
<td></td>
</tr>
<tr>
<td>Time%</td>
<td>98.5%</td>
</tr>
<tr>
<td>Cum.Time%</td>
<td>98.5%</td>
</tr>
<tr>
<td>Time</td>
<td>0.017933</td>
</tr>
<tr>
<td>Calls</td>
<td>4</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.017M/sec</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>5736.235M/sec</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>0.024M/sec</td>
</tr>
<tr>
<td>DC_MISS</td>
<td>0.087M/sec</td>
</tr>
<tr>
<td>User time</td>
<td>0.018 secs</td>
</tr>
<tr>
<td>Utilization rate</td>
<td>100.0%</td>
</tr>
<tr>
<td>HW FP Ops / Cycles</td>
<td>0.00 ops/sec</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>0.024M/sec</td>
</tr>
<tr>
<td>HW FP Ops / WCT</td>
<td>0.024M/sec</td>
</tr>
<tr>
<td>Computation intensity</td>
<td>0.00 ops/ref</td>
</tr>
<tr>
<td>LD &amp; ST per TLB miss</td>
<td>330775.83 ops/miss</td>
</tr>
<tr>
<td>LD &amp; ST per D1 miss</td>
<td>65648.55 ops/miss</td>
</tr>
<tr>
<td>D1 cache hit ratio</td>
<td>100.0%</td>
</tr>
<tr>
<td>% TLB misses / cycle</td>
<td>0.0%</td>
</tr>
<tr>
<td><strong>work</strong></td>
<td></td>
</tr>
<tr>
<td>Time%</td>
<td>1.5%</td>
</tr>
<tr>
<td>Cum.Time%</td>
<td>99.9%</td>
</tr>
<tr>
<td>Time</td>
<td>0.000266</td>
</tr>
<tr>
<td>Calls</td>
<td>1000</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.539M/sec</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>5721.798M/sec</td>
</tr>
</tbody>
</table>
PAPI_FP_OPS 0 ops
DC_MISS 0.554M/sec 147 ops
User time 0.000 secs 530262 cycles
Utilization rate 99.8%
HW FP Ops / Cycles 0.00 ops/sec
HW FP Ops / User time 0 ops
HW FP Ops / WCT
Computation intensity 0.00 ops/ref
LD & ST per TLB miss 10608.57 ops/miss
LD & ST per D1 miss 10319.90 ops/miss
D1 cache hit ratio 100.0%
% TLB misses / cycle 0.0%

Exit status and elapsed time by process:

<table>
<thead>
<tr>
<th>Exit</th>
<th>PE Status</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.040201</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.020671</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.020150</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.020108</td>
</tr>
</tbody>
</table>

Collect information about data translation lookaside buffer misses (PAPI_TLB_DM) and generate report program1.rpt4:

```bash
% setenv PAT_RT_HWPC PAPI_TLB_DM
% yod -sz 4 program1+pat
% pat_report program1+pat+103td.xf > program1.rpt4
```

List program1.rpt4 (snippet):

```bash
% more program1.rpt4
```

Experiment data file:
```
/ufs/home/users/userw/userx/program1+pat+103td.xf (RTS)
```
Runtime environment variables:
    PAT_ROOT=/opt/xt-tools/craypat/1.0.2/cpatx
    PAT_RT_HWPC=PAPI_TLB_DM

Report time environment variables:
    PAT_ROOT=/opt/xt-tools/craypat/1.0.2/cpatx

Hardware performance counter events:
    PAPI_TLB_DM  Data translation lookaside buffer misses
    User_Cycles  Virtual Cycles

========================================================================
Totals for program
========================================================================
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Cum.Time%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Time</td>
<td>0.018343</td>
</tr>
<tr>
<td>Calls</td>
<td>1032</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.025M/sec    463 misses</td>
</tr>
<tr>
<td>User time</td>
<td>0.018 secs 36684586 cycles</td>
</tr>
<tr>
<td>Utilization rate</td>
<td>100.0%</td>
</tr>
<tr>
<td>% TLB misses / cycle</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

========================================================================
main
========================================================================
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>98.2%</td>
</tr>
<tr>
<td>Cum.Time%</td>
<td>98.2%</td>
</tr>
<tr>
<td>Time</td>
<td>0.018016</td>
</tr>
<tr>
<td>Calls</td>
<td>4</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.018M/sec    321 misses</td>
</tr>
<tr>
<td>User time</td>
<td>0.018 secs 36032456 cycles</td>
</tr>
<tr>
<td>Utilization rate</td>
<td>100.0%</td>
</tr>
<tr>
<td>% TLB misses / cycle</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

========================================================================
work
========================================================================
For more information about using CrayPat, see the `craypat(1)` man page and run the `pat_help` utility. For more information about PAPI hardware performance counters, see Appendix C, page 79, the `hwpc(3)` man page, and the PAPI web site at [http://icl.cs.utk.edu/papi/](http://icl.cs.utk.edu/papi/).

### 8.3 Cray Apprentice

Cray Apprentice is a performance data visualization tool. After you have used `pat_build` to instrument a program for a performance analysis experiment, executed the instrumented program, then used `pat_report` to convert the resulting data file to XML format, you can use Cray Apprentice to explore the experiment data file and generate a variety of interactive graphical reports.

To use Cray Apprentice, you need to load the Cray Apprentice module:

```bash
% module load apprentice2
```

---

1 Cray Apprentice is an optional software package available from Cray Inc.
Example 16: Cray Apprentice\textsuperscript{2} Basics

This example shows how to use Cray Apprentice\textsuperscript{2} to create a graphical representation of a CrayPat report.

Using experiment file `program1+pat+4485td.xf` from Example 14, page 53, generate a report in XML format (note the inclusion of the `-f xml` and `-c records` options):

```
% module load apprentice2

% pat_report -f xml -c records program1+pat+4485td.xf \ > program1.xml
```

Run Cray Apprentice\textsuperscript{2}:

```
% app2 program1.xml
```

Cray Apprentice\textsuperscript{2} displays `pat_report` data in graphical form. This example shows the Call Graph display option:
For more information about using Cray Apprentice², see the Cray Apprentice² online help system and the app2(1) and pat_report(1) man pages.
9.1 Compiler Optimization

After you have compiled and debugged your code and analyzed its performance, you can use a number of techniques to optimize performance. For details on compiler optimization and optimization reporting options, see the PGI User's Guide.

Optimization can produce code that is more efficient and runs significantly faster than code that is not optimized. Optimization can be performed at the compilation unit level via compiler driver options or to selected portions of code via the use of directives or pragmas. Note that optimization may increase compilation time and may make debugging difficult. It is best to use performance analysis data to isolate the portions of code where optimization would provide the greatest benefits.

In the following example, a Fortran matrix-multiply subroutine is optimized. The compiler driver option generates an optimization report.
Example 17: Optimization Report

Source code:

```fortran
subroutine mxm(x,y,z,m,n)
  real*8 x(m,n), y(m,n), z(n,n)
  do k = 1,n
    do j = 1,n
      do i = 1,m
        x(i,j) = x(i,j) + y(i,k)*z(k,j)
      enddo
    enddo
  enddo
end
```

Compiler command:

```
% ftn -c -fast -Minfo=all matrix_multiply.f90
```

Optimization report:

Timing stats:
```
Total time 0 millisecs
```
mxm:
```
6, Loop unrolled 4 times
```
Timing stats:
```
schedule 17 millisecs 51%
unroll 16 millisecs 48%
Total time 33 millisecs
```
The Catamount port of glibc supports the functions listed in Table 4. For further information, see the man pages for the functions.

<table>
<thead>
<tr>
<th>a64l</th>
<th>abort</th>
<th>abs</th>
<th>access</th>
</tr>
</thead>
<tbody>
<tr>
<td>addmntent</td>
<td>alarm</td>
<td>alphasort</td>
<td>argz_add</td>
</tr>
<tr>
<td>argz_add_sep</td>
<td>argz_append</td>
<td>argz_count</td>
<td>argz_create</td>
</tr>
<tr>
<td>argz_create_sep</td>
<td>argz_delete</td>
<td>argz_extract</td>
<td>argz_insert</td>
</tr>
<tr>
<td>argz_next</td>
<td>argz_replace</td>
<td>argz_stringify</td>
<td>asctime</td>
</tr>
<tr>
<td>asctime_r</td>
<td>asprintf</td>
<td>atexit</td>
<td>atof</td>
</tr>
<tr>
<td>atoi</td>
<td>atol</td>
<td>atoll</td>
<td>basename</td>
</tr>
<tr>
<td>bcmp</td>
<td>bcopy</td>
<td>bind_textdomain_codeset</td>
<td>bindtextdomain</td>
</tr>
<tr>
<td>bsearch</td>
<td>btowc</td>
<td>bzero</td>
<td>calloc</td>
</tr>
<tr>
<td>catclose</td>
<td>catgets</td>
<td>catopen</td>
<td>cbc_crypt</td>
</tr>
<tr>
<td>chdir</td>
<td>chmod</td>
<td>chown</td>
<td>clearenv</td>
</tr>
<tr>
<td>clearerr</td>
<td>clearerr_unlocked</td>
<td>close</td>
<td>confstr</td>
</tr>
<tr>
<td>copysign</td>
<td>copysignf</td>
<td>copysignl</td>
<td>creat</td>
</tr>
<tr>
<td>ctime</td>
<td>ctime_r</td>
<td>daemon</td>
<td>daylight</td>
</tr>
<tr>
<td>dcgettext</td>
<td>dcngettext</td>
<td>des_setparity</td>
<td>dgettext</td>
</tr>
<tr>
<td>dtime</td>
<td>dirfd</td>
<td>div</td>
<td>dftime</td>
</tr>
<tr>
<td>dprintf</td>
<td>dprintf</td>
<td>dup</td>
<td>dup2</td>
</tr>
<tr>
<td>dysize</td>
<td>dysize</td>
<td>dup</td>
<td>ecsvt</td>
</tr>
<tr>
<td>endfset</td>
<td>endmntent</td>
<td>endttyent</td>
<td>endusershell</td>
</tr>
<tr>
<td>envz_add</td>
<td>envz_entry</td>
<td>envz_get</td>
<td>envz_merge</td>
</tr>
<tr>
<td>envz_remove</td>
<td>envz_strip</td>
<td>err</td>
<td>envz_merge</td>
</tr>
<tr>
<td>errx</td>
<td>exit</td>
<td>fchmod</td>
<td>fchown</td>
</tr>
<tr>
<td>fclose</td>
<td>fcloseall</td>
<td>fcntl</td>
<td>fcvt</td>
</tr>
<tr>
<td>Function</td>
<td>Function</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>fcvt_r</td>
<td>fdatasync</td>
<td>fdopen</td>
<td></td>
</tr>
<tr>
<td>feof_unlocked</td>
<td>ferror</td>
<td>feof_unlocked</td>
<td></td>
</tr>
<tr>
<td>fflush_unlocked</td>
<td>ffs</td>
<td>ffsll</td>
<td></td>
</tr>
<tr>
<td>fgets_unlocked</td>
<td>fgetgrent</td>
<td>fgetpos</td>
<td></td>
</tr>
<tr>
<td>fgets</td>
<td>fgetws_unlocked</td>
<td>fgetwc</td>
<td></td>
</tr>
<tr>
<td>finite</td>
<td>flockfile</td>
<td>fnmatch</td>
<td></td>
</tr>
<tr>
<td>fprintf</td>
<td>fputc</td>
<td>fopen</td>
<td></td>
</tr>
<tr>
<td>fputs_unlocked</td>
<td>fgets_unlocked</td>
<td>fgets_unlocked</td>
<td></td>
</tr>
<tr>
<td>fputwc</td>
<td>fdopen</td>
<td>fgets_unlocked</td>
<td></td>
</tr>
<tr>
<td>fread</td>
<td>frexp</td>
<td>fgetpos</td>
<td></td>
</tr>
<tr>
<td>freopen</td>
<td>ftello</td>
<td>fgetwc</td>
<td></td>
</tr>
<tr>
<td>ftell</td>
<td>ftello</td>
<td>fgetpwent</td>
<td></td>
</tr>
<tr>
<td>ftruncate</td>
<td>ftrylockfile</td>
<td>fgetpwent</td>
<td></td>
</tr>
<tr>
<td>fwprintf</td>
<td>fwrite</td>
<td>fgetwc_unlocked</td>
<td></td>
</tr>
<tr>
<td>get_current_dir_name</td>
<td>getc</td>
<td>fputc</td>
<td></td>
</tr>
<tr>
<td>getchar_unlocked</td>
<td>getcwd</td>
<td>fputc_unlocked</td>
<td></td>
</tr>
<tr>
<td>getdelim</td>
<td>getdelim</td>
<td>fputwc</td>
<td></td>
</tr>
<tr>
<td>getenv</td>
<td>geteuid</td>
<td>free</td>
<td></td>
</tr>
<tr>
<td>getfsspec</td>
<td>getgid</td>
<td>fputw</td>
<td></td>
</tr>
<tr>
<td>getlogin</td>
<td>getpwnum</td>
<td>fputw</td>
<td></td>
</tr>
<tr>
<td>getopt_long</td>
<td>getpw</td>
<td>fputw</td>
<td></td>
</tr>
<tr>
<td>getpass</td>
<td>getpw</td>
<td>ftok</td>
<td></td>
</tr>
<tr>
<td>gettimeofday</td>
<td>getpw</td>
<td>ftok</td>
<td></td>
</tr>
<tr>
<td>getusershell</td>
<td>getpw</td>
<td>ftime</td>
<td></td>
</tr>
<tr>
<td>gsignal</td>
<td>getpw</td>
<td>ftime</td>
<td></td>
</tr>
<tr>
<td>hcreate</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>hcreate_r</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>hdestroy</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>iconv_open</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>imaxabs</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>index</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>initstate</td>
<td>getpw</td>
<td>fwrite</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Function</td>
<td>Function</td>
<td>Function</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>insque</td>
<td>ioctl</td>
<td>isalnum</td>
<td>isalpha</td>
</tr>
<tr>
<td>isascii</td>
<td>isblank</td>
<td>iscntrl</td>
<td>isdigit</td>
</tr>
<tr>
<td>isgraph</td>
<td>isinf</td>
<td>islower</td>
<td>isnan</td>
</tr>
<tr>
<td>isprint</td>
<td>ispunct</td>
<td>isspace</td>
<td>isupper</td>
</tr>
<tr>
<td>iswalnum</td>
<td>iswalpha</td>
<td>iswblank</td>
<td>iswcntrl</td>
</tr>
<tr>
<td>iswctype</td>
<td>iswdigit</td>
<td>iswgraph</td>
<td>iswlower</td>
</tr>
<tr>
<td>iswprint</td>
<td>iswpunct</td>
<td>iswspace</td>
<td>iswupper</td>
</tr>
<tr>
<td>iswdxigit</td>
<td>isxdigit</td>
<td>jrand48</td>
<td>kill</td>
</tr>
<tr>
<td>l64a</td>
<td>labs</td>
<td>llabs</td>
<td>ldexp</td>
</tr>
<tr>
<td>lfind</td>
<td>link</td>
<td>lockf</td>
<td>localeconv</td>
</tr>
<tr>
<td>localtime</td>
<td>localtime_r</td>
<td>lseek</td>
<td>longjmp</td>
</tr>
<tr>
<td>lrand48</td>
<td>lsearch</td>
<td>mbrench</td>
<td>lstat</td>
</tr>
<tr>
<td>malloc</td>
<td>mblen</td>
<td>mbrlen</td>
<td>mbrtowc</td>
</tr>
<tr>
<td>mbsinit</td>
<td>mbsnrtowcs</td>
<td>mbsrtowcs</td>
<td>mbstowcs</td>
</tr>
<tr>
<td>mbtowc</td>
<td>memccpy</td>
<td>memchr</td>
<td>memcmp</td>
</tr>
<tr>
<td>memcpy</td>
<td>memfrob</td>
<td>memmem</td>
<td>memmove</td>
</tr>
<tr>
<td>memrchr</td>
<td>memset</td>
<td>mkdir</td>
<td>mktemp</td>
</tr>
<tr>
<td>mknod</td>
<td>mkstemp</td>
<td>mkt ime</td>
<td>modf</td>
</tr>
<tr>
<td>modff</td>
<td>modfl</td>
<td>mrand48</td>
<td>nanosleep</td>
</tr>
<tr>
<td>ngetgettext</td>
<td>nl_langinfo</td>
<td>nrand48</td>
<td>on_exit</td>
</tr>
<tr>
<td>open</td>
<td>passwd2des</td>
<td>pclose</td>
<td>perror</td>
</tr>
<tr>
<td>opendir</td>
<td>pread</td>
<td>printf</td>
<td>perror</td>
</tr>
<tr>
<td>popen</td>
<td>putc</td>
<td>putchar</td>
<td>putchar_unlocked</td>
</tr>
<tr>
<td>putc</td>
<td>putc_unlocked</td>
<td>puts</td>
<td>putw</td>
</tr>
<tr>
<td>putenv</td>
<td>putpwent</td>
<td>putwchar</td>
<td>putwchar_unlocked</td>
</tr>
<tr>
<td>putwc</td>
<td>putwc_unlocked</td>
<td>qsort</td>
<td>raise</td>
</tr>
<tr>
<td>pwrite</td>
<td>qecvt</td>
<td>qecvt_r</td>
<td>realloc</td>
</tr>
<tr>
<td>qfcvt_r</td>
<td>qgcvt</td>
<td>qsort</td>
<td>re_exec</td>
</tr>
<tr>
<td>rand</td>
<td>random</td>
<td>re_comp</td>
<td>realpath</td>
</tr>
<tr>
<td>read</td>
<td>readv</td>
<td>realloc</td>
<td></td>
</tr>
</tbody>
</table>
regcomp  regerror  regexec  regfree
registerrpc  remove  remque  rename
rewind  rindex  rmdir  scandir
scanf  seed48  setbuf  setbuffer
setegid  setenv  seteuid  setfsent
setgid  setitimer  setjmp  setlocale
setlocale  setlogmask  setmntent  setrlimit
setstate  settyent  setuid  setusershell
setvbuf  sigaction  sigaddset  sigdelset
sigemptyset  sigfillset  sigismember  siglongjmp
signal  sigpending  sigprocmask  sigsuspend
sleep  snprintf  sprintf  srand
rand48  srandom  sscanf  ssignal
stat  stpcpy  stpncpy  strcasecmp
strcat  strchr  strcmp  strcoll
strcpy  strlen  strncasecmp  strncat  strncmp  strncpy  strerror
strerror  strerror_r  strftime  strfmon  strfry  strftime
strtok  strtok  strtok_r  strerror
strverscmp  strxfrm  svcfd_create
swab  swprintf  symlink  syscall
sysconf  tdelete  textdomain  tfind
time  timedgm  timelocal  timezone
tmpfile  toascii  tolower  toupper
towctrans  tolower  towupper  truncate
<table>
<thead>
<tr>
<th>glibc Functions Supported in Catamount [A]</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsearch</td>
</tr>
<tr>
<td>tzset</td>
</tr>
<tr>
<td>ungetc</td>
</tr>
<tr>
<td>usleep</td>
</tr>
<tr>
<td>verr</td>
</tr>
<tr>
<td>vfprintf</td>
</tr>
<tr>
<td>vscanf</td>
</tr>
<tr>
<td>vsprintf</td>
</tr>
<tr>
<td>warn</td>
</tr>
<tr>
<td>wcrtomb</td>
</tr>
<tr>
<td>wcscmp</td>
</tr>
<tr>
<td>wcslen</td>
</tr>
<tr>
<td>wcsncpy</td>
</tr>
<tr>
<td>wcsrchr</td>
</tr>
<tr>
<td>wcstok</td>
</tr>
<tr>
<td>wctomb</td>
</tr>
<tr>
<td>wmemchr</td>
</tr>
<tr>
<td>wmemset</td>
</tr>
<tr>
<td>xdecrypt</td>
</tr>
</tbody>
</table>
The Cray XT3 system provides a set of operating system features that provide users and administrators with a single view of the system (SSV), comparable to that of a traditional Linux workstation.

One such feature is the shared root, which spans all of the service nodes and comprises virtually the entire Linux OS. Only those files that deal with differences in hardware, boot execution, or network configuration are unique to a single node or class of nodes. Consistent with this shared root, the Cray XT3 system maintains a global file system name space for both serial access files (through UFS and NFS) and for parallel access files (through the Lustre parallel file system). User directories and home directories that are maintained on this global file system are visible from all compute nodes and login nodes in the system.

Some of the standard Linux commands are not consistent with SSV. For example the standard `ps` command would list only those processes on the login node on which it is running, not on the entire Cray XT3 system. Cray has replaced some of these commands with Cray XT3 SSV commands

**Note:** The replacement commands have been aliased to the commands they replace, so you need only type, for example, `ps`, to execute the Cray `xtpsh` command.

The following table describes the Linux commands that have been replaced with SSV-compatible commands.

<table>
<thead>
<tr>
<th>Linux or Shell Command</th>
<th>Cray XT3 Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>xthostname</td>
<td>Displays the value in the default xthostname file (<code>/etc/xthostname</code>). The value is set by supplying the name. The xthostname command returns the same value on all login nodes.</td>
</tr>
<tr>
<td>Linux or Shell Command</td>
<td>Cray XT3 Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>kill</td>
<td>xtkill</td>
<td>Allows you to kill a process running on a remote node by specifying the process ID. The xtkill command provides the ability to signal any process in the system, provided the user has sufficient privilege to do so.</td>
</tr>
<tr>
<td>ps</td>
<td>xtps</td>
<td>The xtps command provides process information for all nodes in the system, both for regular processes and compute jobs that are registered with the CPA. For example, you can monitor commands that were initiated from a login session on another login node. The xtps command provides several views of the system also and can correlate information from the system database for more detailed reporting about parallel jobs.</td>
</tr>
<tr>
<td>who</td>
<td>xtwho</td>
<td>Displays the node ID, username, and login time for every user that is logged in to the Cray XT3 system.</td>
</tr>
</tbody>
</table>

For more information on using these XT3 user commands, see the man page for each command.

The following Linux commands are not supported on the Cray XT3 system because their functionality is incongruent with the single-system view:

- **User Information**
  - `w`
  - `finger`
  - `users`

- **Signaling**
  - `killall`
  - `pkill`
  - `skill`
- snice
- renice

• Process Information
  - pstree
  - procinfo
  - top

• System Information
  - vmstat
  - netstat
  - iostat
  - mpstat
  - hostid
  - tload
  - sar
The following table describes the hardware counter presets that are available on the Cray XT3 system. Use these presets to construct an event set as described in Section 8.1.2, page 50.

<table>
<thead>
<tr>
<th>Name</th>
<th>Supported on Cray XT3</th>
<th>Derived from multiple counters?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L3_DCM</td>
<td>No</td>
<td>No</td>
<td>Level 3 data cache misses</td>
</tr>
<tr>
<td>PAPI_L3_ICM</td>
<td>No</td>
<td>No</td>
<td>Level 3 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L1_TCM</td>
<td>Yes</td>
<td>Yes</td>
<td>Level 1 cache misses</td>
</tr>
<tr>
<td>PAPI_L2_TCM</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 cache misses</td>
</tr>
<tr>
<td>PAPI_L3_TCM</td>
<td>No</td>
<td>No</td>
<td>Level 3 cache misses</td>
</tr>
<tr>
<td>PAPI_CA_SNP</td>
<td>No</td>
<td>No</td>
<td>Requests for a snoop</td>
</tr>
<tr>
<td>PAPI_CA_SHR</td>
<td>No</td>
<td>No</td>
<td>Requests for exclusive access to shared cache line</td>
</tr>
<tr>
<td>PAPI_CA_CLN</td>
<td>No</td>
<td>No</td>
<td>Requests for exclusive access to clean cache line</td>
</tr>
<tr>
<td>PAPI_CA_INV</td>
<td>No</td>
<td>No</td>
<td>Requests for cache line invalidation</td>
</tr>
<tr>
<td>PAPI_CA_ITV</td>
<td>No</td>
<td>No</td>
<td>Requests for cache line intervention</td>
</tr>
<tr>
<td>PAPI_L3_LDM</td>
<td>No</td>
<td>No</td>
<td>Level 3 load misses</td>
</tr>
<tr>
<td>PAPI_L3_STM</td>
<td>No</td>
<td>No</td>
<td>Level 3 store misses</td>
</tr>
<tr>
<td>PAPI_BRU_IDL</td>
<td>No</td>
<td>No</td>
<td>Cycles branch units are idle</td>
</tr>
<tr>
<td>Name</td>
<td>Supported on Cray XT3</td>
<td>Derived from multiple counters?</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>---------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>PAPI_FXU_IDL</td>
<td>No</td>
<td>No</td>
<td>Cycles integer units are idle</td>
</tr>
<tr>
<td>PAPI_FPU_IDL</td>
<td>No</td>
<td>No</td>
<td>Cycles floating point units are idle</td>
</tr>
<tr>
<td>PAPI_LSU_IDL</td>
<td>No</td>
<td>No</td>
<td>Cycles load/store units are idle</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>Yes</td>
<td>No</td>
<td>Data translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_TLB_IM</td>
<td>Yes</td>
<td>No</td>
<td>Instruction translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_TLB_TL</td>
<td>Yes</td>
<td>Yes</td>
<td>Total translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_L1_LDM</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 load misses</td>
</tr>
<tr>
<td>PAPI_L1_STM</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 store misses</td>
</tr>
<tr>
<td>PAPI_L2_LDM</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 load misses</td>
</tr>
<tr>
<td>PAPI_L2_STM</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 store misses</td>
</tr>
<tr>
<td>PAPI_BTAC_M</td>
<td>No</td>
<td>No</td>
<td>Branch target address cache misses</td>
</tr>
<tr>
<td>PAPI_PRF_DM</td>
<td>No</td>
<td>No</td>
<td>Data prefetch cache misses</td>
</tr>
<tr>
<td>PAPI_L3_DCH</td>
<td>No</td>
<td>No</td>
<td>Level 3 data cache hits</td>
</tr>
<tr>
<td>PAPI_TLB_SD</td>
<td>No</td>
<td>No</td>
<td>Translation lookaside buffer shotdowns</td>
</tr>
<tr>
<td>PAPI_CSR_FAL</td>
<td>No</td>
<td>No</td>
<td>Failed store conditional instructions</td>
</tr>
<tr>
<td>PAPI_CSR_SUC</td>
<td>No</td>
<td>No</td>
<td>Successful store conditional instructions</td>
</tr>
<tr>
<td>PAPI_CSR_TOT</td>
<td>No</td>
<td>No</td>
<td>Total store conditional instructions</td>
</tr>
<tr>
<td>PAPI_MEM_SCY</td>
<td>Yes</td>
<td>No</td>
<td>Cycles Stalled Waiting for memory accesses</td>
</tr>
<tr>
<td>PAPI_MEM_RCY</td>
<td>No</td>
<td>No</td>
<td>Cycles Stalled Waiting for memory Reads</td>
</tr>
<tr>
<td>Name</td>
<td>Supported on Cray XT3</td>
<td>Derived from multiple counters?</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
<td>---------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>PAPI_MEM_WCY</td>
<td>No</td>
<td>No</td>
<td>Cycles Stalled Waiting for memory writes</td>
</tr>
<tr>
<td>PAPI_STL_ICY</td>
<td>Yes</td>
<td>No</td>
<td>Cycles with no instruction issue</td>
</tr>
<tr>
<td>PAPI_FUL_ICY</td>
<td>No</td>
<td>No</td>
<td>Cycles with maximum instruction issue</td>
</tr>
<tr>
<td>PAPI_STL_CCY</td>
<td>No</td>
<td>No</td>
<td>Cycles with no instructions completed</td>
</tr>
<tr>
<td>PAPI_FUL_CCY</td>
<td>No</td>
<td>No</td>
<td>Cycles with maximum instructions completed</td>
</tr>
<tr>
<td>PAPI_HW_INT</td>
<td>Yes</td>
<td>No</td>
<td>Hardware interrupts</td>
</tr>
<tr>
<td>PAPI_BR_UCN</td>
<td>Yes</td>
<td>No</td>
<td>Unconditional branch instructions</td>
</tr>
<tr>
<td>PAPI_BR_CN</td>
<td>Yes</td>
<td>No</td>
<td>Conditional branch instructions</td>
</tr>
<tr>
<td>PAPI_BR_TKN</td>
<td>Yes</td>
<td>No</td>
<td>Conditional branch instructions taken</td>
</tr>
<tr>
<td>PAPI_BR_NTK</td>
<td>Yes</td>
<td>Yes</td>
<td>Conditional branch instructions not taken</td>
</tr>
<tr>
<td>PAPI_BR_MSP</td>
<td>Yes</td>
<td>No</td>
<td>Conditional branch instructions mispredicted</td>
</tr>
<tr>
<td>PAPI_BR_PRC</td>
<td>Yes</td>
<td>Yes</td>
<td>Conditional branch instructions correctly predicted</td>
</tr>
<tr>
<td>PAPI_FMA_INS</td>
<td>No</td>
<td>No</td>
<td>FMA instructions completed</td>
</tr>
<tr>
<td>PAPI_TOT_IIS</td>
<td>No</td>
<td>No</td>
<td>Instructions issued</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>Yes</td>
<td>No</td>
<td>Instructions completed</td>
</tr>
<tr>
<td>PAPI_INT_INS</td>
<td>No</td>
<td>No</td>
<td>Integer instructions</td>
</tr>
<tr>
<td>PAPI_FP_INS</td>
<td>Yes</td>
<td>No</td>
<td>Floating point instructions</td>
</tr>
<tr>
<td>PAPI_LD_INS</td>
<td>No</td>
<td>No</td>
<td>Load instructions</td>
</tr>
<tr>
<td>PAPI_SR_INS</td>
<td>No</td>
<td>No</td>
<td>Store instructions</td>
</tr>
<tr>
<td>PAPI_BR_INS</td>
<td>Yes</td>
<td>No</td>
<td>Branch instructions</td>
</tr>
<tr>
<td>PAPI_VEC_INS</td>
<td>Yes</td>
<td>No</td>
<td>Vector/SIMD instructions</td>
</tr>
<tr>
<td>Name</td>
<td>Supported on Cray XT3</td>
<td>Derived from multiple counters?</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------</td>
<td>----------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>PAPI_FLOPS</td>
<td>Yes</td>
<td>Yes</td>
<td>Floating point instructions per second</td>
</tr>
<tr>
<td>PAPI_RES_STL</td>
<td>Yes</td>
<td>No</td>
<td>Cycles stalled on any resource</td>
</tr>
<tr>
<td>PAPI_FP_STAL</td>
<td>Yes</td>
<td>No</td>
<td>Cycles in the floating point unit(s) are stalled</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>Yes</td>
<td>No</td>
<td>Total cycles</td>
</tr>
<tr>
<td>PAPI_IPS</td>
<td>Yes</td>
<td>Yes</td>
<td>Instructions per second</td>
</tr>
<tr>
<td>PAPI_LST_INS</td>
<td>No</td>
<td>No</td>
<td>Load/store instructions completed</td>
</tr>
<tr>
<td>PAPI_SYC_INS</td>
<td>No</td>
<td>No</td>
<td>Synchronization instructions completed</td>
</tr>
<tr>
<td>PAPI_L1_DCH</td>
<td>Yes</td>
<td>Yes</td>
<td>Level 1 data cache hits</td>
</tr>
<tr>
<td>PAPI_L2_DCH</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 data cache hits</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 data cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_DCA</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 data cache accesses</td>
</tr>
<tr>
<td>PAPI_L3_DCA</td>
<td>No</td>
<td>No</td>
<td>Level 3 data cache accesses</td>
</tr>
<tr>
<td>PAPI_L1_DCR</td>
<td>No</td>
<td>No</td>
<td>Level 1 data cache reads</td>
</tr>
<tr>
<td>PAPI_L2_DCR</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 data cache reads</td>
</tr>
<tr>
<td>PAPI_L3_DCR</td>
<td>No</td>
<td>No</td>
<td>Level 3 data cache reads</td>
</tr>
<tr>
<td>PAPI_L1_DCW</td>
<td>No</td>
<td>No</td>
<td>Level 1 data cache writes</td>
</tr>
<tr>
<td>PAPI_L2_DCW</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 data cache writes</td>
</tr>
<tr>
<td>PAPI_L3_DCW</td>
<td>No</td>
<td>No</td>
<td>Level 3 data cache writes</td>
</tr>
<tr>
<td>PAPI_L1_ICH</td>
<td>No</td>
<td>No</td>
<td>Level 1 instruction cache hits</td>
</tr>
<tr>
<td>PAPI_L2_ICH</td>
<td>No</td>
<td>No</td>
<td>Level 2 instruction cache hits</td>
</tr>
<tr>
<td>PAPI_L3_ICH</td>
<td>No</td>
<td>No</td>
<td>Level 3 instruction cache hits</td>
</tr>
<tr>
<td>PAPI_L1_ICA</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 instruction cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_ICA</td>
<td>Yes</td>
<td>No</td>
<td>Level 2 instruction cache accesses</td>
</tr>
<tr>
<td>Name</td>
<td>Supported on Cray XT3</td>
<td>Derived from multiple counters?</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------</td>
<td>----------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>PAPI_L3_ICA</td>
<td>No</td>
<td>No</td>
<td>Level 3 instruction cache accesses</td>
</tr>
<tr>
<td>PAPI_L1_ICR</td>
<td>Yes</td>
<td>No</td>
<td>Level 1 instruction cache reads</td>
</tr>
<tr>
<td>PAPI_L2_ICR</td>
<td>No</td>
<td>No</td>
<td>Level 2 instruction cache reads</td>
</tr>
<tr>
<td>PAPI_L3_ICR</td>
<td>No</td>
<td>No</td>
<td>Level 3 instruction cache reads</td>
</tr>
<tr>
<td>PAPI_L1_ICW</td>
<td>No</td>
<td>No</td>
<td>Level 1 instruction cache writes</td>
</tr>
<tr>
<td>PAPI_L2_ICW</td>
<td>No</td>
<td>No</td>
<td>Level 2 instruction cache writes</td>
</tr>
<tr>
<td>PAPI_L3_ICW</td>
<td>No</td>
<td>No</td>
<td>Level 3 instruction cache writes</td>
</tr>
<tr>
<td>PAPI_L1_TCH</td>
<td>No</td>
<td>No</td>
<td>Level 1 total cache hits</td>
</tr>
<tr>
<td>PAPI_L2_TCH</td>
<td>No</td>
<td>No</td>
<td>Level 2 total cache hits</td>
</tr>
<tr>
<td>PAPI_L3_TCH</td>
<td>No</td>
<td>No</td>
<td>Level 3 total cache hits</td>
</tr>
<tr>
<td>PAPI_L1_TCA</td>
<td>Yes</td>
<td>Yes</td>
<td>Level 1 total cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_TCA</td>
<td>No</td>
<td>No</td>
<td>Level 2 total cache accesses</td>
</tr>
<tr>
<td>PAPI_L3_TCA</td>
<td>No</td>
<td>No</td>
<td>Level 3 total cache accesses</td>
</tr>
<tr>
<td>PAPI_L1_TCR</td>
<td>No</td>
<td>No</td>
<td>Level 1 total cache reads</td>
</tr>
<tr>
<td>PAPI_L2_TCR</td>
<td>No</td>
<td>No</td>
<td>Level 2 total cache reads</td>
</tr>
<tr>
<td>PAPI_L3_TCR</td>
<td>No</td>
<td>No</td>
<td>Level 3 total cache reads</td>
</tr>
<tr>
<td>PAPI_L1_TCW</td>
<td>No</td>
<td>No</td>
<td>Level 1 total cache writes</td>
</tr>
<tr>
<td>PAPI_L2_TCW</td>
<td>No</td>
<td>No</td>
<td>Level 2 total cache writes</td>
</tr>
<tr>
<td>PAPI_L3_TCW</td>
<td>No</td>
<td>No</td>
<td>Level 3 total cache writes</td>
</tr>
<tr>
<td>PAPI_FML_INS</td>
<td>Yes</td>
<td>No</td>
<td>Floating point multiply instructions</td>
</tr>
<tr>
<td>PAPI_FAD_INS</td>
<td>Yes</td>
<td>No</td>
<td>Floating point add instructions</td>
</tr>
<tr>
<td>PAPI_FDV_INS</td>
<td>No</td>
<td>No</td>
<td>Floating point divide instructions</td>
</tr>
<tr>
<td>Name</td>
<td>Supported on Cray XT3</td>
<td>Derived from multiple counters?</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PAPI_FSQ_INS</td>
<td>No</td>
<td>No</td>
<td>Floating point square root instructions</td>
</tr>
<tr>
<td>PAPI_FNV_INS</td>
<td>Yes</td>
<td>Yes</td>
<td>Floating point inverse instructions. This event is available only if you compile with the -DDEBUG flag.</td>
</tr>
</tbody>
</table>
blade
1) A Cray XT3 field-replaceable physical entity. A service blade consists of two AMD Opteron sockets, memory, four Cray SeaStar chips, up to four PCI-X cards, and a blade control processor. A compute blade consists of four AMD Opteron sockets, memory, four Cray SeaStar 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 microkernel operating system developed by Sandia National Laboratories and implemented to run on Cray XT3 compute nodes. See also compute node.

class
A group of service nodes of a particular type, such as login or I/O. See also specialization.

compute node
Runs a microkernel and performs only computation. System services cannot run on compute nodes. See also node; service node.

compute processor allocator (CPA)
A program that coordinates with yod to allocate processing elements. See also yod.

deferred implementation
The label used to introduce information about a feature that will not be implemented until a later release.

distributed memory
The kind of memory in a parallel processor where each processor has fast access to its own local memory and where to access another processor’s memory it must send a message via the interprocessor network.
Etnus TotalView
A symbolic source-level debugger designed for debugging the multiple processes of parallel Fortran, C, or C++ programs.

module
See blade.

Modules
A package on a Cray system that allows you to dynamically modify your user environment by using module files. (This term is not related to the module statement of the Fortran language; it is related to setting up the Cray system environment.) The user interface to this package is the module command, which provides a number of capabilities to the user, including loading a module file, unloading a module file, listing which module files are loaded, determining which module files are available, and others.

node
For UNICOS/lc systems, the logical group of processor(s), memory, and network components acting as a network end point on the system interconnection network. See also processing element.

node ID
A decimal number used to reference each individual node. The node ID (NID) can be mapped to a physical location.

processing element
The smallest physical compute group in a Cray XT3 system. The system has two types of processing elements. A compute processing element consists of an AMD Opteron processor, memory, and a link to a Cray SeaStar chip. A service processing element consists of an AMD Opteron processor, memory, a link to a Cray SeaStar chip, and PCI-X links.

service node
A node that performs support functions for applications and system services. Service nodes run SUSE LINUX and perform specialized functions. There are six types of predefined service nodes: login, IO, network, boot, database, and syslog.
**service partition**
The logical group of all service nodes.

**specialization**
The process of setting files on the shared-root file system so that unique files can be present for a node or for a class of node.

**system interconnection network**
The high-speed network that handles all node-to-node data transfers.

**UNICOS/lc**
The operating system for Cray XT3 systems.

**xtshowmesh**
A utility that identifies the state of the nodes.

**yod**
Application launching utility. See also *compute processor allocator (CPA)*.
A
Accounts, 37
ACML, 1, 11
AMD Core Math Library, 11
APIs, 11
Applications
running in parallel, 40
Authentication, 5

B
Batch job
submitting through PBS Pro, 37
Batch Job
MPI program example, 41
using a script to create, 42
Batch processing, 2
BLACS, 1, 11
BLAS, 1, 11

C
C compiler, 1
C++ compiler, 1
Catamount
behavior of stderr under, 24
behavior of stdin under, 24
behavior of stdout under, 24
C Runtime functions in, 69
glibc support, 23
I/O handling, 36
programming considerations, 23, 28
Signal handling, 36
Compiler
C, 1
C++, 1
Fortran, 1
Compilers
commands to invoke, 31
Cray XT3 drivers for, 31

Complier commands, 31
Compute nodes
managing from an MPI program, 36
monitoring status, 33
Compute Processor Allocator (CPA), 34
Core files, 28
Cray Apprentice², 2, 64
Cray MPICH2, 1, 12
limitations, 12
programming considerations, 13
unsupported functions, 12
Cray SHMEM, 1, 17
sample program, 18
Cray XT3 LibSci, 1, 11
CrayPat, 2, 52

D
Debugging, 45

E
Environment variables
using modules to update software, 6
Event set
how to create in PAPI, 50
Examples
basic Cray SHMEM functions, 18
combining results with MPI, 15
MPI work distribution program, 13

F
FFT, 1, 11
File system
Lustre, 2, 25
Fortran compiler, 1

G
getpagesize()
Catamount implementation of, 28
glibc, 2, 11
  runtime functions implemented in
  Catamount, 69
  support in Catamount, 23
GNU C library, 2, 11

H
Hardware Counter Presets
  PAPI, 79
Hardware performance counters, 52
hostname command, 75

I
I/O, 24
  improving performance, 24
  stride I/O functions, 27
Instrumenting a program, 52

J
Job accounting, 37
Job launch
  MPMD application, 35
  sharing nodes, 34
  specifying nodes for, 34
Job scripts, 38
Job status, 39
Jobs
  running, 2

K
Kernel, 23
kill command, 75

L
LAPACK, 1, 11
Launching applications, 33–34
Launching jobs, 2
Libraries, 11
Library
  ACML, 1, 11
  BLACS, 1, 11
  BLAS, 1, 11
Cray MPICH2, 12
Cray SHMEM, 1
Cray XT3 LibSci, 1, 11
FFT, 1, 11
glibc, 11
LAPACK, 1, 11
MPICH2, 1
ScaLAPACK, 1, 11
SuperLU, 1, 11
Lightweight kernel, 23
Linux commands
  unsupported, 76
Loadfile
  launching MPMD applications with, 35
Lustre
  programming considerations, 25
Lustre file system, 2
Lustre library, 25

M
malloc(), 29
  Catamount implementation of, 28
Message Passing Interface, 1
Microkernel, 23
module command, 7
Modules, 6
  installing software with, 6
MPI, 1, 12
  managing compute nodes from, 36
  running program interactively, 41
  running program under PBS Pro, 41
  unsupported functions, 12
mpicc, 31
MPMD applications, 35

O
Optimization, 67

P
PAPI, 2, 49
  counter presets for constructing an event set, 79
  high-level interface, using, 49
Index

low-level interface, using, 50
PATH environment variable
  how to modify, 7
PBS Pro, 2, 37
  Cray specific functions, 40
Performance analysis
  Cray Apprentice^2, 64
  CrayPat, 52
  PAPI, 49
Performance API, 2
PGI compilers, 1
  limitations, 23
Portals interface, 2, 12
Portals message-passing API, 17
Process Control Thread (PCT), 34
Processors
  allocating through qsub, 37
Programming Environment, 1
Project accounting, 37
ps command, 75

Q
qdel command, 40
qstat command, 39

R
Reports
  CrayPat, 52
RSA authentication, 5
  with passphrase, 5
  without passphrase, 6
Running applications, 2, 33–34

S
ScaLAPACK, 1, 11
Scientific libraries, 11
Script
  creating and running a batch job with, 42
Scripts
  PBS Pro, 38
Secure shell, 5
Service nodes
  monitoring status, 33
Shared root
  SSV-compatible commands, 75
Signal handling, 36
Single-system view, 2
Single-system view commands, 75
Software packages
  locations of, 7
ssh, 5
SSV, 2
stderr, 24
stdin, 24
stdio
  improving performance, 24
stdout, 24
SuperLU, 1, 11

T
Timers
  Catamount support for, 28
Timing measurements, 27
TotalView, 45–46
  Cray specific functions, 46

U
User environment, 5

W
who command, 75

X
xthostname command, 75
xtkill command, 75
xtps command, 75
xtshowmesh, 33
xtshowmesh command, 2
xtwho command, 75

Y
yod, 34
I/O handling, 36