

Cray XT™ Programming
Environment User's Guide

S-2396-21



© 2004–2008 Cray Inc. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of Cray Inc.

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

U.S. GOVERNMENT RESTRICTED RIGHTS NOTICE

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

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

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

Cray, LibSci, and UNICOS are federally registered trademarks and Active Manager, Cray Apprentice2, Cray Apprentice2 Desktop, Cray C++ Compiling System, Cray CX1, Cray Fortran Compiler, Cray Linux Environment, Cray SeaStar, Cray SeaStar2, Cray SeaStar2+, Cray SHMEM, Cray Threadstorm, Cray X1, Cray X1E, Cray X2, Cray XD1, Cray XMT, Cray XR1, Cray XT, Cray XT3, Cray XT4, Cray XT5, Cray XT5_h, CrayDoc, CrayPort, CRInform, ECOPhex, Libsci, RapidArray, UNICOS/lc, UNICOS/mk, and UNICOS/mp are trademarks of Cray Inc.

AMD is a trademark of Advanced Micro Devices, Inc. Copyrighted works of Sandia National Laboratories include: Catamount/QK, Compute Processor Allocator (CPA), and xtshowmesh. DDN is a trademark of DataDirect Networks. FFTW is Copyright © 2003 Matteo Frigo, Copyright © 2003 Massachusetts Institute of Technology. GCC is a trademark of the Free Software Foundation, Inc. Intrepid GCCUPC compiler is Copyright © 2007, Intrepid Technology, Inc. Linux is a trademark of Linus Torvalds. Lustre was developed and is maintained by Cluster File Systems, Inc. under the GNU General Public License. MySQL is a trademark of MySQL AB. Opteron is a trademark of Advanced Micro Devices, Inc. PathScale is a trademark of PathScale, Inc. PBS Professional is a trademark of Altair Grid Technologies. PETSc, Copyright, 1995-2004 University of Chicago. The Portland Group and PGI are trademarks of STMicroelectronics. SUSE is a trademark of SUSE LINUX Products GmbH, a Novell business. TotalView is a trademark of TotalView Technologies, LLC. UNIX, the "X device," X Window System, and X/Open are trademarks of The Open Group in the United States and other countries. All other trademarks are the property of their respective owners.

New Features

Quad-core compute nodes

Added support of Cray XT4 quad-core compute nodes (see [Section 4.2.8, page 33](#)).

Dual-socket, quad-core compute nodes

Added support of Cray XT5 dual-socket, quad-core compute nodes (see [Section 4.2.9, page 34](#)).

Huge pages Added support of 2 MB huge pages for CNL applications (see [Section 4.2.10, page 35](#)).

Unified Parallel C (UPC)

Added support of Unified Parallel C (see [Section 3.9, page 25](#), [Section 14.5, page 122](#), and [Section 15.5, page 164](#)).

Memory affinity optimization

Added support of memory affinity optimization tools for Cray XT5 applications (see [Section 13.2, page 110](#) and [Section 14.12, page 144](#)).

CPU affinity optimization

Added support of CPU affinity optimization tools for Cray XT multicore compute nodes (see [Section 13.3, page 111](#) and [Section 14.13, page 146](#)).

MPI topology aware

For CNL applications, MPI now allows each process to create the most optimal messaging path to every other process in the job, based on the topology of the given ranks (see [Section 13.4.1, page 112](#)).

Suppress INFO message

Added the `XTPE_INFO_MESSAGE_OFF` environment variable to allow you to suppress the output of the target architecture INFO message (see [Section 5.2, page 49](#)).

CrayPat module name change

Documented CrayPat module name change from `craypat` to `xt-craypat` (see [Section 12.2, page 104](#)).

PAPI module name change

Documented PAPI module name change from `papi` to `xt-papi` (see [Section 12.1, page 103](#)).

PETSc

Documented the external packages that the Cray implementation of PETSc is configured to support (see [Section 3.3, page 17](#)). Added a PETSc example (see [Section 14.6, page 123](#)).

Killing processes

Documented support of the `kill` command and the `kill()` system call on CNL compute nodes (see [Section 4.2.6, page 33](#)). Documented support of the `kill()` system call on Catamount compute nodes (see [Section 4.3.6, page 47](#)).

Unsupported PGI compiler options

Added note that the `-mpfi`, `-mpfo`, and `-mconcur` PGI compiler command options are not supported under Catamount (see [Section 4.3.11, page 48](#)).

Running user programs on service nodes

Documented the process for compiling and running user programs on service nodes (see [Chapter 9, page 85](#)).

Improving memory allocation

Documented the use of environment variables to make memory allocation more efficient (see [Section 4.2.11, page 37](#)).

New job and node status display command

Documented the use of the `xtnodestat` command to display current job and node status (see [Chapter 6, page 57](#)).

New core status display

Documented the use of the `apstat -n` option to display core status (see [Section 7.2, page 69](#)).

Core dump file may be truncated

Documented conditions under which a core dump file for a Catamount application may be truncated without user notification (see [Section 4.3.7, page 47](#)).

CRAFFT library

Documented support of the CRay Adaptive Fast Fourier Transform (CRAFFT) library (see [Section 3.2.5, page 16](#)).

Fast_mv library

Documented support of the Fast_mv library of high- performance math intrinsic functions (see [Section 3.6, page 20](#)).

lgdb debugger

(Deferred implementation) Documented support of lgdb, the GNU debugger for CNL applications (see [Section 11.2, page 99](#)).

Record of Revision

| <i>Version</i> | <i>Description</i> |
|----------------|--|
| 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. |
| 1.2 | August 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.2 and UNICOS/lc 1.2 releases. |
| 1.3 | November 2005 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.3 and UNICOS/lc 1.3 releases. |
| 1.4 | April 2006 Supports Cray XT3 systems running the Cray XT3 Programming Environment 1.4 and UNICOS/lc 1.4 releases. |
| 1.5 | August 2006 Supports limited availability (LA) release of Cray XT systems running the Cray XT Programming Environment 1.5 and Cray Linux Environment (CLE)1.5 releases. |
| 1.5 | November 2006 Supports general availability (GA) release of Cray XT systems running the Cray XT Programming Environment 1.5 and UNICOS/lc 1.5 releases. |
| 2.0 | June 2007 Supports limited availability (LA) release of Cray XT systems running the Cray XT Programming Environment 2.0 and UNICOS/lc 2.0 releases. |
| 2.0 | October 2007 Supports general availability (GA) release of Cray XT systems running the Cray XT Programming Environment 2.0 and UNICOS/lc 2.0 releases. |

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

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

Contents

| | <i>Page</i> |
|---|-------------|
| Preface | xi |
| Accessing Product Documentation | xi |
| Conventions | xii |
| Reader Comments | xiii |
| Cray User Group | xiii |
| About the Cray XT Development Environment [1] | 1 |
| About the Cray XT System Environment | 1 |
| About the Cray XT Programming Environment | 2 |
| About Cray XT Documentation | 4 |
| Setting Up Your Environment [2] | 7 |
| Setting Up a Secure Shell | 7 |
| Setting up RSA Authentication with a Passphrase | 7 |
| Setting up RSA Authentication without a Passphrase | 8 |
| Using Modules | 9 |
| Modifying the PATH Variable | 10 |
| Using the Lustre File System | 11 |
| About Libraries and Functions [3] | 13 |
| About the C Language Run Time Library | 13 |
| About the Cray Scientific Library | 13 |
| About the BLAS and LAPACK Libraries | 13 |
| About the ScaLAPACK and BLACS Libraries | 14 |
| About the Iterative Refinement Toolkit (IRT) | 15 |
| About the SuperLU Library | 16 |
| About the Cray Adaptive Fast Fourier Transform (CRAFFT) Library | 16 |

| | <i>Page</i> |
|---|-------------|
| About the PETSc Library | 17 |
| About the AMD Core Math Library (ACML) | 19 |
| About the FFTW Libraries | 19 |
| About the Fast_mv Library | 20 |
| About the Cray MPT Library | 21 |
| About the MPICH2 Library | 21 |
| About the SHMEM Library | 22 |
| About the OpenMP Library | 24 |
| About the UPC Functions | 25 |
| Cray XT Programming Considerations [4] | 27 |
| Programming Considerations for all Developers | 27 |
| About PGI Compilers | 27 |
| About Default MPICH2 and SHMEM Libraries | 27 |
| About Unsupported C++ Header Files | 28 |
| About Restrictions on Large Data Objects | 28 |
| About the FORTRAN STOP Message | 28 |
| Suppressing PGI Vectorization | 29 |
| About the PGI Debugger | 29 |
| About the PathScale Fortran Compiler | 29 |
| About Little-endian Support | 30 |
| About the Portals Message Size Limit | 30 |
| About Shared Libraries | 30 |
| Programming Considerations for CNL Users | 30 |
| About CNL glibc Functions | 30 |
| About I/O Support Operations under CNL | 31 |
| Connecting to External Services under CNL | 32 |
| About Timing Functions under CNL | 32 |
| About Signal Support under CNL | 32 |
| Killing Processes under CNL | 33 |
| About Core Files under CNL | 33 |

| | <i>Page</i> |
|---|-------------|
| Using Cray XT4 Quad-core Processors | 33 |
| Using Cray XT5 Compute Nodes | 34 |
| Using Huge Pages and Base Pages under CNL | 35 |
| Allocating Memory under CNL | 37 |
| About Resource Limits under CNL | 39 |
| About the One Application Per Node Limitation under CNL | 39 |
| About Parallel Programming Models under CNL | 39 |
| About the Modified Copy-on-write Process under CNL | 39 |
| About Unsupported PGI Compiler Command Options under CNL | 40 |
| Programming Considerations for Catamount Users | 40 |
| About Catamount glibc Functions | 40 |
| About I/O Support Functions under Catamount | 42 |
| Improving Fortran I/O Performance under Catamount | 42 |
| Improving C++ I/O Performance under Catamount | 43 |
| Improving <code>stdio</code> Performance under Catamount | 43 |
| Improving the Performance of Large File, Sequential I/O under Catamount | 43 |
| Using Stride I/O Functions to Improve Performance under Catamount | 44 |
| Reducing Memory Fragmentation under Catamount | 45 |
| About the Limitations of External Connectivity under Catamount | 45 |
| About Timing Functions under Catamount | 45 |
| About Signal Support under Catamount | 46 |
| Killing Processes under Catamount | 47 |
| About Core Files under Catamount | 47 |
| Changing Page Size under Catamount | 47 |
| About Resource Limits under Catamount | 48 |
| About the Limitations on Parallel Programming Models under Catamount | 48 |
| About Unsupported PGI Compiler Command Options under Catamount | 48 |
| Using Compilers [5] | 49 |
| Setting Your Target Architecture | 49 |
| Using the Compiler Driver Commands | 49 |

| | <i>Page</i> |
|--|-------------|
| Using PGI Compilers | 50 |
| Using GNU Compilers | 52 |
| Using PathScale Compilers | 53 |
| Getting Compute Node Status [6] | 57 |
| Running CNL Applications [7] | 61 |
| Using the <code>aprun</code> Command | 61 |
| Using the <code>apstat</code> Command | 69 |
| Using the <code>cnselect</code> Command | 71 |
| Understanding How Much Memory is Available to CNL Applications | 72 |
| Launching an MPMD Application | 73 |
| Managing Compute Node Processors from an MPI Program | 74 |
| About <code>aprun</code> Input and Output Modes | 74 |
| About <code>aprun</code> Resource Limits | 75 |
| About <code>aprun</code> Signal Processing | 76 |
| Running Catamount Applications [8] | 77 |
| Using the <code>yod</code> Command | 77 |
| Using the <code>cnselect</code> Command | 78 |
| Understanding How Much Memory is Available to Catamount Applications | 79 |
| Launching an MPMD Application | 80 |
| Managing Compute Node Processors from an MPI Program | 82 |
| Using Input and Out Modes under <code>yod</code> | 82 |
| About <code>yod</code> Signal Handling | 82 |
| Associating a Project or Task with a Job Launch | 83 |
| Running User Programs on Service Nodes [9] | 85 |
| Using PBS Professional [10] | 87 |
| Creating Job Scripts | 87 |
| Submitting Batch Jobs | 88 |

| | <i>Page</i> |
|---|-------------|
| Using <code>aprun</code> with <code>qsub</code> | 88 |
| Using <code>yod</code> with <code>qsub</code> | 89 |
| Terminating Failing Processes in an MPI Program | 89 |
| Getting Job Status | 90 |
| Removing a Job from the Queue | 91 |
| Debugging an Application [11] | 93 |
| Using the TotalView Debugger | 93 |
| Using TotalView to Debug an Application | 94 |
| Using TotalView to Debug a Core File | 96 |
| Using TotalView to Attach to a Running Process | 97 |
| Using TotalView to Alter Standard I/O | 98 |
| About the Limitations of TotalView on Cray XT Systems | 99 |
| Using the GNU Debugger | 99 |
| Using the <code>lgdb</code> Debugger | 100 |
| Using the <code>xtgdb</code> Debugger | 100 |
| Troubleshooting Catamount Application Failures | 101 |
| Analyzing Performance [12] | 103 |
| Using the Performance API (PAPI) | 103 |
| Using the High-level PAPI Interface | 103 |
| Using the Low-level PAPI Interface | 104 |
| Using the Cray Performance Analysis Tool (CrayPat) | 104 |
| Running Tracing and Sampling Experiments | 106 |
| Visualizing Performance Data | 108 |
| Optimizing Applications [13] | 109 |
| Using Compiler Optimization Options | 109 |
| Using <code>aprun</code> Memory Affinity Options | 110 |
| Using <code>aprun</code> CPU Affinity Optimizations | 111 |
| Optimizing Process Placement on Multicore Nodes | 112 |
| Optimizing MPI and SHMEM Applications Running under CNL | 112 |

| | <i>Page</i> |
|---|-------------|
| Optimizing MPI and SHMEM Applications Running under Catamount | 113 |
| Example CNL Applications [14] | 115 |
| Running a Basic Application under CNL | 115 |
| Running an MPI Application under CNL | 116 |
| Using the Cray <code>shmem_put</code> Function under CNL | 118 |
| Using the Cray <code>shmem_get</code> Function under CNL | 120 |
| Running a UPC Application under CNL | 122 |
| Running a PETSc Application under CNL | 123 |
| Running an OpenMP Application under CNL | 135 |
| Running a PBS Professional Interactive Job under CNL | 138 |
| Running a PBS Professional Job Script under CNL | 139 |
| Running Multiple Sequential Applications under CNL | 140 |
| Running Multiple Parallel Applications under CNL | 142 |
| Using <code>aprun</code> Memory Affinity Options | 144 |
| Using the <code>aprun -S</code> Option | 144 |
| Using the <code>aprun -sl</code> Option | 144 |
| Using the <code>aprun -sn</code> Option | 145 |
| Using the <code>aprun -ss</code> Option | 145 |
| Using <code>aprun</code> CPU Affinity Options | 146 |
| Using the <code>aprun -cc cpu_list</code> Option | 146 |
| Using the <code>aprun -cc keyword</code> Options | 147 |
| Running Compute Node Commands under CNL | 147 |
| Using the High-level PAPI Interface under CNL | 148 |
| Using the Low-level PAPI Interface under CNL | 149 |
| Using CrayPat under CNL | 150 |
| Using Cray Apprentice2 under CNL | 156 |
| Example Catamount Applications [15] | 157 |
| Running a Basic Application under Catamount | 157 |
| Running an MPI Application under Catamount | 158 |

| | <i>Page</i> |
|--|-------------|
| Using the Cray <code>shmem_put</code> Function under Catamount | 160 |
| Using the Cray <code>shmem_get</code> Function under Catamount | 162 |
| Running a UPC Application under Catamount | 164 |
| Using <code>dclock()</code> to Calculate Elapsed Time under Catamount | 165 |
| Specifying a Buffer for I/O under Catamount | 166 |
| Changing the Default Buffer Size for I/O-to-file Streams under Catamount | 167 |
| Improving the Performance of <code>stdout</code> under Catamount | 169 |
| Running a PBS Professional Job Script under Catamount | 171 |
| Running Multiple Sequential Applications under Catamount | 172 |
| Running Multiple Parallel Applications under Catamount | 174 |
| Using <code>xtgdb</code> under Catamount | 175 |
| Using the High-level PAPI Interface under Catamount | 176 |
| Using the Low-level PAPI Interface under Catamount | 178 |
| Using CrayPat under Catamount | 179 |
| Appendix A glibc Functions Supported in CNL | 187 |
| Appendix B glibc Functions Supported in Catamount | 193 |
| Appendix C PAPI Hardware Counter Presets | 199 |
| Appendix D MPI Error Messages | 207 |
| Appendix E ALPS Error Messages | 209 |
| Appendix F <code>yod</code> Error Messages | 211 |
| Appendix G PETSc Makefiles | 215 |
| Glossary | 217 |
| Index | 221 |
| Figures | |
| Figure 1. Cray XT5 Compute Node | 34 |

| | <i>Page</i> |
|--|-------------|
| Figure 2. TotalView Root Window | 94 |
| Figure 3. TotalView Process Window | 95 |
| Figure 4. Debugging a Core File | 96 |
| Figure 5. Attaching to a Running Process | 97 |
| Figure 6. Altering Standard I/O | 98 |
| Figure 7. Cray Apprentice2 Display | 156 |

Tables

| | |
|---|-----|
| Table 1. Manuals and Man Pages Included with This Release | 4 |
| Table 2. PGI Compiler Commands | 51 |
| Table 3. GNU Compiler Commands | 53 |
| Table 4. PathScale Compiler Commands | 54 |
| Table 5. aprun versus qsub Options | 88 |
| Table 6. yod versus qsub Options | 89 |
| Table 7. RPCs to yod | 101 |
| Table 8. Supported glibc Functions for CNL | 187 |
| Table 9. Supported glibc Functions for Catamount | 193 |
| Table 10. PAPI Presets | 199 |
| Table 11. MPI Error Messages | 207 |
| Table 12. ALPS Error Messages | 209 |
| Table 13. yod Error Messages | 211 |

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:

- CrayPort** CrayPort is the external Cray website for registered users that offers documentation for each product. CrayPort has portal pages for each product that contains links to all of the documentation that is associated to that particular product. CrayPort allows you to quickly access and search Cray books, man pages, and in some cases, third-party documentation. You access CrayPort using the following URL:
- crayport.cray.com
- CrayDoc** CrayDoc is the Cray documentation delivery system. CrayDoc allows you to quickly access and search Cray books, man pages, and in some cases, third-party documentation. Access the HTML and PDF documentation via CrayDoc at the following locations.
- The local network location defined by your system administrator
 - The CrayDoc public website: docs.cray.com
- Man pages** Man pages are textual help files available from the command line on Cray machines. To access man pages, enter 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**
- Third-party documentation that is not provided through CrayPort or CrayDoc is included as part of the third-party product.

## Conventions

These conventions are used throughout Cray documentation:

| <u>Convention</u>    | <u>Meaning</u>                                                                                                                                                                                                                        |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>command</code> | This fixed-space font denotes literal items, such as file names, pathnames, man page names, command names, and programming language elements.                                                                                         |
| <i>variable</i>      | Italic typeface indicates an element that you will replace with a specific value. For instance, you may replace <i>filename</i> with the name <code>datafile</code> in your program. It also denotes a word or concept being defined. |
| <b>user input</b>    | This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.                                                                                         |
| [ ]                  | Brackets enclose optional portions of a syntax representation for a command, library routine, system call, and so on.                                                                                                                 |
| . . .                | Ellipses indicate that a preceding element can be repeated.                                                                                                                                                                           |
| name(N)              | 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.                                                                    |

Section numbers are used to group man pages into categories, as defined by the usage of the commands in that section. For example, section 1 man pages are typically user commands and section 8 man pages are typically system administrator commands.

To find the meaning of each section number for your particular system, enter the following command:

```
% man man
```

## 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](mailto: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:**

Customer Documentation  
Cray Inc.  
1340 Mendota Heights Road  
Mendota Heights, MN 55120-1128  
USA

## Cray User Group

The Cray User Group (CUG) is an independent, volunteer-organized international corporation of member organizations that own or use Cray Inc. computer systems. CUG facilitates information exchange among users of Cray systems through technical papers, platform-specific e-mail lists, workshops, and conferences. CUG memberships are by site and include a significant percentage of Cray computer installations worldwide. For more information, contact your Cray site analyst or visit the CUG website at [www.cug.org](http://www.cug.org).



# About the Cray XT Development Environment [1]

---

This guide describes the Cray XT 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 build and run optimized applications. The intended audience is application programmers and users of Cray XT systems. Prerequisite knowledge is a familiarity with the topics in the *Cray XT System Overview*. For information about managing system resources, system administrators can use the *Cray XT System Management* manual.

The UNICOS/lc operating system was renamed Cray Linux Environment (CLE). Documentation associated with this release may use the terms UNICOS/lc and CLE interchangeably. The transition to the CLE name will be complete in the next release.

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

## 1.1 About the Cray XT System Environment

The system on which you run your Cray XT applications is an integrated set of Cray XT compute node and service node components. You log in to a Cray XT login node and use the Cray XT Programming Environment and related products to create your executables. You run your executables on Cray XT compute nodes.

The operating system is Cray Linux Environment (CLE); it has compute node and service node components. Compute nodes run either the CNL or the Catamount operating system. Service nodes run SUSE LINUX. For details about the differences between CNL and Catamount, see [Section 4.2, page 30](#).

## 1.2 About the Cray XT Programming Environment

The Cray XT Programming Environment includes these products and services:

- PGI compilers for C, C++, and Fortran (see [Section 5.2.1, page 50](#)).
- GNU compilers for C, C++, and Fortran (see [Section 5.2.2, page 52](#)).
- PathScale compilers for C, C++, and Fortran (see [Section 5.2.3, page 53](#)).
- Parallel programming models:
  - Cray Message Passing Toolkit (MPT). MPT consists of the MPI and SHMEM libraries (see [Section 3.7, page 21](#)).
  - OpenMP shared memory model routines, Fortran directives, and C and C++ pragmas (see [Section 3.8, page 24](#)). OpenMP is not supported for applications running under Catamount.
  - Unified Parallel C (UPC) (see [Section 3.9, page 25](#)).
- Cray XT-LibSci scientific library, which includes:
  - Basic Linear Algebra Subprograms (BLAS)
  - Linear Algebra (LAPACK) routines
  - ScaLAPACK routines
  - Basic Linear Algebra Communication Subprograms (BLACS)
  - Iterative Refinement Toolkit (IRT) routines
  - SuperLU routines
  - Cray Adaptive Fast Fourier Transform (CRAFFT) routines

For further information about Cray XT-LibSci, see [Section 3.2, page 13](#).

- PETSc (Portable, Extensible Toolkit for Scientific Computation). For further information, see [Section 3.3, page 17](#).
- AMD Core Math Library (ACML), which includes:
  - Fast Fourier Transform (FFT) routines
  - Math transcendental library routines
  - Random number generators
  - GNU Fortran libraries

For further information about ACML, see [Section 3.4, page 19](#).

- FFTW (Fastest Fourier Transform in the West) routines (see [Section 3.5, page 19](#))
- Fast\_mv library of high-performance math intrinsic functions (see [Section 3.6, page 20](#)).
- A subset of the glibc GNU C Library routines for compute node applications (see [Section 3.1, page 13](#)).
- The Performance API (PAPI) (see [Section 12.1, page 103](#)).

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

- The Application Level Placement Scheduler (ALPS) utility for launching applications on CNL compute nodes (`aprun` command), getting status about applications and cores (`apstat` command), and killing processes (`apkill` command). See [Section 7.1, page 61](#) for a description of `aprun`, [Section 7.2, page 69](#) for a description of `apstat`, and [Appendix E, page 209](#) for a description of common ALPS error messages.
- The `yod` command for launching applications on Catamount compute nodes (see [Section 4.2, page 30](#)).
- The `cnselect` command for generating a candidate list of compute nodes based on user-specified selection criteria. For CNL applications, you can use this list on the `aprun -L node_list` option or the `qsub -lmpnodes` option. For Catamount applications, you can use this list on the `yod -list processor-list` option. See [Section 7.3, page 71](#).
- Lustre parallel file system (see [Section 2.4, page 11](#)).
- The `xtprocadmin -A` command for generating a report showing the attributes of the compute nodes (see [Chapter 6, page 57](#)).
- The `xtnodestat` command for generating reports showing the status of jobs and nodes (see [Chapter 6, page 57](#)).

These optional products are available for Cray XT systems:

- PBS Professional batch processing system (see [Chapter 10, page 87](#)).  
**Note:** If your site has installed another batch system, please contact the appropriate vendor for the necessary installation, configuration, and administration information. For example, contact Cluster Resources, Inc. (<http://www.clusterresources.com/>) for documentation specific to Moab products.
- TotalView debugger (see [Section 11.1, page 93](#)). TotalView debugger documentation is available from TotalView Technologies, LLC (<http://www.totalviewtech.com/Documentation/>).
- GNU debuggers: `lgdb` for CNL applications (Deferred implementation) and `xtgdb` for Catamount applications (see [Section 11.2, page 99](#)).
- CrayPat performance analysis tools (see [Section 12.2, page 104](#)).
- Cray Apprentice2 performance visualization tool (see [Section 12.3, page 108](#)).

### 1.3 About Cray XT Documentation

**Table 1** lists the manuals and man pages that are provided with this release. All manuals are provided as PDF files, and some are also 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 XT System Documentation Site Map on CrayDoc to link to all Cray manuals and man pages included with this release.

Table 1. Manuals and Man Pages Included with This Release

---

|                                                                                           |
|-------------------------------------------------------------------------------------------|
| <i>Cray XT Programming Environment User's Guide</i> (this manual)                         |
| <i>Cray XT Programming Environment man pages</i> (cc(1), CC(1), ftm(1), aprun(1), yod(1)) |
| <i>Executable and Linking Format elf(5) man page</i>                                      |
| <i>Cray XT System Software Release Overview</i>                                           |
| <i>Cray XT System Overview</i>                                                            |
| <i>PGI User's Guide</i>                                                                   |
| <i>PGI Fortran Reference</i>                                                              |

*PGI Tools Guide*

*Cray Programming Environments Installation Guide*

*Modules software package man pages*

*Cray MPICH2 man pages* (read `intro_mpi(3)` first)

*Cray SHMEM man pages* (read `intro_shmem(3)` first)

*Cray XT-LibSci man pages* (read `intro_libsci(3s)` first)

*SuperLU Users' Guide*

*Iterative Refinement Toolkit man pages* (read `intro_irt(3)` first)

*Cray Adaptive Fast Fourier Transform (CRAFFT) man pages* (read `intro_crafft(3s)` first)

*AMD Core Math Library (ACML) manual*

*PETSc man pages* (see

<http://www-unix.mcs.anl.gov/petsc/petsc-as/documentation/index.html>)

*FFT man pages* (`intro_fft(3)`, `intro_fftw2(3)`, `intro_fftw3(3)`)

*Fast\_mv library* (read `intro_fast_mv(3)` first)

*PBS Professional 9.0 Quick Start Guide*

*PBS Professional 9.0 User's Guide*

*TotalView man page* (`totalview(1)`)

*Performance API (PAPI) man pages*

*Using Cray Performance Analysis Tools guide*

*CrayPat and Cray Apprentice2 man pages* (read `intro_craypat(1)` and `app2(1)` first)

---

Additional sources of information:

- PGI manuals at <http://www.pgroup.com> and the `pgcc(1)`, `pgCC(1)`, and `pgf95(1)` man pages available through the `man` command.
- *Using the GNU Compiler Collection (GCC)* manual at <http://gcc.gnu.org/> and the `gcc(1)`, `g++(1)`, and `gfortran(1)` man pages available through the `man` command.
- *PathScale Compiler Suite User's Guide* at <http://www.pathscale.com/> and the `pathcc(1)`, `pathCC(1)`, `pathf95(1)`, and `eko(7)` man pages available through the `man` command.

- MPICH2 documents at <http://www-unix.mcs.anl.gov/mpi/mpich2/> and <http://www.mpi-forum.org>.
- OpenMP documents at <http://www.openmp.org>.
- Unified Parallel C (UPC) documents: Berkeley UPC website (<http://upc.lbl.gov/docs/>) and Intrepid UPC website ([http://www.intrepid.com/upc/cray\\_xt3\\_upc.html](http://www.intrepid.com/upc/cray_xt3_upc.html)).
- The *ScaLAPACK Users' Guide* at <http://www.netlib.org/scalapack/slug/>.
- SuperLU documents at <http://crd.lbl.gov/~xiaoye/SuperLU/>.
- PETSc documents at <http://www-unix.mcs.anl.gov/petsc/petsc-as/documentation/index.html>
- FFTW documents at <http://www.fftw.org/>.
- PAPI documents at <http://icl.cs.utk.edu/papi/>.
- Lustre documentation at <http://manual.lustre.org/>.
- SUSE LINUX man pages available through the `man` command.

# Setting Up Your Environment [2]

---

Configuring your user environment on a Cray XT system is similar to configuring a typical Linux workstation. However, there are steps specific to Cray XT systems that you must take before you begin developing applications.

## 2.1 Setting Up a Secure Shell

Cray XT 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 must generate an RSA authentication key. The process for generating the key depends on the authentication method you use. There are two methods of passwordless authentication: with or without a passphrase. Although both methods are described here, you must use the latter method to access the compute nodes through a script or when using a system monitor command such as `xtps`.

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. For further information about system monitor commands, see the *Cray XT System Management* manual.

### 2.1.1 Setting up RSA Authentication with a Passphrase

To enable `ssh` with a passphrase, complete these steps.

1. Create a `$HOME/.ssh` directory and set permissions so that only the file's owner can access them.

```
% mkdir $HOME/.ssh
% chmod 700 $HOME/.ssh
```

2. Generate the RSA keys.

```
% ssh-keygen -t rsa
```

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

3. The public key is stored in your `$HOME/.ssh` directory. Copy the key to your home directory on the remote host(s).

```
% scp $HOME/.ssh/key_filename.pub \
username@system_name:~/.ssh/authorized_keys
```

4. Connect to the remote host.

If you are using a C shell, use:

```
% eval `ssh-agent`
```

```
% ssh-add
```

If you are using a Bourne shell, use:

```
$ eval `ssh-agent -s`
```

```
$ ssh-add
```

5. Type your passphrase when prompted.
6. Connect to the remote host.

```
% ssh remote_host_name
```

### 2.1.2 Setting up RSA Authentication without a Passphrase

To enable `ssh` without a passphrase, complete these steps.

1. Create a `$HOME/.ssh` directory and set permissions so that only the owner of the file can access them.

```
% mkdir $HOME/.ssh
% chmod 700 $HOME/.ssh
```

2. Generate the RSA keys.

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

Following the prompts.

3. Copy the public key to your home directory on the remote host(s).

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

```
% scp $HOME/.ssh/key_filename.pub \
username@system_name:~/.ssh/authorized_keys
```

4. Connect to the remote host.

```
% ssh remote_host_name
```

## 2.2 Using Modules

The Cray XT system uses 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. You can use the default version of an application, or you can choose another version by using Modules system commands.

At login, the `PrgEnv-pgi` and `Base-opts` modules are loaded into your user environment. You should never unload the `Base-opts` module because it contains the setup for CLE.

To change compiler environments, swap out the `PrgEnv-pgi` module, leaving the `Base-opts` module in place. For example:

```
% module swap PrgEnv-pgi PrgEnv-gnu
```

or

```
% module swap PrgEnv-pgi PrgEnv-pathscale
```

The target environment module is automatically loaded at log in. If the compute nodes are running CNL, the `xtpe-target-cn1` module is automatically loaded. If the compute nodes are running Catamount, the `xtpe-target-catamount` module is automatically loaded.

For some products, additional modules may have to be loaded. The chapters addressing those products and the related example programs specify the module names and the conditions under which they must be loaded.

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.

To find out what modules have been loaded, use:

```
% module list
```

To get a list of all available modules, use:

```
% module avail
```

**Note:** Executing `module avail` produces an alphabetical list of every modulefile in your `module use path`. It has no option for "grepping." If you are on a large system, executing `module avail` with no argument may produce several screens of module list items because `module avail` lists every version of every modulefile alphabetically. In such cases, it is better to use an argument, such as `module avail xt`; doing this allows you to get a list of modules in a class within a product name. For example:

```
% module avail xt
```

displays all the modulefiles named `xt*`.

```
% module avail xt-pe
```

displays just the `xt-pe` modulefiles.

```
% module avail a
```

lists all modulefiles that start with an `a`.

For further information about the Module utility, see the `module(1)` and `modulefile(4)` man pages.

## 2.3 Modifying the PATH Variable

You may need to modify the `PATH` variable for your environment. 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 `ssh`, use:

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

If you are using bash, use:

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

## 2.4 Using the Lustre File System

Lustre is the Cray XT parallel file system. To use Lustre, you must direct file operations to paths within a Lustre mount point. You can use the `df -t lustre` or `lfs df` command to locate Lustre mount points:

```
% lfs df
UUID 1K-blocks Used Available Use% Mounted on
nid00008_mds_UUID 179181084 11508832 167672252 6% /lus/nid00008[MDT:0]
ost1_UUID 1666447096 529153360 1137293736 31% /lus/nid00008[OST:0]
ost2_UUID 1666447096 540479248 1125967848 32% /lus/nid00008[OST:1]
<snip>
filesystem summary: 26663153536 8644943268 18018210268 32% /lus/nid00008
```

If your environment has not been set up to use Lustre for I/O, see your system administrator. The Lustre I/O interface is transparent to the application programmer; I/O functions are handled by the Lustre client running on the compute nodes.

If you want to create a file with a specific striping pattern, use the Lustre `lfs` command. Lustre file systems include Object Storage Servers (OSSs). Each OSS hosts Object Storage Targets (OSTs), which transfer data objects that can be striped across Redundant Array of Independent Disks (RAID) storage devices.

You may choose to create a file of multiple stripes if your application requires a higher transmission rate to a single file than can be provided by a single OSS. You may also need to stripe a file if a single OST does not have enough free space to hold the entire file. For example, the command:

```
% lfs setstripe results2 1048576 1 4
```

stripes file `results2` on four OSTs, (starting with `ost1`). The stripe size is 1048576 bytes.

For further information, see the `lfs(1)` man page.



# About Libraries and Functions [3]

---

The Cray XT development environment the following libraries and functions.

## 3.1 About the C Language Run Time Library

The Cray XT supports subsets of the GNU C library, `glibc`, for CNL and Catamount applications. For details on `glibc` for CNL, see [Section 4.2.1, page 30](#) and [Appendix A, page 187](#). For details on the Catamount port of `glibc`, see [Section 4.3.1, page 40](#) and [Appendix B, page 193](#).

## 3.2 About the Cray Scientific Library

The Cray XT scientific library, `XT-LibSci`, includes:

- Basic Linear Algebra Subroutines (BLAS)
- Linear algebra routines (LAPACK)
- Parallel linear algebra routines (ScaLAPACK)
- Basic Linear Algebra Communication Subprograms (BLACS)
- Iterative Refinement Toolkit (IRT)
- SuperLU sparse solver routines
- CRay Adaptive Fast Fourier Transform (CRAFFT) routines

**Note:** By default, the `xt-libsci` module is loaded. Use this module for `XT-LibSci` single- and dual-core applications. For `XT-LibSci` quad-core, single-thread applications, load the `xtpc-quadcore` module. For `LibSci` quad-core, multi-thread applications, load the `xtpc-quadcore` module and include `-lsci_mp` on the compiler command line.

For additional information about `XT-LibSci` routines, see the scientific libraries man pages (read `intro_libsci(3s)` first).

### 3.2.1 About the BLAS and LAPACK Libraries

The BLAS and LAPACK libraries include routines from the 64-bit `libGoto` library from the University of Texas.

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

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

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

**Note:** C programmers using the Fortran interface must order arrays in the Fortran column-major manner.

### 3.2.2 About the ScaLAPACK and BLACS Libraries

ScaLAPACK is a distributed-memory, parallel linear algebra library. The XT-LibSci version of ScaLAPACK is modified to work more efficiently on Cray XT compute nodes.

The BLACS library is a set of communication routines used by ScaLAPACK and the user to set up a problem and handle the communications.

The ScaLAPACK and BLACS libraries can be used in MPI and SHMEM applications. Cray XT-LibSci under CNL also supports hybrid MPI/ScaLAPACK applications, which use threaded BLAS on a compute node and MPI between nodes. To use ScaLAPACK in a hybrid application:

1. Adjust the process grid dimensions in ScaLAPACK to account for the decrease in BLACS nodes.
2. Ensure that the number of BLACS processes required is equal to the number of nodes required, **not** the number of cores.
3. Set the `GOTO_NUM_THREADS`.

To run a ScaLAPACK application in regular mode (that is, 1 MPI process per core) with 16 BLACS processes on a 4x4 computational grid, use the `#PBS -lmpwidth` option to specify the number of processing elements needed (16) and the `#PBS -lmpnppn` option to specify the number of PEs per node (2).

```
#!/usr/bin/csh

#PBS -lmpwidth=16
#PBS -lmpnppn=2
cd /lus/nid00007/user1
aprun -n 16 -N 2 ./scalapack1
```

To run the same job using a hybrid application, first reduce the number of BLACS processes from 16 to 8 (by specifying either a 2x4 or possibly a 4x2 computational grid). The additional parallelism within a node is provided through use of the threaded BLAS.

In the PBS script, only those tasks actually recognized are requested. So set `mppwidth` equal to the number of nodes required (8) and `mppnppn` equal to the number of PEs per node (1).

```
#!/usr/bin/csh

#PBS -l mppwidth=8
#PBS -l mppnppn=1
#PBS -l mppdepth=2
cd /lus/nid00007/user1
setenv GOTO_NUM_THREADS 2
aprun -n 8 -N 1 -d 2 ./scalapack1
```

### 3.2.3 About the Iterative Refinement Toolkit (IRT)

The Iterative Refinement Toolkit (IRT) is a library of factorization routines, solvers, and tools that can be used to solve systems of linear equations more efficiently than the full-precision solvers in Cray XT-LibSci or ACML.

IRT exploits the fact that single-precision solvers can be up to twice as fast as double-precision solvers. IRT uses an iterative refinement process to obtain solutions accurate to double precision.

IRT provides two interfaces:

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

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

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

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

```
call irt_factorization-method_data-type_processing-mode(arguments)
```

such as: `call irt_po_real_parallel(arguments)`.

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

### 3.2.4 About the SuperLU Library

The SuperLU library routines solve large, sparse nonsymmetric systems of linear equations. Cray XT-LibSci SuperLU provides 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.

### 3.2.5 About the CRay Adaptive Fast Fourier Transform (CRAFFT) Library

CRAFFT is a library of Fortran subroutines that compute the discrete Fourier transform in one, two, or three dimensions; of arbitrary input size; and of both real and complex data. CRAFFT provides a simplified interface to FFT and allows the FFT library itself to choose the fastest FFT kernel.

To use the CRAFFT library:

- Load the `xt-libsci` module.
- Add the Fortran `use crafft` statement to any source file that calls a CRAFFT routine.
- Call `crafft_init()` before any other CRAFFT routine. This sets up the CRAFFT library for run time use.

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

See the `intro_crafft(3s)` man page for more information about the CRAFFT library. See the `crafft_set_planner(3s)` and `crafft_get_planner(3s)` man pages for more information on the planning of FFT kernels.

### 3.3 About the PETSc Library

The Programming Environment supports the Portable, Extensible Toolkit for Scientific Computation (PETSc). PETSc is an open source library of routines that solve partial differential equations. The toolkit includes a comprehensive suite of sparse iterative solvers and preconditioners. PETSc is available from Argonne National Lab; Cray's implementation differs from the public version only in performance.

The Cray implementation of PETSc is configured to use these external packages (automatically linked through the PETSc modules):

**MUMPS** MUMPS (MULTifrontal Massively Parallel sparse direct Solver) is a package of parallel, sparse, direct linear-system solvers based on a multifrontal algorithm. For further information, see <http://graal.ens-lyon.fr/MUMPS/>.

**SuperLU** SuperLU is a sequential version of SuperLU\_dist (not included with `petsc-complex`). For further information, see <http://crd.lbl.gov/~xiaoye/SuperLU/>.

**SuperLU\_dist**

SuperLU\_dist is a package of parallel, sparse, direct linear-system solvers (available in Cray LibSci). For further information, see <http://crd.lbl.gov/~xiaoye/SuperLU/>.

**ParMETIS** ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering) is a library of routines that partition unstructured graphs and meshes and compute fill-reducing orderings of sparse matrices. For further information, see <http://glaros.dtc.umn.edu/gkhome/views/metis/>.

**HYPRE** HYPRE is a library of high-performance preconditioners that use parallel multigrid methods for both structured and unstructured grid problems (not included with `petsc-complex`). For further information, see [http://www.llnl.gov/CASC/linear\\_solvers/](http://www.llnl.gov/CASC/linear_solvers/).

PETSc uses standard MPI functions for all message-passing communication.

Before compiling programs that use PETSc calls, load the appropriate PETSc module:

- `petsc` for real data
- `petsc-complex` for complex data

By loading the PETSc module, all header and library locations are automatically set corresponding to your environment. This removes the burden of managing `bmake` materials used in conventional PETSc processing.

For a PETSC example, see [Section 14.6, page 123](#). This example uses `makefile.F`. The corresponding `makefile.c` and the makefiles for conventional PETSc processing (`makefile_conventional.c` and `makefile_conventional.F`) are listed in [Appendix G, page 215](#).

For further information, see  
<http://www-unix.mcs.anl.gov/petsc/petsc-as/index.html>.

### 3.4 About the AMD Core Math Library (ACML)

The AMD Core Math Library (ACML) module is no longer loaded as part of the default PrgEnv environment. BLAS and LAPACK functionality is now provided by Cray XT-LibSci (see [Section 3.2.1, page 13](#)). However, if you need ACML for FFT functions, math functions, or random number generators, you can load the library using the `acml` module:

```
% module load acml
```

ACML includes:

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

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

### 3.5 About the FFTW Libraries

The Programming Environment includes versions 3.1.1 and 2.1.5 of the Fastest Fourier Transform in the West (FFTW) library. FFTW is a C subroutine library with Fortran interfaces for computing the discrete Fourier transform in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, such as the discrete cosine/sine transforms). The Fast Fourier Transform algorithm is applied for many problem sizes.

To use the default FFTW library, use:

```
% module load fftw
```

To use the FFTW 3.1.1 library, use:

```
% module load fftw/3.1.1
```

To use the FFTW 2.1.5.1 library, use:

```
% module load fftw/2.1.5.1
```

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

The FFTW 3.1.1 and FFTW 2.1.5.1 modules cannot be loaded at the same time. You must first unload the other module, if already loaded, before loading the desired one. For example, if you have loaded the FFTW 3.1.1 library and want to use FFTW 2.1.5.1 instead, use:

```
% module swap fftw/3.1.1 fftw/2.1.5.1
```

## 3.6 About the Fast\_mv Library

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

To use Fast\_mv functions, load the `libfast` module:

```
% module load libfast
```

Currently, the library contains:

- `exp()`, the 64-bit exponential function (the constant  $e$  raised to a power)
- `expf()`, the 32-bit exponential function (the constant  $e$  raised to a power)
- `frda_exp()`, the 64-bit exponential function for all elements in an array
- `frsa_expf()`, the 32-bit exponential function for all elements in an array

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

## 3.7 About the Cray MPT Library

The Cray MPT (Message Passing Toolkit) library includes MPICH2 and SHMEM functions.

### 3.7.1 About the MPICH2 Library

MPICH2 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 libraries are implemented on top of the Portals low-level message-passing engine. The Portals interface is transparent to the application programmer.

All Cray XT compilers support MPICH2 applications. There are two versions of the MPICH2 library available for users of the PGI or PathScale Fortran compilers. One version supports applications where the data size for the Fortran default types integer, real, and logical is 32 bits, and the other version supports applications where the data size is 64 bits. For further details, see [Section 4.1.1.1, page 27](#) and [Section 4.1.3, page 29](#).

For examples showing how to compile, link, and run MPI applications, see [Section 14.2, page 116](#) and [Section 15.2, page 158](#).

**Note:** Programs that use MPI library routines for parallel control and communication should call the `MPI_Finalize()` routine at the conclusion of the program.

For a list of MPI error messages and suggested workarounds, see [Appendix D, page 207](#).

For information about MPI environment variables, see the `intro_mpi(3)` man page.

There are some limitations to Cray XT MPICH2 you should take into consideration:

- 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 the `#include "mpi.h"` statement. Alternatively, if the application requires the `stdio.h` naming,

your application should have the `#include "mpi.h"` statement before the `#include <stdio.h>` or `#include <iostream>` statement.

- These process-creation functions are not supported and, if used, generate aborts at run time:
  - `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.

### 3.7.2 About the SHMEM Library

The Cray shared memory access (SHMEM) library is a set of logically shared, distributed memory access routines. Cray SHMEM routines are similar to MPI routines; they pass data between cooperating parallel processes. The Cray SHMEM library is implemented on top of the Portals low-level message-passing engine. The Portals interface is transparent to the application programmer.

All Cray XT compilers support SHMEM applications. There are two versions of the SHMEM library available for users of the PGI or PathScale Fortran compilers. One version supports applications where the data size for the Fortran default types integer, real, and logical is 32 bits; the other version supports applications where the size is 64 bits. For further details, see [Section 4.1.1.1, page 27](#) and [Section 4.1.3, page 29](#).

Cray SHMEM routines can be used in programs that perform computations in separate address spaces and that explicitly pass data by means of put and get functions to and from different processing elements in the program. Cray SHMEM routines can be called from Fortran, C, and C++ programs and used either by themselves or with MPI functions.

Portals and the Cray SHMEM library support these SHMEM atomic memory operations:

- atomic swap
- atomic conditional swap
- atomic fetch and increment
- atomic fetch and add
- atomic lock

An operation is defined as atomic if the steps cannot be interrupted and are done as a unit.

When running on CNL, you can use the environment variable `XT_LINUX_SHMEM_HEAP_SIZE` to set the size (in bytes) of the private heap. To set the stack size, use the `XT_LINUX_SHMEM_STACK_SIZE` environment variable. The size of the stack is limited by the value of `stacksize`, unless `stacksize` is set to `unlimited`, in which case the default size of the stack is 16 MB.

When running on Catamount, you can use the `yod` command line options `-stack`, `-heap`, and `-shmem` to set the size (in bytes) of the stack, private heap, and symmetric heap, respectively. See the `yod(1)` man page for details. On Catamount, SHMEM applications can use all available memory per node (total memory minus memory for the kernel and the process control thread (PCT)). SHMEM does not impose any restrictions on stack, heap, or symmetric heap memory regions.

You can use the `XT_SYMMETRIC_HEAP_SIZE` environment variable on either Catamount or CNL to control the size of the symmetric heap.

**Note:** To build, compile, and run Cray SHMEM applications, you need to call `start_pes(int npes)` or `shmem_init()` as the first Cray SHMEM call and `shmem_finalize()` as the last Cray SHMEM call.

If you use both MPI and SHMEM functions in an application, the sequence of required calls is:

```
MPI_Init
start_pes
...
shmem_finalize
MPI_Finalize
```

For examples showing how to compile, link, and run SHMEM applications on CNL, see [Section 14.3, page 118](#) and [Section 14.4, page 120](#). For Catamount examples, see [Section 15.3, page 160](#) and [Section 15.4, page 162](#).

When using SHMEM functions, be aware of these issues:

- The performance of strided operations is poor. The Portals network protocol stack on Cray XT is optimized for block transfers. It does not support efficient access of non-contiguous remote memory. Repackaging data into contiguous blocks in your application and then calling a `shmem_put()` or `shmem_get()` function give better performance than calling strided operations.
- Atomic memory operations are implemented in software and are relatively slow. You should not use these operations for high fan-in synchronization because the injection rate is much larger than the processing rate, leading to a buildup of requests and degraded performance.
- Barrier functions are implemented in software and are relatively slow. Cray recommends that you minimize the use of barriers.
- Avoid this type of construct:

```
while (remval != 0) {
 shmem_get64(&remval, &rem_flag, 1, pe);
}
```

It can severely tax the Portals network protocol stack, particularly if many processes are "spinning" on a variable at a single target process. If possible, use other synchronization mechanisms that rely on spinning on local memory.

### 3.8 About the OpenMP Library

The Cray XT system supports version 2.5 of the OpenMP standard. OpenMP is a shared-memory, parallel programming model that allows you to use threads. OpenMP provides library routines, Fortran directives, C and C++ pragmas, and environment variables. The PGI, GCC, and PathScale compilers support OpenMP.

To use OpenMP, use these compiler command options:

|           |            |
|-----------|------------|
| PGI       | -mp=nonuma |
| PathScale | -mp        |
| GCC       | -fopenmp   |

You also need to set the `OMP_NUM_THREADS` environment variable to the number of threads in the team.

The number of CPUs hosting OpenMP threads at any given time is fixed at program startup and specified by the `aprun -d depth` option (see [Section 7.1, page 61](#) for further information).

For an example showing how to compile, link, and run OpenMP applications, see [Section 14.7, page 135](#).

You can use OpenMP applications in hybrid OpenMP/MPI applications, but OpenMP applications cannot cross node boundaries. In OpenMP/MPI applications, MPI calls can be made from master or sequential regions but not parallel regions. OpenMP is supported on CNL but not Catamount.

For further information about running OpenMP applications, see the `aprun(1)` man page. For further information about OpenMP functions, see the OpenMP website (<http://www.openmp.org>), the PGI website (<http://www.pgroup.com/>), the PathScale website (<http://www.pathscale.com/>), or the GNU OpenMP website (<http://gcc.gnu.org/projects/gomp/>).

### 3.9 About the UPC Functions

The Cray XT system supports Unified Parallel C (UPC), a C language extension for parallel program development. You use UPC language syntax to read and write memory of other processes with simple assignment statements. Program synchronization occurs only when explicitly programmed; there is no implied synchronization.

Cray XT-UPC contains these front ends:

- Berkeley UPC translator, a UPC-to-C translator based on Open64.
- Intrepid GCCUPC, a UPC-to-assembly compiler based on GNU GCC.

Both front ends generate code that is linked with the Berkeley UPC run time library and communication system. These components comply with the UPC Language Specification, v1.2.

For examples showing you how to compile and run UPC programs, see [Section 14.5, page 122](#). and [Section 15.5, page 164](#).

For more information about UPC, see the `upcc(1)` and `upcrun(1)` man pages, the Berkeley UPC website (<http://upc.lbl.gov/docs/>), and the Intrepid UPC website ([http://www.intrepid.com/upc/cray\\_xt3\\_upc.html](http://www.intrepid.com/upc/cray_xt3_upc.html)).

# Cray XT Programming Considerations [4]

---

The manuals and man pages for third-party and open source Cray XT Programming Environment products provide platform-independent descriptions of product features. This chapter provides information specific to Cray XT systems that you should consider when using those products to develop CNL or Catamount applications.

## 4.1 Programming Considerations for all Developers

This section describes programming considerations that apply to CNL and Catamount applications.

### 4.1.1 About PGI Compilers

When using the PGI compilers, be aware of the following factors.

#### 4.1.1.1 About Default MPICH2 and SHMEM Libraries

If you use the PGI Fortran compiler, you can promote default integer, real, and logical operations to 64-bit precision. Include the `-default64` option on the `ftn` command line; this passes the `-i8` and `-r8` options to the compiler. The `-i8` option directs the compiler to use 64 bits for the data size of default integer and logical operations. The `-r8` option directs the compiler to use 64 bits for the data size of default real variables.

Use this method to compile all Fortran source files that contain default integer, logical, real, or complex variables.

The `-default64` option directs the linker to use the `default64` version of the MPI or SHMEM library. If you compile using `-default64` but omit the `-default64` option when linking, the linker attempts to use the `default32` libraries; the executable probably will not run.

**Note:** The `-default64` option does not affect the sizes of data types that use explicit kind and star values.

#### 4.1.1.2 About Unsupported C++ Header Files

PGI does not provide a complete set of the old C++ Standard Library and STL header files. However, PGI C++ does support some old header files (`iostream.h`, `exception.h`, `iomanip.h`, `ios.h`, `istream.h`, `ostream.h`, `new.h`, `streambuf.h`, `strstream.h`, and `typeinfo.h`), which include their C++ Standard Library counterpart.

To use an unsupported header file:

- Delete the `.h`. For example, change `<vector.h>` to `<vector>`, **or**
- Create your own `headerfile.h` file and use the `-I` compiler option to direct the compiler to access the header file in your directory:

```
#ifndef __VECTOR_H
#define __VECTOR_H
#include <vector>
using std::vector;
#endif
```

#### 4.1.1.3 About Restrictions on Large Data Objects

PGI compilers support data objects larger than 2 GB. However, the Cray XT Programming Environment has restrictions because the user-level libraries (MPI, SHMEM, and LibSci) are compiled in the small memory model.

The only way to build an application with data objects larger than 2 GB is to limit the static data sections to less than 2 GB by converting static data to dynamically allocated data.

#### 4.1.1.4 About the FORTRAN STOP Message

The PGI Fortran `stop` statement writes a FORTRAN STOP message to standard output. In a parallel application, every process that executes the `stop` statement writes this message. This is not scalable and will cause performance and, potentially, reliability problems in very large applications.

To turn off the STOP message, use the `NO_STOP_MESSAGE` environment variable.

#### 4.1.1.5 Suppressing PGI Vectorization

To suppress vectorization in PGI applications, use:

- The `-Mnovect` compiler option, which suppresses vectorization for the entire source file.
- The `!pgi$r novector` directive or `#pragma routine novector` statement placed before the start of a routine, which suppresses vectorization for the entire routine.
- The `!pgi$ novector` directive or `#pragma loop novector` statement placed before a loop, which suppresses vectorization for the loop. In most cases, use the directive on innermost loops because the directive does not suppress vectorization for loops nested inside the targeted loop.

For more information, see the *PGI User's Guide*.

#### 4.1.2 About the PGI Debugger

The PGI debugger, PGDBG, is not supported on Cray XT systems.

#### 4.1.3 About the PathScale Fortran Compiler

If you use PathScale Fortran, you can promote default integer, real, and logical operations to 64-bit precision. By including the `-default64` option on the `ftn` command line, you pass the `-i8` and `-r8` options to the compiler. The `-i8` option directs the compiler to use 64 bits for the data size of default integer and logical operations. The `-r8` option directs the compiler to use 64 bits for the data size of default real variables. Compile all Fortran source files containing default integer, logical, real, or complex variables this way.

Also, for MPI applications the `-default64` option directs the linker to use the `default64` version of the MPI or SHMEM library.

Link in `default64` mode. If you compile with the `-default64` option but link without the `-default64` option, the compiler tries to link to the `default32` libraries; the resulting executable probably will not run.

**Note:** The sizes of data types that use explicit kind and star values are not affected by this option.

#### 4.1.4 About Little-endian Support

The Cray XT system supports little-endian byte ordering. The least significant value in a sequence of bytes is stored first in memory.

#### 4.1.5 About the Portals Message Size Limit

A single Portals message cannot be longer than 2 GB.

#### 4.1.6 About Shared Libraries

The Cray XT systems currently do not support dynamic loading of executable code or shared libraries. Also, the related `LD_PRELOAD` environment variable is not supported.

### 4.2 Programming Considerations for CNL Users

This section describes the factors to consider when developing CNL applications.

#### 4.2.1 About CNL glibc Functions

CNL provides limited support of the glibc process control functions, such as `popen()`, `fork()`, `exec()`, and `system()`. The resulting processes execute in the limited RAM disk environment on each compute node.

The `exec()` function can execute the `scp` and `ksh` commands and these BusyBox commands:

---

|                     |                     |                      |
|---------------------|---------------------|----------------------|
| <code>ash</code>    | <code>cat</code>    | <code>chmod</code>   |
| <code>chown</code>  | <code>cp</code>     | <code>cpio</code>    |
| <code>dmesg</code>  | <code>free</code>   | <code>grep</code>    |
| <code>gunzip</code> | <code>kill</code>   | <code>killall</code> |
| <code>ksh</code>    | <code>ln</code>     | <code>logger</code>  |
| <code>ls</code>     | <code>mkdir</code>  | <code>mktemp</code>  |
| <code>more</code>   | <code>nice</code>   | <code>ping</code>    |
| <code>ps</code>     | <code>renice</code> | <code>rm</code>      |
| <code>sleep</code>  | <code>tail</code>   | <code>test</code>    |
| <code>vi</code>     | <code>zcat</code>   |                      |

---

For more information, see [Section 14.14, page 147](#) and the `busybox(1)` man page.

You can access the `cpuinfo` and `meminfo` `/proc` files. These files contain information about your compute node.

CNL also supports the `ttyname()` function. See [Section 4.2.2, page 31](#) for information about how `aprun` handles `stdin`, `stdout`, and `stderr`.

CNL `glibc` does not support:

- The `getgrgid()`, `getgrnam()`, `gethostbyname()`, `getpwnam()`, or `getpwuid()` functions.
- Customer-provided functions that require a daemon.

[Appendix A, page 187](#) lists the `glibc` functions that CNL supports.

#### 4.2.2 About I/O Support Operations under CNL

I/O operations allowed in CNL applications are Fortran, C, and C++ I/O calls; Cray MPICH2, Cray SHMEM, and OpenMP I/O functions; and the underlying Linux Lustre client I/O functions.

I/O to Lustre is supported in CNL. Files in other remote file systems cannot be accessed. One exception is the handling of `stdin`, `stdout`, and `stderr`.

The `aprun` utility handles `stdin`, `stdout`, and `stderr`. The `aprun` file descriptor 0 forwards `stdin` data to processing element 0 (PE 0) only; `stdin` is closed on all other PEs. The `stdout` and `stderr` data from all PEs is sent to `aprun`, which forwards the data to file descriptors 1 and 2.

Files local to the compute node, such as ones in `/proc` or `/tmp`, can be accessed by an application. See [Section 4.2.1, page 30](#).

In Catamount, I/O is possible to any file system accessible to `yod`. Lustre I/O is handled as a special case.

### 4.2.3 Connecting to External Services under CNL

Cray XT systems support external connectivity to or from compute nodes running CNL. You can use IP functions in your programs to access network services. To determine if your site has configured CNL compute nodes for network connectivity, see your system administrator.

### 4.2.4 About Timing Functions under CNL

CNL supports these timing functions:

- CPU timers. The Fortran `cpu_time(time)` intrinsic subroutine returns the processor time; `time` 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 amount of CPU time (in seconds) used by the program.
- Elapsed time counter. CNL supports the `MPI_Wtime()` and `MPI_Wtick()` functions and the Fortran `system_clock()` intrinsic subroutine.

The `MPI_Wtime()` function returns the elapsed time. The `MPI_Wtick()` function returns the resolution of `MPI_Wtime()` in seconds.

CNL does not support the `dclock()` or `etime()` functions.

### 4.2.5 About Signal Support under CNL

The `aprun` utility catches `SIG*` signals and forwards them to applications. For more information, see [Section 7.8, page 75](#).

#### 4.2.6 Killing Processes under CNL

To kill CNL applications, you can use the `apkill` command. Also, the Linux `kill` command and the `kill()` system call are supported on CNL compute nodes. Each process on a node can send signals to itself and other processes on that node.

Use the `xtkill` command to kill processes on service nodes.

#### 4.2.7 About Core Files under CNL

When an application fails on CNL, one core file is generated for the first failing process. An application generates no core file at all if a file named `core` already exists in the current directory.

#### 4.2.8 Using Cray XT4 Quad-core Processors

Cray XT systems support single-socket, quad-core compute nodes (Cray XT4). Cray XT4 compute nodes run CNL.

To run an application exclusively on Cray XT4 compute nodes:

1. Use the `cnselect` command to get a list of all Cray XT4 compute nodes.

```
% cnselect coremask.eq.15
70-79
```

2. Load the `xtpc-quadcore` module; this module adds the appropriate quad-core option on the compiler command line:

- PGI: `-tp barcelona-64`
- GCC: `-march=barcelona`
- PathScale: `-march=barcelona`

where `barcelona` is the code name of the Quad-Core AMD Opteron processor.

3. For interactive jobs, include all or a subset of the candidate list on the `aprun -L` option.

```
% aprun -n 16 -L 70-75 ./a.out
```

For batch and interactive batch jobs, include all or a subset of the candidate list on the `qsub -lmpnodes` option:

```
% qsub -lmpwidth=16 -lmpnodes="/70-75/"
```

#### 4.2.9 Using Cray XT5 Compute Nodes

Cray XT systems support Cray XT5 compute nodes, which run CNL. Each Cray XT5 compute node consists of two sockets, with each socket housing a NUMA node. NUMA node 0 has a quad-core processor (logical CPUs 0-3), memory, and connections to the SeaStar interconnection network. NUMA node 1 has a quad-core processor (logical CPUs 4-7) and memory.

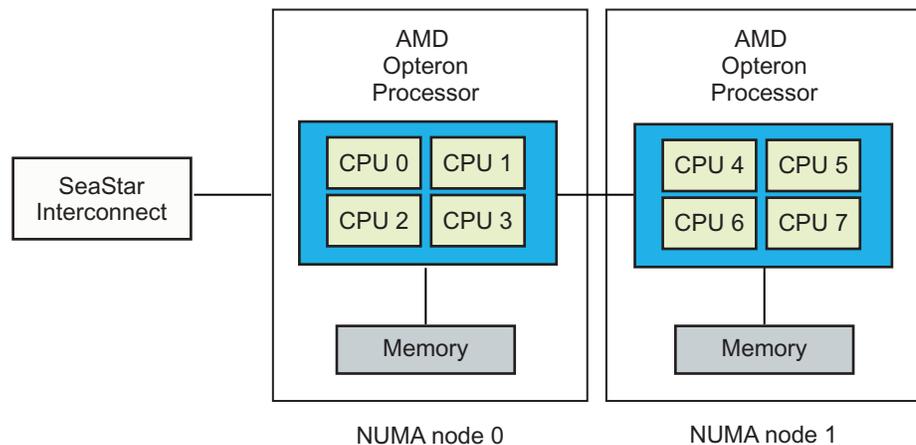


Figure 1. Cray XT5 Compute Node

**Note:** Having a Cray XT5 compute node reserved for your job does not guarantee that you can use both NUMA nodes. You have to request sufficient resources through `aprun` placement options (`-n`, `-N`, `-d`, `-m`, or default values) to be able to use both NUMA nodes. If you do not have access to both NUMA nodes, `aprun` memory affinity and CPU affinity options that reference the second NUMA node are either ignored or your job is terminated. For examples showing how to use memory affinity options, see [Section 14.12, page 144](#). For examples showing how to use CPU affinity options, see [Section 14.13, page 146](#).

To run an application exclusively on Cray XT5 compute nodes:

1. Use the `cnselect` command to get a candidate list.

```
% cnselect coremask.eq.255
24-95,128-223,256-351,384-447
```

2. Load the `xtpc-quadcore` module; this module adds the appropriate quad-core option on the compiler command line:

- PGI: `-tp barcelona-64`
- GCC: `-march=barcelona`
- PathScale: `-march=barcelona`

3. For interactive jobs, include all or a subset of the candidate list on the `aprun -L` option.

```
% aprun -n 16 -L 384-447 ./a.out
```

For batch and interactive batch jobs, include all or a subset of the candidate list on the `qsub -lmpnodes` option:

```
% qsub -lmpwidth=16 -lmpnodes="/384-447/"
```

#### 4.2.10 Using Huge Pages and Base Pages under CNL

Cray XT systems support 2 MB huge pages and 4 KB base pages for CNL applications. Previous versions of CNL supported only base pages. For applications that use a large amount of virtual memory, 4 KB pages can put a heavy load on the virtual memory subsystem. Huge pages can provide a significant performance increase for such applications.

The 4 KB base pages remain the default. To specify huge pages:

1. Load the huge pages library, `hugetlbfs.a`, during the linking phase. For example, you could use the following commands:

```
% cc -c my_hugepages_app.c
% cc -o my_hugepages_app my_hugepages_app.o -lhugetlbfs
```

**Note:** The `-lhugetlbfs` argument, as shown, is required. Do not use the separated form, `-l hugetlbfs`.

2. Set the huge pages environment variable.

```
% setenv HUGETLB_MORECORE yes
```

or

```
$ export HUGETLB_MORECORE=yes
```

If you do not set this environment variable or if you set it to `no`, CNL uses 4 KB pages.

3. Add a huge pages suffix to the `aprun -m size` option.

`-m sizeh`                   **Requests** *size* of huge pages to be allocated to each processing element. All nodes use as much huge page memory as they are able to allocate and afterward use 4 KB pages.

`-m sizehs`                   **Requires** *size* of huge pages to be allocated to each processing element. If the request cannot be satisfied, an error message is issued and `aprun` terminates the request.

This command requests 700 MB of huge pages per processing element (PE), or 1400 MB per node on dual-core nodes:

```
% aprun -m700h -n 2 -N 2 ./my_hugepages_app
```

You can run base-page and huge-page applications on the same machine at the same time. The applications can run in any order in succession on any groups of nodes.

For example, you could run a base-page application, a huge-page application, and then a base-page application:

```
% aprun -n 64 -N 2 ./my_4kbpage_app
% setenv HUGETLB_MORECORE yes
% aprun -m700h -n 64 -N 2 ./my_hugepages_app
% aprun -n 64 -N 2 ./my_4kbpage_app
```

Only the heap is placed on huge pages. All other program segments (code, initialized data, BSS data, and the stack) are on 4 KB pages. The heap is not placed on huge pages if the application uses an allocation function other than `glibc malloc()` or overrides the `glibc malloc morecore()` function.

The memory available for huge pages is less than the total memory on the node. You must leave enough memory for CNL and I/O buffers. Also, because of memory fragmentation, less memory is available for huge pages after a node has run other jobs.

There is no guaranteed amount of huge page memory available to an application. Cray recommends that you not request more than these values for huge pages:

| Total Memory on the Node | Memory Available for Huge Pages |
|--------------------------|---------------------------------|
| 2 GB                     | 1000 MB per node                |
| 4 GB                     | 3000 MB per node                |
| 8 GB                     | 6400 MB per node                |
| 16 GB                    | 14100 MB per node               |
| 32 GB                    | 30000 MB per node               |

Memory allocated as huge pages is unavailable for I/O, whether the application uses the memory or not. Less available memory for I/O buffers may result in performance degradation.

If you do not include the `-msizehs` option and not enough huge pages are available, run times may be inconsistent.

**Note:** The Linux `alloc_hugepages()` and `free_hugepages()` system calls are no longer supported. CNL uses the `hugetlbfs` file system instead; `hugetlbfs` is a pseudo file system used for mapping huge pages into an application's virtual address space.

#### 4.2.11 Allocating Memory under CNL

CNL provides environment variables to control how the system memory allocation routine `malloc()` behaves. The environment variables are:

- `MALLOC_MMAP_MAX_`
- `MALLOC_TRIM_THRESHOLD_`

Note the trailing underscores on the environment variable names.

#### `MALLOC_MMAP_MAX_`

Using memory-mapped (MMAP) regions is very costly compared to using the heap. MMAP regions exist to allow a program to return unused memory to the system more easily so it may be used by other processes on the node. This is most applicable and helpful on SMP nodes with multiple programs sharing a node. Because Cray XT systems do not run multiple applications on the same compute node, using MMAP regions is generally not helpful.

By default, `MALLOC_MMAP_MAX_` is 64, meaning that your program may have as many as 64 non-heap, memory-mapped regions. Cray recommends that you set `MALLOC_MMAP_MAX_` to 0. This means that your program will not use any MMAP regions, eliminating `mmap()` and `munmap()` system calls.

There may be cases where it is inappropriate to set the variable to zero. MMAP regions could help if an MPI program has one process on a node that uses a lot of memory and frees it, and later another process on the same node uses a lot of memory. However, memory usage is typically not asymmetric in this way. Also, an application that does a significant amount of I/O near the end of its run, after freeing a lot of memory, may also benefit from using MMAP regions. However, experimental data has not shown that this potential benefit outweighs the extra cost of `mmap()` and `munmap()` calls during the life of the program.

#### `MALLOC_TRIM_THRESHOLD_`

`MALLOC_TRIM_THRESHOLD_` is the amount of free space at the top of the heap that needs to exist before `malloc()` will return the memory to CNL. Returning memory to CNL is costly. For example, the default setting of 128 KB is much too low for a node with 4 GB of memory and one application running. Cray suggest setting `MALLOC_TRIM_THRESHOLD_` to 536870912 (that is, 0.5 GB). This reduces the number of `sbrk()` and `brk()` calls, improving performance.

If your application uses less than 0.5 GB of memory, set the variable to the amount your application uses. Also, if your application needs a lot of memory for I/O buffering and setting this variable to a smaller value would free some memory, you may want to set it lower than the recommended 0.5 GB.

**Note:** If you use PGI compilers, you have an alternative to using these environment variables. The PGI `-msmartialloc` option is the equivalent of setting `MALLOC_MMAP_MAX_` to 2 and `MALLOC_TRIM_THRESHOLD_` to 1 GB. The trade-offs are that `-msmartialloc` cannot be used with GCC or PathScale compilers, you need to recompile your application, and the `-msmartialloc` settings may not be optimal values.

#### 4.2.12 About Resource Limits under CNL

Memory limits are defined by the node default or the `aprun -m` option. Time limits are inherited from the `aprun` process limits or specified with the `aprun -t` option. Other limits are inherited from the limits of `aprun`. All limits apply to individual processing elements; there are no aggregate application limits that can be specified with `aprun` options.

#### 4.2.13 About the One Application Per Node Limitation under CNL

Cray XT systems currently do not support running more than one CNL application on a compute node.

#### 4.2.14 About Parallel Programming Models under CNL

The MPI, SHMEM, OpenMP, and UPC parallel programming models are supported on CNL applications.

#### 4.2.15 About the Modified Copy-on-write Process under CNL

Under Linux, `fork()` uses a copy-on-write process to conserve time and memory resources. When a process forks a child process, most of the pages in the parent process' address space are initially shared with the child process. The parent and child processes can continue sharing a page until one of the processes tries to modify the page. At that point, the process modifying the page creates a new page for its private use, copies the previously-shared page's data into it, and continues to use this new page instead of the previously-shared page. The previously-shared page now belongs solely to the other process.

The copy-on-write process can adversely affect Cray XT user applications that use Portals. To correct this problem, Cray modified the Portals kernel to perform a partial copy when a process forks a child process. For each region of a process' address space that is registered with Portals for Remote Direct Memory Access (RDMA), the first and last page of the region are copied to a private page in the child's address space as the fork occurs. This ensures that Portals can continue to transfer data using these pages in the parent's address space, and also ensures that any data residing on these pages that were not intended for Portals transfers (such as heap variables) can be referenced in the child's address space.

The implications are:

- Pages in the middle of a Portals memory region (likely maps to any large MPI message buffers) are not accessible in the child process. Copy the necessary data out of the parent's message buffer before forking.
- More memory is allocated and copied than in a normal fork. This could cause unexpected memory exhaustion if you have many Portals memory regions.

#### 4.2.16 About Unsupported PGI Compiler Command Options under CNL

These PGI compiler command options are not supported on CNL applications:

- `-mprof=mpi`
- `-Mmpi`
- `-Mscalapack`

**Note:** Not all PGI options have been tested under CNL.

### 4.3 Programming Considerations for Catamount Users

This section describes the factors to consider when developing Catamount applications.

#### 4.3.1 About Catamount glibc Functions

Because Catamount is designed to emphasize 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:

- Dynamic process control (such as `exec()`, `popen()`, `fork()`, or system library calls).
- 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()`, you may have to resolve data alignment issues. See the `malloc()` man page for details.

**Note:** The Cray XT system provides two implementations of `malloc()`: Catamount `malloc()` and GNU `malloc()`. Catamount provides a custom implementation of the `malloc()` function. This implementation is tuned to Catamount's non-virtual-memory operating system and favors applications that allocate large, contiguous data arrays. The function uses a first-fit, last-in-first-out (LIFO) linked list algorithm. For information about gathering statistics on memory usage, see the `heap_info(3)` man page. In some cases, GNU `malloc()` may improve performance.

- The `profil()` function.
- Any of the `getpwd*()`, `getgr*()`, and `getpw*()` families of library calls.
- Terminal control.
- Customer-provided functions that require a daemon.
- Any functions that require a database, such as Network Block Device (NDB) functions. For example, there is no support for the `uid` and `gid` family of queries that are based on the NDB functions.
- There is limited support for signals and `ioctl()`. See the `ioctl(2)` man page for details.

[Appendix B, page 193](#) lists the glibc functions that Catamount supports. The glibc functions that Catamount does not support are so noted in their man pages.

### 4.3.2 About I/O Support Functions under Catamount

I/O support for Catamount 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) I/O functions.

Keep in mind these behaviors:

- I/O is offloaded to the service I/O nodes. The `yod` application launcher handles `stdin`, `stderr`, and `stdout`. For more information, see [Section 8.6, page 82](#).
- Calling an I/O function such as `open()` with a bad address causes the application to fail with a page fault. On the service nodes, a bad address causes the function to set `errno = EFAULT` and return `-1`.
- Catamount does not support I/O on named pipes.

The following sections describe techniques you can use to improve I/O performance.

#### 4.3.2.1 Improving Fortran I/O Performance under Catamount

To increase buffer size in a Fortran program, use the `setvbuf3f()` function:

```
integer function setvbuf3f(lu, type, size)
```

where:

`integer lu` The logical unit

`integer type`

- 0 — Full buffering
- 1 — Line buffering
- 2 — No buffering

`integer size`

The size of the new buffer

The `setvbuf3f()` function returns 0 on success, nonzero on failure. For more information, see the `setbuf(3)` man page.

#### 4.3.2.2 Improving C++ I/O Performance under Catamount

The standard stream I/O facilities defined in the Standard C++ header file `<iostream>` are unbuffered. You can use the routine `pubsetbuf()` to specify a buffer for I/O. [Section 15.7, page 166](#) shows how `pubsetbuf()` can improve performance.

I/O-to-file streams defined in `<fstream>` are buffered with a default buffer size of 4096. You can use `pubsetbuf()` to specify a buffer of a different size. Specify the buffer size before the program performs a read or write to the file; otherwise, the call to `pubsetbuf()` is ignored and the default buffer is used. [Section 15.7, page 166](#) shows how to use `pubsetbuf()` to specify a buffer for `<fstream>` file I/O. Avoid calls to member function `endl` to prevent the buffer from being flushed.

#### 4.3.2.3 Improving `stdio` Performance under Catamount

By default, `stdin`, `stdout`, and `stderr` are unbuffered. Under Catamount, this limits the data transfer rate to approximately 10 bytes per second because read and write calls are offloaded to `yod`. To improve performance, call `setvbuf()` to buffer `stdin` input or `stdout/stderr` output. For an example showing how to improve `stdio` performance, see [Section 15.9, page 169](#).

#### 4.3.2.4 Improving the Performance of Large File, Sequential I/O under Catamount

IOBUF is an I/O buffering library that can reduce the I/O wait time for programs that read or write large files sequentially. IOBUF intercepts standard I/O calls such as `fread()` and `fopen()` and replaces the `stdio` layer of buffering with a replacement layer of buffering, thus improving program performance by enabling asynchronous prefetching and caching of file data. In addition, IOBUF can gather run time statistics and print a summary report of I/O activity for each file.

No program source changes are needed to use IOBUF. Instead, relink your program with the IOBUF library and set one or more environment variables.

To use IOBUF:

1. Load the `iobuf` module.  

```
% module load iobuf
```
2. Relink the program.
3. Set the `IOBUF_PARAMS` environment variable.

The `IOBUF_PARAMS` environment variable specifies patterns for selecting I/O files and sets parameters for buffering. If you do not set this environment variable, the default state is no buffering and the I/O call is passed on to the next layer without intervention.

The general format of the `IOBUF_PARAMS` environment variable is a comma-separated list of specifications:

```
IOBUF_PARAMS 'spec1,spec2,spec3,...'
```

Each specification begins with a file name pattern. When a file is opened, the list of specifications is scanned and the first matching file name pattern is selected. If no pattern matches, the file is not buffered. The file name pattern follows standard shell pattern matching rules. For example, to buffer `stdout`, you would use:

```
% setenv IOBUF_PARAMS '%stdout'
```

or:

```
% export IOBUF_PARAMS='%stdout'
```

#### 4. Execute the program.

**Note:** IOBUF works with PGI Fortran programs but does not work with PathScale Fortran or GNU Fortran programs. Also, IOBUF works with the PGI, PathScale, and GNU C compilers. IOBUF works with C++ programs that use `stdio` but does not work with the C++ standard buffered I/O stream class `<iostream>`.

C programs that use POSIX-style I/O calls such as `open()`, `read()`, `write()`, and `close()` are not affected by IOBUF. A workaround is to replace POSIX I/O calls in the C program with their equivalent IOBUF-specific calls. The IOBUF calls are identical to their POSIX counterparts but are prefixed with `iobuf_`.

For further information, see the `iobuf(3)` man page.

#### 4.3.2.5 Using Stride I/O Functions to Improve Performance under Catamount

You can improve file I/O performance of C and C++ programs by using the `readx()`, `writex()`, `ireadx()`, and `iwritex()` stride I/O functions. For further information, see the man pages.

#### 4.3.2.6 Reducing Memory Fragmentation under Catamount

In past releases, small memory allocations could become interspersed throughout memory, preventing the allocation of very large arrays (that is, arrays larger than half of available memory). To solve this problem, small allocations (those less than or equal to 100 MB, by default) are still allocated into the beginning of the first available free area of memory, but large allocations are now allocated into the end of the last available free area. This allows very large arrays to be allocated/freed in a separate area of memory, making memory fragmentation less likely.

You can use the `CATMALLOC_LARGE_ALLOC_SIZE` environment variable to change the default small versus large delineation line.

#### 4.3.3 About the Limitations of External Connectivity under Catamount

Cray XT does not support external connectivity to or from compute nodes running Catamount. Pipes, sockets, remote procedure calls, or other types of TCP/IP communication are not supported. The parallel programming model functions and the underlying Portals interface are the only supported communication mechanisms.

#### 4.3.4 About Timing Functions under Catamount

Catamount supports these timing functions:

- Interval timer. Catamount supports the `setitimer ITIMER_REAL` function. It does not support the `setitimer ITIMER_VIRTUAL` or the `setitimer ITIMER_PROF` function. Also, Catamount does not support the `getitimer()` function.
- CPU timers. Catamount supports the glibc `getrusage()` and the Fortran `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(time)` intrinsic subroutine returns the processor time, where *time* has a data type of `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 amount of CPU time (in seconds) used by the program.
- Elapsed time counter. The `dclock()`, Catamount `clock()`, and `MPI_Wtime()` functions and the `system_clock()` Fortran intrinsic subroutine calculate elapsed time. The `etime()` function is not supported.

The `dclock()` value rolls over approximately every 14 years and has a nominal resolution 100 nanoseconds on each node.

**Note:** The `dclock()` function is based on the configured processor frequency, which may vary slightly from the actual frequency. The clock frequency is not calibrated. Furthermore, 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. For an example showing how to use `dclock()` to calculate elapsed time, see [Section 15.6, page 165](#).

The resolution of the `system_clock` subroutine depends on the compiler and kind of argument. The count rate can be obtained by specifying the `count_rate` argument to `system_clock`.

The `clock()` function is now supported on Catamount; it estimates elapsed time as defined for the `dclock()` function. The Catamount `clock()` function is not the same as the Linux `clock()` function. The Linux `clock()` function measures processor time used. For Catamount compute node applications, Cray recommends that you use the `dclock()` function or an intrinsic timing routine in Fortran such as `cpu_time()` instead of `clock()`. For further information, see the `dclock(3)` and `clock(3)` man pages.

The `MPI_Wtime()` function returns the elapsed time. The `MPI_Wtick()` function returns the resolution of `MPI_Wtime()` in seconds.

#### 4.3.5 About Signal Support under Catamount

In previous Cray XT releases, Catamount did not correctly provide extra arguments to signal handlers when the user requested them through `sigaction()`. Signal handlers installed through `sigaction()` have the prototype:

```
void (*handler) (int, siginfo_t *, void *)
```

which allows a signal handler to optionally request two extra parameters. On Catamount compute nodes, these extra parameters are provided in a limited fashion when requested.

The `siginfo_t` pointer points to a valid structure of the correct size but contains no data.

The `void *` parameter points to a `ucontext_t` structure. The `uc_mcontext` field within that structure is a platform-specific data structure that, on compute nodes, is defined as a `sigcontext_t` structure. Within that structure, the general purpose and floating-point registers are provided to the user. You should rely on no other data.

For a description of how `yod` propagates signals to running applications, see [Section 8.7, page 82](#).

#### 4.3.6 Killing Processes under Catamount

The `kill()` system call is supported on Catamount compute nodes. A process can use it to send signals only to itself. If a process sends a signal to another process, `-1` is returned and `errno` is set to `ESRCH` (meaning the `pid` or process group does not exist). The Linux `kill` command is not supported on Catamount compute nodes.

Use the `xtkill` command to kill processes on service nodes.

#### 4.3.7 About Core Files under Catamount

By default, when an application fails on Catamount, only one core file is generated: that of the first failing process. For information about overriding the defaults, see the `core(5)` man page. Use caution with the overrides because dumping core files from all processes is not scalable.

Under certain conditions, the core file may be truncated without notifying you. These conditions are:

- The disk is full, or
- The core file exceeds the user limit. The user limit is set through the `limit coredumpsize size` command for `csh` and the `ulimit -c size` command for `bash`.

#### 4.3.8 Changing Page Size under Catamount

The `yod -small_pages` option allows you to specify 4 KB pages instead of the default 2 MB pages. Locality of memory references affects the optimum choice between the default 2 MB pages and the 4 KB pages. Because it is often difficult to determine how the compiler is allocating your data, the best approach is to try both the default and the `-small_pages` option and compare performance numbers.

**Note:** For each 1 GB of memory, 2 MB of page table space are required.

The Catamount `getpagesize()` function returns 4 KB.

#### 4.3.9 About Resource Limits under Catamount

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.

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.

#### 4.3.10 About the Limitations on Parallel Programming Models under Catamount

The MPI, SHMEM, and UPC parallel programming models are supported on Catamount applications. OpenMP is not supported on Catamount.

#### 4.3.11 About Unsupported PGI Compiler Command Options under Catamount

These PGI compiler command options are not supported on Catamount applications:

- `-mprof=mpi`
- `-Mmpi`
- `-Mscalapack`
- `-Mpfi`
- `-Mpfo`
- `-Mconcur`

Note: Not all PGI options have been tested under Catamount.

# Using Compilers [5]

---

The Programming Environment includes Fortran, C, and C++ compilers from PGI, GNU, and PathScale. You access the compilers through Cray XT compiler drivers, which perform the necessary initializations and load operations, such as linking in the header files and system libraries (`libc.a` and `libmpich.a`, for example) before invoking the compilers.

## 5.1 Setting Your Target Architecture

Before you begin to compile programs, you must verify that the target architecture is set correctly. The target architecture is used by the compilers and linker in creating executables to run on either CNL or Catamount compute nodes; it is set automatically when you log in. If the compute nodes are running CNL, the `xtpe-target-cnl` module is loaded and the `XTPE_COMPILE_TARGET` environment variable is set to `linux`. If the compute nodes are running Catamount, the `xtpe-target-catamount` module is loaded and `XTPE_COMPILE_TARGET` is set to `catamount`.

To determine the current target architecture, use the `module list` command. Either `xtpe-target-cnl` or `xtpe-target-catamount` will be loaded.

You cannot run a CNL application on compute nodes running Catamount nor a Catamount application on compute nodes running CNL. However, you can create CNL or Catamount executables at any time by configuring your environment properly.

For example, if the target architecture were `catamount` and you wanted to create an executable that would run later under CNL, you would swap `xtpe-target` modules:

```
% module swap xtpe-target-catamount xtpe-target-cnl
```

## 5.2 Using the Compiler Driver Commands

The syntax for the compiler driver commands is:

```
% [cc | CC | ftn] [PGI_options|GCC_options|PathScale_options]
files [-lhugetlbfs]
```

For example, to use the PGI Fortran compiler to compile `prog1_cn1.f90` and create default executable `a.out` to be run on CNL compute nodes, first use the `module list` command to verify that these modules have been loaded:

```
PrgEnv-pgi
xtpe-target-cn1
```

Then use this command:

```
% ftn prog1_cn1.f90
```

If you next want to use the PathScale C compiler to compile `prog2_qk.c` and create default executable `a.out` to be run on Catamount compute nodes, use these commands:

```
% module swap PrgEnv-pgi PrgEnv-pathscale
% module swap xtpe-target-cn1 xtpe-target-catamount
% cc prog2_qk.c
```

**Note:** Store your CNL and Catamount executables in separate directories or differentiate them by file name. If you try to run a CNL application when Catamount is running or a Catamount application when CNL is running, your application will abort.

If you want to suppress the output of the target architecture `INFO` message, use this environment variable:

```
XTPE_INFO_MESSAGE_OFF
```

The following example shows the effect of setting the `XTPE_INFO_MESSAGE_OFF` environment variable:

```
% ftn prog1.f90
ftn prog1.f90/opt/xt-asyncpe/1.0.0/bin/ftn: INFO: linux target is being used
% setenv XTPE_INFO_MESSAGE_OFF 1
% ftn prog1.f90
%
```

By default, the `INFO` message is displayed.

### 5.2.1 Using PGI Compilers

To use the PGI compilers, run the `module list` command to verify that the `PrgEnv-pgi` module is loaded. If it is not, use a `module swap` command, such as:

```
% module swap PrgEnv-gnu PrgEnv-pgi
```

`PrgEnv-pgi` loads the product modules that define the system paths and environment variables needed to use the PGI compilers.

For a description of new and modified PGI compiler features, see the *PGI Server 7.1 and Workstation 7.1 Installation Guide*.

**Note:** When linking in ACML routines, you must compile and link all program units with `-Mcache_align` or an aggregate option such as `fastsse`, which incorporates `-Mcache_align`.

[Table 2](#) lists the commands for invoking the PGI compilers and the source file extensions.

Table 2. PGI Compiler Commands

| Compiler               | Command          | Source File                                               |
|------------------------|------------------|-----------------------------------------------------------|
| C compiler             | <code>cc</code>  | <code>filename.c</code>                                   |
| C++ compiler           | <code>CC</code>  | <code>filename.CC</code>                                  |
|                        |                  | <code>filename.cc</code>                                  |
|                        |                  | <code>filename.cpp</code>                                 |
|                        |                  | <code>filename.cxx</code>                                 |
| Fortran 90/95 compiler | <code>ftn</code> | <code>filename.f</code> (fixed source, no preprocessing)  |
|                        |                  | <code>filename.f90</code> (free source, no preprocessing) |
|                        |                  | <code>filename.f95</code> (free source, no preprocessing) |
|                        |                  | <code>filename.F</code> (fixed source, preprocessing)     |
|                        |                  | <code>filename.F90</code> (free source, preprocessing)    |
|                        |                  | <code>filename.F95</code> (free source, preprocessing)    |



**Caution:** To invoke a PGI compiler, use the `cc`, the `CC`, or the `ftn` command. If you invoke a compiler directly using a `pgcc`, `pgCC`, or `pgf95` command, the resulting executable will not run on Cray XT compute nodes.

The `cc(1)`, `CC(1)`, and `ftn(1)` man pages contain information about the compiler driver commands, whereas the `pgcc(1)`, `pgCC(1)`, and `pgf95(1)` man pages contain descriptions of the PGI compiler command options.

The *PGI User's Guide* and the *PGI Fortran Reference* manual include information about compiler features unique to Cray (see <http://www.pgroup.com/resources/docs.htm>).

To verify that you are using the correct version of a compiler, use the `-V` option on a `cc`, `CC`, or `ftn` command.

**Note:** The `-Mconcur` (auto-concurrentization of loops) option documented in the PGI manuals is not supported on Cray XT systems.

## 5.2.2 Using GNU Compilers

To use the GNU compilers, run the `module list` command to verify that the `PrgEnv-gnu` module is loaded. If it is not, use a `module swap` command, such as:

```
% module swap PrgEnv-pgi PrgEnv-gnu
```

`PrgEnv-gnu` loads the product modules that define the system paths and environment variables needed to use the GNU compilers. GCC includes the Fortran 95, C, and C++ compilers.

To compile an application that will run on quad-core compute nodes, load these modules:

```
% module load gcc/4.2.0.quadcore
% module load xtpe-quadcore
```

Table 3 lists the commands for invoking the GNU compilers and the source file extensions.

Table 3. GNU Compiler Commands

| Compiler            | Command          | Source File                                                                           |
|---------------------|------------------|---------------------------------------------------------------------------------------|
| C compiler          | <code>cc</code>  | <code>filename.c</code>                                                               |
| C++ compiler        | <code>CC</code>  | <code>filename.cc</code> ,<br><code>filename.c++</code> ,<br><code>filename.C</code>  |
| Fortran 95 compiler | <code>ftn</code> | <code>filename.f</code> ,<br><code>filename.f90</code> ,<br><code>filename.f95</code> |

The *Using the GNU Compiler Collection (GCC)* manual provides general information about the GNU compilers. The *Using GNU Fortran* includes information about compiler features unique to Cray (see <http://gcc.gnu.org/onlinedocs/>).



**Caution:** To invoke a GNU compiler, use the `cc`, the `CC`, or the `ftn` command. If you invoke a compiler directly using a `gcc`, `g++`, or `gfortran` command, the resulting executable will not run on Cray XT compute nodes.

The `cc(1)`, `CC(1)`, and `ftn(1)` man pages contain information about the compiler driver commands, whereas the `gcc(1)`, `g++(1)`, and `gfortran(1)` man pages contain descriptions of the GNU C compiler command options.

To verify that you are using the correct version of a GNU compiler, use the `--version` option on a `cc`, `CC`, or `ftn` command.

**Note:** To use CrayPat with a GNU program to trace functions, use the `-finstrument-functions` option instead of `-Mprof=func` when compiling your program.

### 5.2.3 Using PathScale Compilers

To use the PathScale compilers, run the `module list` command to verify that the `PrgEnv-pathscales` module is loaded. If it is not, use a `module swap` command, such as:

```
% module swap PrgEnv-pgi PrgEnv-pathscales
```

PrgEnv-pathscale loads the product modules that define the system paths and environment variables needed to use the PathScale compilers.

Table 4 lists the commands for invoking the PathScale compilers and the source file extensions:

Table 4. PathScale Compiler Commands

| Compiler               | Command | Source File                                         |
|------------------------|---------|-----------------------------------------------------|
| C compiler             | cc      | <i>filename.c</i>                                   |
| C++ compiler           | CC      | <i>filename.CC</i>                                  |
|                        |         | <i>filename.cc</i>                                  |
|                        |         | <i>filename.cpp</i>                                 |
|                        |         | <i>filename.cxx</i>                                 |
| Fortran 90/95 compiler | ftn     | <i>filename.f</i> (fixed source, no preprocessing)  |
|                        |         | <i>filename.f90</i> (free source, no preprocessing) |
|                        |         | <i>filename.f95</i> (free source, no preprocessing) |
|                        |         | <i>filename.F</i> (fixed source, preprocessing)     |
|                        |         | <i>filename.F90</i> (free source, preprocessing)    |
|                        |         | <i>filename.F95</i> (free source, preprocessing)    |

To verify that you are using the correct version of a PathScale compiler, use the `-version` option on a `cc`, `CC`, or `ftn` command.



**Caution:** To invoke a PathScale compiler, use either the `cc`, `CC`, or `ftn` command. If you invoke a compiler directly using a `pathcc`, `pathCC`, or `path95` command, the resulting executable will not run on Cray XT compute nodes.

The `cc(1)`, `CC(1)`, and `ftn(1)` man pages contain information about the compiler driver commands, whereas the `pathcc(1)`, `pathCC(1)`, and `path95(1)` man pages contain descriptions of the PathScale compiler command options.

The `eko(7)` man page gives the complete list of options and flags for the PathScale compiler suite.

For more information about using the compiler commands, see the PathScale manuals (<http://www.pathscale.com/docs/html>) and the following man pages:

- Introduction to PathScale compilers: `pathscale-intro(1)` man page
- C compiler: Cray `cc(1)` man page and PathScale `pathcc(1)` and `eko(7)` man pages
- C++ compiler: Cray `CC(1)` man page and PathScale `pathCC(1)` and `eko(7)` man pages
- Fortran compiler: Cray `ftn(1)` man page and PathScale `path95(1)` and `eko(7)` man pages



# Getting Compute Node Status [6]

---

Before running applications, you should check the status of the compute nodes. First, use either the `xtprocadmin -a` or the `cnselect -L osclass` command to find out whether CNL or Catamount is running on the compute nodes.

For the `xtprocadmin -a` report, the OS field value is either CNL or Catamount for all compute nodes, and (service) for all service nodes. For the `cnselect -L osclass` report, `osclass` is 1 for Catamount and 2 for CNL.

```
% xtprocadmin -a os
 NID (HEX) NODENAME TYPE OS
 0 0x0 c0-0c0s0n0 service (service)
 3 0x3 c0-0c0s0n3 service (service)
 4 0x4 c0-0c0s1n0 service (service)
 7 0x7 c0-0c0s1n3 service (service)
 8 0x8 c0-0c0s2n0 service (service)
 11 0xb c0-0c0s2n3 service (service)
 12 0xc c0-0c0s3n0 compute CNL
 13 0xd c0-0c0s3n1 compute CNL
 14 0xe c0-0c0s3n2 compute CNL
 15 0xf c0-0c0s3n3 compute CNL
 16 0x10 c0-0c0s4n0 compute CNL
 17 0x11 c0-0c0s4n1 compute CNL
<snip>
 94 0x5e c0-0c2s7n2 compute CNL
 95 0x5f c0-0c2s7n3 compute CNL

% cnselect -L osclass
2
```

Then use the `xtnodestat` command to display current job and node status. This command combines replaces the `xtshowmesh` and `xtshowcabs` commands. Each character in the display represents a single node. For systems running a large number of jobs, more than one character may be used to designate a job.

% `xtnodestat`

Current Allocation Status at Thu Apr 24 09:10:44 2008

```

 C0-0 C1-0 C2-0 C3-0
n3 ----- ----- ---bbc bd -----kA
n2 ----- ----- ---abbc d -----kk
n1 ----- ----- ----bbc d -----kA
c2n0 ----- ----- ----bbc b -----kA
n3 Y----- ----- ----- iih---j-
n2 ----- ----- ----- iihf--jj
n1 ----- ----- ----- iiAf--jj
c1n0 Y----- ----- ----- iAif----
n3 SSSSSSS ----- ----- eeAggfhi
n2 ----- ----- ----- eeAgggfh
n1 ----- ----- ----- eeAfAgfh
c0n0 SSSSSSS ----- ----- eeeAAGfh
 s01234567 01234567 01234567 01234567

```

Legend:

```

nonexistent node S service node
; free interactive compute CNL - free batch compute node CNL
A allocated, but idle compute node ? suspect compute node
X down compute node Y down or admin down service node
Z admin down compute node R node is routing

```

Available compute nodes:            0 interactive,            270 batch

| Job ID | User         | Size | Age   | command line |
|--------|--------------|------|-------|--------------|
| a      | 340549 user1 | 1    | 0h00m | multi_stress |
| b      | 340548 user2 | 10   | 0h00m | sc           |
| c      | 340512 user3 | 4    | 0h00m | openfile     |
| d      | 340439 user4 | 3    | 0h01m | joto         |
| e      | 340437 user5 | 9    | 0h01m | ftest03      |
| f      | 340569 user6 | 8    | 0h00m | dup2_06      |
| g      | 340531 user7 | 7    | 0h00m | ptl_flood    |
| h      | 340461 user8 | 6    | 0h01m | env          |

|   |        |        |   |       |           |
|---|--------|--------|---|-------|-----------|
| i | 340428 | user9  | 9 | 0h01m | growfiles |
| j | 340564 | user10 | 5 | 0h00m | lmd       |
| k | 340559 | user11 | 5 | 0h00m | growfiles |

**Note:** If `xtnodestat` indicates that no compute nodes have been allocated for interactive processing, you can still run your job interactively by using the `qsub -I` command and then, when your job has been queued, using either the `aprun` or the `yod` application launch command.

For more information, see the `xtprocadmin(1)` and `xtnodestat(1)` man pages.



# Running CNL Applications [7]

---

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

This chapter describes how to run applications interactively on CNL compute nodes and get application status reports. For a description of batch job processing, see [Chapter 10, page 87](#).

## 7.1 Using the `aprun` Command

Use the `aprun` command to specify the resources your application requires, request application placement, and initiate application launch.

**Note:** Verify that you are in a Lustre-mounted directory before using the `aprun` command (see [Section 2.4, page 11](#)).

The format of the `aprun` command is:

```
aprun [-a arch] [-b] [-cc cpu_list | keyword]
[-cp cpu_placement_file_name] [-d depth] [-D value]
[-L node_list] [-m size[h|hs]] [-n pes]
[-N pes_per_node] [-q] [-S pes_per_numa_node]
[-sl list_of_numa_nodes] [-sn numa_nodes_per_node]
[-ss] [-t sec] executable [arguments_for_executable]
```

where:

`-a arch` Specifies the architecture type of the compute node on which the application will run; *arch* is *xt*. If you are using `aprun` to launch a compiled and linked executable, you need not include the `-a` option; ALPS can determine the compute node architecture type from the ELF header (see the `elf(5)` man page). However, if you are using `aprun` to run a shell script, you need to include the `-a` option.

`-b` Bypasses the transfer of the executable to compute nodes. By default, the executable is transferred to the compute nodes as part of the `aprun` process of launching an application. You would likely use the `-b` option only if the executable to be launched was part of the compute node's boot image file system. For an example, see [Section 14.14, page 147](#).

`-cc cpu_list | keyword`

Binds processing elements (PEs) to CPUs. This option applies to all multicore compute nodes. For further information about binding (*CPU affinity*), see [Section 13.3, page 111](#).

The *cpu\_list* is a comma-separated or hyphen-separated list of logical CPU numbers and/or CPU ranges. As PEs are created, they are bound to the CPU in *cpu\_list* corresponding to the number of PEs that have been created at that point. For example, the first PE created is bound to the first CPU in *cpu\_list*, the second PE created is bound to the second CPU in *cpu\_list*, and so on. If more PEs are created than given in *cpu\_list*, binding starts over at the beginning of *cpu\_list* and resumes with the first CPU in *cpu\_list*. The *cpu\_list* can also contain an `x`, which indicates that the application-created process at that location in the fork sequence should **not** be bound to a CPU.

Out-of-range *cpu\_list* values are ignored unless all CPU values are out of range. For example, if you want to bind PEs starting with the highest CPU on a compute node and work down from there, you might use this `-cc` option:

```
% aprun -n 8 -cc 7-0 ./a.out
```

If the PEs were placed on a Cray XT5 compute node, the specified `-cc` range would be valid. However, if the PEs were placed on Cray XT4 compute nodes, CPUs 7-4 would be out of range and therefore not used. For more information about CPU affinity, see [Section 14.13, page 146](#).

The following *keyword* values are supported:

- The `cpu` keyword (the default) binds each PE to a CPU within the assigned NUMA node. You do not have to indicate a specific CPU. The `-cc cpu` option is the typical use case for an MPI application.

A PE's threads or child processes can be constrained to the CPUs closest to the PE's CPU. If an application creates more threads or child processes than CPUs allocated for the PE *depth*, the thread or child process wraps to the PE's CPU, and process placement continues within the range specified by *depth*. For example, this command makes room for a PE to create three threads or child processes:

```
% aprun -n 8 -N 2 -d 4 -cc cpu a.out
```

The processes would be placed like this:

```
cpu 0: child0_of_PE0
cpu 1: child1_of_PE0
cpu 2: child2_of_PE0
cpu 3: child3_of_PE0
cpu 4: child0_of_PE1
cpu 5: child1_of_PE1
cpu 6: child2_of_PE1
cpu 7: child3_of_PE1
```

If the application created one additional thread or child process per PE, it would look like this:

```
cpu 0: child0_of_PE0 + child4_of_PE0
cpu 1: child1_of_PE0
cpu 2: child2_of_PE0
cpu 3: child3_of_PE0
cpu 4: child0_of_PE1 + child4_of_PE1
cpu 5: child1_of_PE1
cpu 6: child2_of_PE1
cpu 7: child3_of_PE1
```

- The `numa_node` keyword causes a PE to be constrained to the CPUs within the assigned NUMA node. CNL can migrate a PE among the CPUs in the assigned NUMA node but not off the assigned NUMA node. For example, if PE2 is assigned to NUMA node 0, CNL can migrate PE2 among CPUs 0-3 but not among CPUs 4-7.

If PEs create threads, the threads are constrained to the same NUMA-node CPUs as the PEs. There is one exception. If *depth* is greater than the number of CPUs per NUMA node, once the number of threads created by the PE has exceeded the number of CPUs per NUMA node, the remaining threads are constrained to CPUs within the next NUMA node on the compute node. For example, if *depth* is 5, threads 0-3 are constrained to CPUs 0-3 and thread 4 is constrained to CPUs 4-7.

- The `none` keyword allows PE migration within the assigned *cpuset*. A *cpuset* is a process container that controls memory and CPU usage.

`-cp` *cpu\_placement\_file\_name*

(Deferred implementation) Provides the name of a CPU binding placement file. This option applies to all multicore compute nodes. This file must be located on a file system accessible from the compute nodes. The CPU placement file provides more extensive CPU binding instructions than the `-cc` options.

`-D` *value*

Directs `aprun` to write debug messages to `stdout`, where *value* is a positive integer. Increasing *value* increases the verbosity of the debug messages. Do not specify the `-D` option with the `-q` (quiet) option; `aprun` terminates the application if both options are specified. Debugging is disabled by default.

`-d` *depth*

Specifies the number of CPUs to host OpenMP threads. ALPS allocates the number of CPUs equal to *depth* times *pes*. The default *depth* is 1.

For OpenMP applications, use the `-d` option with the `OMP_NUM_THREADS` environment variable to specify the number of threads and the number of CPUs hosting the threads. ALPS creates `-n pes` instances of the executable, and the executable spawns `OMP_NUM_THREADS-1` additional threads per PE. For an example, see [Section 14.7, page 135](#)

`-L node_list`

Specifies the candidate nodes to constrain application placement. The syntax allows a comma-separated list of nodes (such as `-L 32,33,40`), a range of nodes (such as `-L 41-87`), or a combination of both formats. Node values can be expressed in decimal, octal (preceded by 0), or hexadecimal (preceded by 0x). The first number in a range must be less than the second number (8-6, for example, is invalid), but the nodes in a list can be in any order. This option is used for interactive jobs; use the `qsub -lmpnodes` option for batch and interactive batch jobs.

If the placement node list contains fewer nodes than the number required, a fatal error is produced. If resources are not currently available, `aprun` continues to retry.

A common source of node lists is the `cnselect` command. See the `cnselect(1)` man page for details.

`-m size[h|hs]`

Specifies the per-PE required Resident Set Size (RSS) memory size in megabytes. K, M, and G suffixes (case insensitive) are supported (16M = 16m = 16 megabytes, for example). If you do not include the `-m` option, the default amount of memory available to each PE equals the minimum compute node memory size divided by the maximum number of CPUs on any compute node. For example, given Cray XT5 compute nodes with 32 GB of memory, the default per-PE memory size is  $32 \text{ GB} / 8 \text{ CPUs} = 4 \text{ GB}$ . For an example, see [Section 7.4, page 72](#).

If you want huge pages (2 MB) allocated to your application, use the `h` or `hs` suffix, as shown in the following table. The default is base page size is 4 KB.

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-m sizeh</code>  | <p><b>Requests</b> <i>size</i> MB of huge pages to be allocated to each PE. All nodes use as much huge page memory as they are able to allocate and 4 KB pages thereafter. This command <b>requests</b> 4000 MB of huge pages per PE:</p> <pre>% aprun -n 8 -m4000mh -q ./hugepgs</pre>                                                                                                                                                    |
| <code>-m sizehs</code> | <p><b>Requires</b> <i>size</i> of huge pages to be allocated to each PE. If the request cannot be satisfied, an error message is issued and <code>aprun</code> terminates the request. This example terminates because the <b>required</b> 4000 MB of huge pages per PE are not available:</p> <pre>% aprun -n 8 -m4000mhs -q ./huhepgs [NID 00045] Apid 651277: unable to acquire enough huge memory: desired 16000M, actual 12396M</pre> |

**Note:** To use huge pages, you must first load the huge pages library during the linking phase, such as:

```
% cc -c my_hugepages_app.c
% cc -o my_hugepages_app my_hugepages_app.o
-lhugetlbf
```

Then set the huge pages environment variable:

```
% setenv HUGETLB_MORECORE yes
```

OR

```
% export HUGETLB_MORECORE=yes
```

- `-n pes` Specifies the number of processing elements (PEs) needed for your application. A PE is an instance of an ALPS-launched executable. The number of PEs can be expressed in decimal, octal, or hexadecimal form. If *pes* has a leading 0, it is interpreted as octal (`-n 16` specifies 16 PEs, but `-n 016` is interpreted as 14 PEs). If *pes* has a leading 0x, it is interpreted as hexadecimal (`-n 16` specifies 16 PEs, but `-n 0x16` is interpreted as 22 PEs). Default is 1.
- `-N pes_per_node` Specifies the number of PEs to place per node. You can use this option to reduce the number of PEs per node, thereby making more resources available for each PE. Compute nodes must have at least *pes\_per\_node* CPUs. The default is the number of CPUs on a node.
- `-q` Specifies quiet mode and suppresses all `aprun`-generated non-fatal messages. Do not use this option with the `-D` (debug) option; `aprun` terminates the application if both options are specified. Even with the `-q` option, `aprun` writes its help message and any fatal messages when exiting.
- `-S pes_per_numa_node` Specifies the number of PEs to allocate per NUMA node. This option applies to Cray XT5 compute nodes.
- The *pes\_per\_numa\_node* value can be 1, 2, 3, or 4. The default is 4. A zero value is not allowed and is a fatal error. For further information, see [Section 13.2, page 110](#).
- `-sl list_of_numa_nodes` Specifies the NUMA node or nodes (comma separated or hyphen separated) to use for application placement. This option applies to Cray XT5 compute nodes.
- The *list\_of\_numa\_nodes* value can be `-sl 0`, `-sl 1`, `-sl 0,1` (or its equivalent, `-sl 0-1`). The default is `-sl 0,1`.
- List NUMA nodes in ascending order; `-sl 1-0` and `-sl 1,0` are invalid. For more information, see [Section 13.2, page 110](#).

`-sn numa_nodes_per_node`

Specifies the number of NUMA nodes per node to be allocated. This option applies to Cray XT5 compute nodes. The `numa_nodes_per_node` value can be 1 or 2. The default is 2. You can use this option to find out if restricting your PEs to one NUMA node per node affects performance.

A zero value is not allowed and is a fatal error. For more information, see [Section 13.2, page 110](#).

`-ss`

Specifies strict memory containment per NUMA node. This option applies to Cray XT5 compute nodes. When you use the `-ss` option, a PE can allocate only the memory local to its assigned NUMA node. The default is to allow remote-NUMA-node memory allocation.

By default, any PE running on NUMA node 0 can access NUMA node 1 memory and vice versa. You can use the `-ss` option to find out if restricting each PE's memory access to local-NUMA-node memory affects performance. For more information, see [Section 13.2, page 110](#).

`-t sec`

Specifies the per-PE CPU time limit in seconds. The `sec` time limit is constrained by your CPU time limit on the login node. For example, if your time limit on the login node is 3600 seconds but you specify a `-t` value of 5000, your application is constrained to 3600 seconds per PE. If your time limit on the login node is unlimited, the `sec` value is used (or, if not specified, the time per-PE is unlimited).

:

Separates the names of executables and their associated options for Multiple Program, Multiple Data (MPMD) mode.

For single-core nodes, ALPS creates `-n` PEs and launches them on `pes` nodes. For example, the command:

```
% aprun -n 64 ./prog1
```

creates 64 instances of `prog1` and launches them on 64 nodes.

For multicore nodes, ALPS creates `-n` PEs and uses the `-N pes_per_node` value to determine where to place them. Whenever possible, ALPS packs the PEs, using the smallest number of nodes to fulfill the `-n` requirements. If you specify `-N 1`, ALPS assigns one PE per node. For example, the command:

```
% aprun -n 32 ./prog1
```

creates 32 instances of `prog1` and launches them on 16 dual-core nodes, 8 quad-core nodes, or 4 dual-socket quad-core nodes. In contrast, the command:

```
% aprun -n 32 -N 1 ./prog1
```

creates 32 instances of `prog1` and launches them on one core per node of 32 nodes. The other cores on those nodes are unused.

## 7.2 Using the `apstat` Command

The `apstat` command provides status information about reservations, compute resources, pending and placed applications, and cores. The format of the `apstat` command is:

```
apstat [-a] [-A apid ... | -R resid ...][-n] [-p]
[-r] [other arguments]
```

You can use `apstat` to display the following types of status information:

- all applications
- placed applications
- applications by application IDs (APIDs)
- applications by reservation IDs (ResIDs)
- nodes and cores
- pending applications
- confirmed and claimed reservations

For example:

```
% apstat -a
Total placed applications: 3
Placed Apid ResID User PEs Nodes Age State Command
 48062 39 bill 2 1 2h39m run MPI_Issend_perf
 48108 1588 jim 4 1 0h15m run gtp
 48109 1589 sue 4 1 0h07m run bench6
```

An Apid in the apstat display is also displayed after aprun execution results.  
For example:

```
% aprun -n 2 -d 2 ./ompl
Hello from rank 0 (thread 0) on nid00540
Hello from rank 1 (thread 0) on nid00541
Hello from rank 0 (thread 1) on nid00540
Hello from rank 1 (thread 1) on nid00541
Application 48109 resources: utime 0, stime 0%
```

You can use the -n option to display core status. For example:

```
% apstat -nv
NID Arch State HW Rv Pl PgSz Avl Conf Placed PEs Apids
 24 XT UP B 8 8 - 4K 4096000 2048000 0 0
 25 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
 26 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
 27 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
 28 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
 29 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
 30 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
<snip>
 88 XT UP B 8 8 8 4K 4096000 2048000 2048000 8 832494
 89 XT UP B 8 8 - 4K 4096000 2048000 0 0
 90 XT UP B 8 - - 4K 4096000 0 0 0
 91 XT UP B 8 - - 4K 4096000 0 0 0
 92 XT UP B 8 - - 4K 4096000 0 0 0
 93 XT UP B 8 - - 4K 4096000 0 0 0
 94 XT UP B 8 - - 4K 4096000 0 0 0
 95 XT UP B 8 - - 4K 4096000 0 0 0
128 XT UP B 8 8 8 4K 8192000 2048000 2048000 8 832489
<snip>
Compute node summary
 arch config up use held avail down
 XT 356 356 152 2 202 0
```

where `HW` is the number of cores in the node, `Rv` is the number of cores held in a reservation, and `P1` is the number of cores being used by an application. If you want to display a 0 instead of a - in the `Rv` and `P1` fields, add the `-z` option to the `apstat` command.

For further information, see the `apstat(1)` man page.

### 7.3 Using the `cnselect` Command

The `aprun` utility supports manual and automatic node selection. For manual node selection, first use the `cnselect` command to get a candidate list of compute nodes that meet the criteria you specify. Then for interactive jobs use the `aprun -L node_list` option. For batch and interactive batch jobs, add `-lmpnodes="node_list"` to the job script or the `qsub` command line.

The format of the `cnselect` command is:

```
cnselect -l | -L fieldname | -V | [-c] [-e] expression
```

where:

- `-l` lists names of fields in the compute nodes attributes database.
- `-L fieldname` lists the current possible values for a given field.
- `-V` prints the version number and exits.
- `-c` gives a count of the number of nodes rather than a list of the nodes themselves.
- `[-e] expression` queries the compute node attributes database.

You can use `cnselect` to get a list of nodes selected by such characteristics as number of cores per node (`coremask`), amount of memory on the node (in megabytes), and processor speed (in megahertz). For example, to run an application on Cray XT5 nodes with 16 GB of memory or more, use:

```
% cnselect availmem.gt.16000 .and. coremask.eq.255
128-223,256-351,384-447
% aprun -n 16 -L 128-223 ./appl
```

You can also use `cnselect` to get a list of nodes if there is a site-defined label. For example, to run an application on quad-core nodes, you might use:

```
% cnselect -L label1
SINGLE-CORE
DUAL-CORE
QUAD-CORE
8-CORE
% cnselect -e label1.eq.'QUAD-CORE'
60-63,76,82
% aprun -n 6 -L 60-63,76,82 ./app1
```

You could accomplish the same by using:

```
% cnselect coremask.eq.15
60-63,76,82
```

If you do not include the `-L` option on the `aprun` command or the `-lmpnodes` option on the `qsub` command, ALPS automatically places the application using available resources.

## 7.4 Understanding How Much Memory is Available to CNL Applications

When running large applications, it is important to understand how much memory will be available per node. CNL uses memory on each node for CNL and for other functions such as I/O buffering. The remaining memory is available for user executables; user data arrays; stacks, libraries and buffers; and SHMEM symmetric stack heap.

The amount of memory CNL uses depends on the number of cores, memory size, and whether or not optional software has been configured on the compute nodes. For a quad-core node with 8 GB of memory, 7.2 to 7.5 GB of memory is available for applications.

The default stack size is 16 MB. You can determine the maximum stack size by using the `limit` command (csh) or the `ulimit -a` command (bash). The memory used for the MPI libraries is approximately 72 MB per node.

**Note:** The actual amount of memory CNL uses varies depending on the total amount of memory on the node and the OS services configured for the node.

You can use the `aprun -m size` option to specify the per-PE memory limit. For example, this command launches `xthi` on cores 0 and 1 of compute nodes 472 and 473. Each node has 8 GB of available memory.

```
% aprun -n 4 -N 2 -m4000 -q ./xthi | sort
PE 0 nid00472 Core affinity = 0,1
PE 1 nid00472 Core affinity = 0,1
PE 2 nid00473 Core affinity = 0,1
PE 3 nid00473 Core affinity = 0,1
% aprun -n 4 -N 2 -m4001 -q ./xthi | sort
Claim exceeds reservation's memory
```

You can change MPI buffer sizes and stack space from the defaults by setting certain environment variables. For more details, see the `intro_mpi(3)` man page.

## 7.5 Launching an MPMD Application

The `aprun` utility supports multiple-program, multiple-data (MPMD) mode. To run an application in MPMD mode under `aprun`, use the `-n pes executable1: -n pes executable2: ...` format. All of the executables share the same `MPI_COMM_WORLD` process communicator.

For example, this command launches 128 instances of `program1` and 256 instances of `program2`:

```
aprun -n 128 ./program1: -n 256 ./program2
```

## 7.6 Managing Compute Node Processors from an MPI Program

MPI programs should call the `MPI_Finalize()` routine at the conclusion of the program. This call waits for all processing elements to complete before exiting. If one of the programs fails to call `MPI_Finalize()`, the program never completes and `aprun` stops responding. There are two ways to prevent this behavior:

- Use the PBS Professional elapsed (wall clock) time limit to terminate the job after a specified time limit (such as `-l walltime=2:00:00`).
- Use the `aprun -t sec` option to terminate the offending program. This option specifies the per-PE CPU time limit in seconds. A process will terminate only if it reaches the specified amount of CPU time (not wallclock time).

For example, if you use:

```
% aprun -n 8 -t 120 ./myprog1
```

and a PE uses more than two minutes of CPU time, `aprun` terminates the application.

## 7.7 About `aprun` Input and Output Modes

The `aprun` utility handles standard input (`stdin`) on behalf of the user and handles standard output (`stdout`) and standard error messages (`stderr`) for user applications.

For other I/O considerations, see [Section 4.2.2, page 31](#).

## 7.8 About aprun Resource Limits

The `aprun` command currently forwards its user resource limits, both soft and hard (see the `getrlimit(P)` man page) to each compute node, where those limits are set for the application. The limits that are currently forwarded are:

- `RLIMIT_CPU`
- `RLIMIT_FSIZE`
- `RLIMIT_DATA`
- `RLIMIT_STACK`
- `RLIMIT_CORE`
- `RLIMIT_RSS`
- `RLIMIT_NPROC`
- `RLIMIT_NOFILE`
- `RLIMIT_MEMLOCK`
- `RLIMIT_AS`
- `RLIMIT_LOCKS`
- `RLIMIT_SIGPENDING`
- `RLIMIT_MSGQUEUE`
- `RLIMIT_NICE`
- `RLIMIT_RTPRIO`

This forwarding of user resource limits can cause problems on systems where the login node's limits are more restrictive than the default compute node limits. Setting the `APRUN_XFER_LIMITS` environment variable to 0 (`export APRUN_XFER_LIMITS=0`) will disable the forwarding of user resource limits (except for `RLIMIT_CORE`).

**Note:** The default forwarding of user resource limits will be eliminated in a future release of ALPS.

## 7.9 About aprun Signal Processing

The `aprun` utility forwards the following signals to an application:

- `SIGHUP`
- `SIGINT`
- `SIGQUIT`
- `SIGTERM`
- `SIGABRT`
- `SIGUSR1`
- `SIGUSR2`
- `SIGURG`
- `SIGWINCH`

The `aprun` utility ignores `SIGPIPE` and `SIGTTIN` signals. All other signals remain at their default behavior and are not forwarded to an application. The default behaviors that terminate `aprun` also cause ALPS to terminate the application with a `SIGKILL` signal.

# Running Catamount Applications [8]

---

The `yod` utility launches applications on Catamount compute nodes. When you start a `yod` process, the application launcher coordinates with the Compute Processor Allocator (CPA) to allocate nodes for the application and then uses Process Control Threads (PCTs) to transfer the executable 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.

This chapter describes how to run applications interactively on Catamount compute nodes. For a description of batch job processing, see [Chapter 10, page 87](#).

## 8.1 Using the `yod` Command

When launching an application with the `yod` command, you can specify the number of processors to allocate to the application.

The format of the `yod` command is:

```
% yod -sz n [other arguments] executable
```

where *n* is the number of processors on which the application will run.

The `yod -sz`, `-size`, and `-np` options are synonymous.

The following paragraphs describe the differences in the way processors are allocated on single-core and dual-core processor systems.

- Running applications on single-core processor systems

On single-core processor systems, each compute node has one single-core AMD Opteron processor. Applications are allocated `-sz` nodes.

For example, the command:

```
% yod -sz 6 prog1
```

launches `prog1` on six nodes.

Single-core processing is the default. However, sites can change the default to dual-core processor mode. Use `-SN` if the default is dual-core processor mode and you want to run applications in single-core processor mode.

**Note:** The `yod -VN` option turns on virtual node processing mode. The `yod` utility runs the program on both cores of a dual-core processor. If you use the `-VN` option on a single-core system, the application load will fail.

- Running applications on dual-core processor systems

On dual-core processor systems, each compute node has one dual-core AMD Opteron processor. The processors are managed by the Catamount Virtual Node (CVN) kernel. To launch an application, you must include the `-VN` option on the `yod` command unless your site has changed the default.

On a dual-core system, if you do not include the `-VN` option, your program will run on one core per node, with the other core idle. You may do this if you must use all the memory on a node for each processing element or if you want the fastest possible run time and do not mind letting the second core on each node sit idle.

## 8.2 Using the `cnselect` Command

The `yod` utility supports automatic and manual node selection. To use manual node selection, first use the `cnselect` command to get a list of compute nodes that meet the criteria you specify. Then use the `yod -list processor-list` option to launch the application. If the number of nodes in the list is greater than the `-sz n` value, `yod` selects `n` of the `processor-list` nodes on which to launch the application.

The format of the `cnselect` command is:

```
cnselect -l | -L fieldname | -V | [-c] [-y] [-e] expression
```

where:

- `-l` lists names of fields in the compute nodes attributes database.
- `-L fieldname` lists the current possible values for a given field.
- `-V` prints the version number and exits.
- `-c` gives a count of the number of nodes rather than a list of the nodes themselves.
- `-y` puts the `yod` range separator ( . . ) in output ranges in place of the default hyphen (-).
- `[-e] expression` queries the compute node attributes database.

You can use `cselect` to get a list of nodes selected by such characteristics as number of cores per node (`coremask`), available memory (in megabytes), and processor speed (in megahertz). For example, to run an application on dual-core nodes with 2 GB of memory or more, use:

```
% cselect -y availmem .ge. 2000 .and. coremask .gt. 1
44..63,76,82
% yod -vN -sz 16 -list 44..59 ./app1
```

If you do not include the `-list` option or the `-lmpnodes` option on the `qsub` command, `yod` automatically places the application per available resources.

### 8.3 Understanding How Much Memory is Available to Catamount Applications

When running large applications on a dual-core processor system, it is important to understand how much memory will be available per node for your job.

If you are running in single-core mode on a dual-core system, Catamount (the kernel plus the process control thread (PCT)) uses approximately 120 MB of memory. The remaining memory is available for the user program executable, user data arrays, the stack, libraries and buffers, and SHMEM symmetric stack heap.

For example, on a node with 2.147 GB of memory, memory is allocated as follows:

|                                                                                   |                       |
|-----------------------------------------------------------------------------------|-----------------------|
| Catamount                                                                         | 120 MB (approximate)  |
| Executable, data arrays, stack, libraries and buffers, SHMEM symmetric stack heap | 2027 MB (approximate) |

If you are running in dual-core mode, Catamount uses approximately 120 MB of memory (the same as for single-core mode). The PCT divides the remaining memory, allocating half to each core. The memory allocated to each core is available for the user executable, user data arrays, stack, libraries and buffers, and SHMEM symmetric stack heap.

For example, on a node with 2.147 GB of memory, memory is allocated as follows:

---

|                                                                                              |                       |
|----------------------------------------------------------------------------------------------|-----------------------|
| Catamount                                                                                    | 120 MB (approximate)  |
| Executable, data arrays, stack, libraries and buffers, SHMEM symmetric stack heap for core 0 | 1013 MB (approximate) |
| Executable, data arrays, stack, libraries and buffers, SHMEM symmetric stack heap for core 1 | 1013 MB (approximate) |

---

The default stack size is 16 MB. (You can determine the maximum stack size through the `limit` command (csh) or the `ulimit -a` command (bash).)

The memory used for the Lustre and MPI libraries is as follows:

---

|                                |                     |
|--------------------------------|---------------------|
| Lustre library                 | 17 MB (approximate) |
| MPI library and default buffer | 72 MB (approximate) |

---

You can change MPI buffer sizes and stack space from the defaults by setting certain environment variables. For more details, see the `intro_mpi(3)` man page.

## 8.4 Launching an MPMD Application

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

These `yod` options are valid within a loadfile:

`-heap size`

Specifies the number of bytes to reserve for the heap. The minimum value of *size* is 16 MB. On dual-core systems, each core is allocated *size* bytes.

`-list processor-list`

Lists the candidate compute nodes on which to run the application, such as: `-list 42,58,64..100,150..200`. Use the `cnsselect` command with the `-y` option to generate the list. See the `cnsselect(1)` man page for details.

`-shmem size`

Specifies the number of bytes to reserve for the symmetric heap for the SHMEM library. The heap size is rounded up in order to address physical page-boundary issues. The minimum value of *size* is 2 MB. On dual-core systems, each core is allocated *size* bytes.

`-size|-sz|-np n`

Specifies the number of processors on which to run the application. In `SN` mode, `-size n` is the number of nodes. In `VN` mode, `-size n` is the number of cores. You can use the `-size` option with the `-list` option to launch an application on a subset of the `-list processor-list` nodes.

`-stack size`

Specifies the number of bytes to reserve for the stack. On dual-core systems, each core is allocated *size* bytes.

This 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, use:

```
% yod -F loadfile
```

## 8.5 Managing Compute Node Processors from an MPI Program

MPI programs should call the `MPI_Finalize()` routine at the conclusion of the program. This call waits for all PEs to complete before exiting. However, if one of the processes fails to start or stop for any reason, the program never completes and `yod` stops responding. To prevent this behavior, use the `yod -tlimit` option to terminate the application after a specified number of seconds. For example,

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

terminates all processes remaining after 30K (30 \* 1024) seconds so that `MPI_Finalize()` can complete. You can also use the environment variable `YOD_TIME_LIMIT`. The time limit specified on the command line overrides the value specified by the environment variable.

## 8.6 Using Input and Out 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.

For other I/O considerations, see [Section 4.3.2, page 42](#).

## 8.7 About `yod` Signal Handling

The `yod` utility uses two signal handlers, one for the load sequence and one for application execution. During the load operation, any signal sent to `yod` during the load operation terminates the operation. After 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` interprets 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 effectively terminate the running application. The application can ignore the signals that `yod` passes along to it; `SIGTERM`, for example, does not necessarily terminate an application. However, a `SIGINT` delivered to `yod` initiates a forced termination of the application.

## 8.8 Associating a Project or Task with a Job Launch

Use the `yod -Account "project task"` or `-A "project task"` option or the `qsub -A "project task"` 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" -sz 16 myapp123
```

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

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



# Running User Programs on Service Nodes [9]

---

To compile a program that you want to run on a login or other service node, call the PGI, GCC, or PathScale compiler directly.

- For PGI programs, use the `pgcc`, `pgCC`, or `pgf95` command.
- For GCC programs, use the `gcc`, `g++`, or `gfortran` command.
- For PathScale programs, use the `pathcc`, `pathCC`, or `path95` command.

These compilers will find the appropriate header files and libraries in their normal Linux locations.

For example, to run program `my_utility` on a service node, first compile the program:

```
% module load pgi
% pgCC -o my_utility my_utility.C
```

Then run `my_utility`:

```
% my_utility
In main(0)
In functionx(0)
Back in main()
```



# Using PBS Professional [10]

---

Your Cray XT system may include the optional PBS Professional batch scheduling software package from Altair Grid Technologies. If so, your system can be configured with a given number of interactive job processors and a given number of batch processors. 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 remains in the batch queue but never exits.

**Note:** At any time, the system administrator can change the designation of any node from interactive to batch or vice versa. However, this does not affect jobs already running on those nodes. It applies only to jobs already in the queue and jobs submitted later.

The basic process for creating and running batch jobs is to create a PBS Professional job script that includes `aprun` or `yod` commands, then use the PBS Professional `qsub` command to run the script.

## 10.1 Creating Job Scripts

A job script may consist of directives, comments, and executable statements:

```
#PBS -N job_name
#PBS -l resource_type=specification
#
command
command
...
```

PBS Professional provides a number of *resource\_type* options for specifying, allocating, and scheduling compute node resources, such as `mppwidth` (number of processing elements), `mppdepth` (number of threads), `mppnppn` (number of PEs per node), and `mppnodes` (manual node placement list). See [Table 5, page 88](#), [Table 6, page 89](#), and the `pbs_resources(7B)` man page for details.

## 10.2 Submitting Batch Jobs

To submit a job to the batch scheduler, use these commands:

```
% module load pbs
% qsub [-l resource_type=specification] jobscript
```

where *jobscript* is the name of a job script that includes one or more `aprun` or `yod` commands.

The `qsub` command scans the lines of the script file for directives. An initial line in the script that has only the characters `#!` or the character `:` is ignored and scanning starts at the next line. A line with `#!/bin/shell` invokes the *shell* from within the script. Scanning continues until the first executable line. An executable line is defined as a line that is not blank, not a directive, and does not start with `#`). If directives occur on subsequent lines, they are ignored. When you run the script, `qsub` displays the Job ID. You can use the `qstat` command to check on the status of your job and the `qdel` command to remove a job from the queue.

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, are processed as if you included them on the command line.

### 10.2.1 Using `aprun` with `qsub`

[Table 5](#) lists `aprun` options and their counterpart `qsub -l` options:

Table 5. `aprun` versus `qsub` Options

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

For further information about `qsub -l` options, see the `pbs_resources(7B)` man page.

For examples of batch jobs that use `aprun`, see [Section 14.9, page 139](#).

### 10.2.2 Using `yod` with `qsub`

On a single-core system, the PBS Professional `mppwidth` parameter is equivalent to the `yod sz` option.

On a dual-core system, the PBS Professional `mppwidth` parameter is **not** equivalent to the `yod sz` option. The PBS Professional `mppwidth` parameter refers to the number of **nodes** to be allocated for a job. The `yod sz` option refers to the number of **cores** to be allocated for a job (two cores per node).

For example, these commands:

```
% qsub -I -V -l mppwidth=6
% yod -size 12 -VN prog1
```

allocate 6 nodes to the job and launch `prog1` on both cores of each of the 6 nodes.

[Table 6](#) lists `yod` options and their counterpart `qsub -l` options:

Table 6. `yod` versus `qsub` Options

| <code>yod</code> option  | <code>qsub -l</code> option        | Description                        |
|--------------------------|------------------------------------|------------------------------------|
| <code>-sz 4</code>       | <code>-l mppwidth=4</code>         | Number of processors (single core) |
| <code>-VN -sz 8</code>   | <code>-l mppwidth=4</code>         | Number of processors (dual core)   |
| <code>-list 5,6,7</code> | <code>-l mppnodes="\5,6,7\"</code> | Node List                          |

For examples of batch jobs that use `yod`, see [Chapter 15, page 157](#).

### 10.3 Terminating Failing Processes in an MPI Program

Jobs 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 never completes and `aprun` or `yod` stops responding. To prevent this behavior, use the PBS Professional time limit to terminate remaining processes so that `MPI_Finalize()` can complete.

## 10.4 Getting Job Status

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

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

For example:

```
% qstat
Job id Name User Time Use S Queue

84.nid00003 test_ost4_7 usera 03:36:23 R workq
33.nid00003 run.pbs userb 00:04:45 R workq
34.nid00003 run.pbs userb 00:04:45 R workq
35.nid00003 STDIN userc 00:03:10 R workq
```

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

```
% qstat -a
nid00003:

 Time In Req'd Req'd Elap
Job ID Username Queue Jobname SessID Queue Nodes Time S Time

163484 usera workq test_ost4_ 9143 003:48 64 -- R 03:47
163533 userb workq run.pbs 15040 000:48 64 00:30 R 00:15
163534 userb workq run.pbs 15045 000:48 64 00:30 R 00:15
163536 userc workq STDIN 15198 000:10 5 -- R 00:09

Total generic compute nodes allocated: 197
```

For details, see the `qstat(1B)` man page.

## 10.5 Removing a Job from the Queue

The `qdel` command removes a PBS Professional 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`. For more information, see the `qdel(1B)` man page and the *PBS Professional 9.0 User's Guide*.



# Debugging an Application [11]

---

This chapter describes the TotalView debugger, the `lgdb` debugger for CNL applications, and the `xtgdb` debugger for Catamount applications. In addition, this chapter documents the process for analyzing `yod` diagnostics for RPC calls.

**Note:** Before launching a debugger, you need to compile your application with the `-g` option.

## 11.1 Using the TotalView Debugger

Cray XT systems support the TotalView debugger, an optional product from TotalView Technologies, LLC, that provides source-level debugging of applications running on multiple compute nodes. TotalView is compatible with the PGI, GCC, and PathScale compilers.

TotalView:

- Provides both a graphical user interface and a command-line interface (with command-line help)
- Supports the x86-64 Assembler
- Supports programs written in mixed languages
- Supports debugging of up to 4096 compute node processes
- Supports watchpoints
- Provides a memory debugger

TotalView typically is run interactively. If your site has not designated any compute nodes for interactive processing, use the `qsub -I` command.

For more information about the TotalView graphical and command line interfaces, see the `totalview(1)` man page. For more information about TotalView, including details about running on a Cray XT system, see <http://www.totalviewtech.com/Documentation>.

### 11.1.1 Using TotalView to Debug an Application

To debug a CNL application, use this command format to launch an instance of `aprun`, which in turn launches the *executable*:

```
% totalview aprun -a [other_aprun_arguments] ./executable
```

**Note:** The `-a` is a TotalView option indicating that the arguments that follow apply to `aprun`. If you want to use the `aprun -a arch` option, you need to include a second `-a`, as in:

```
% totalview aprun -a -a xt -n 2 ./a.out
```

For example, to debug application `xt1`, use:

```
% totalview aprun -a -n 2 ./xt1
```

The TotalView Root and Process windows appear.

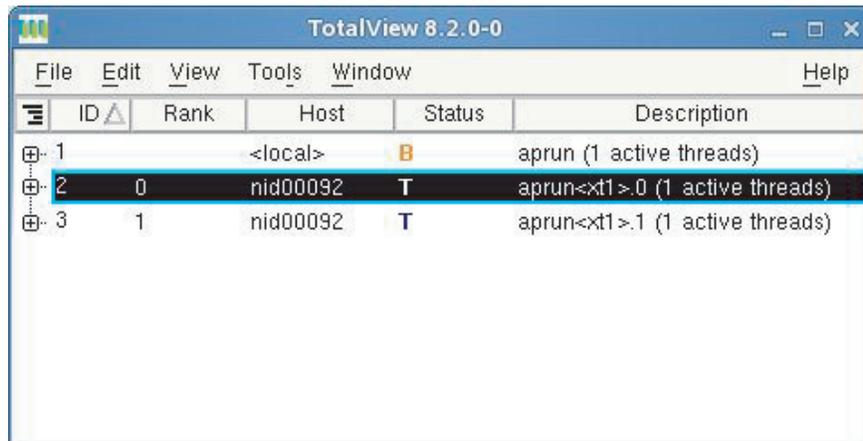


Figure 2. TotalView Root Window

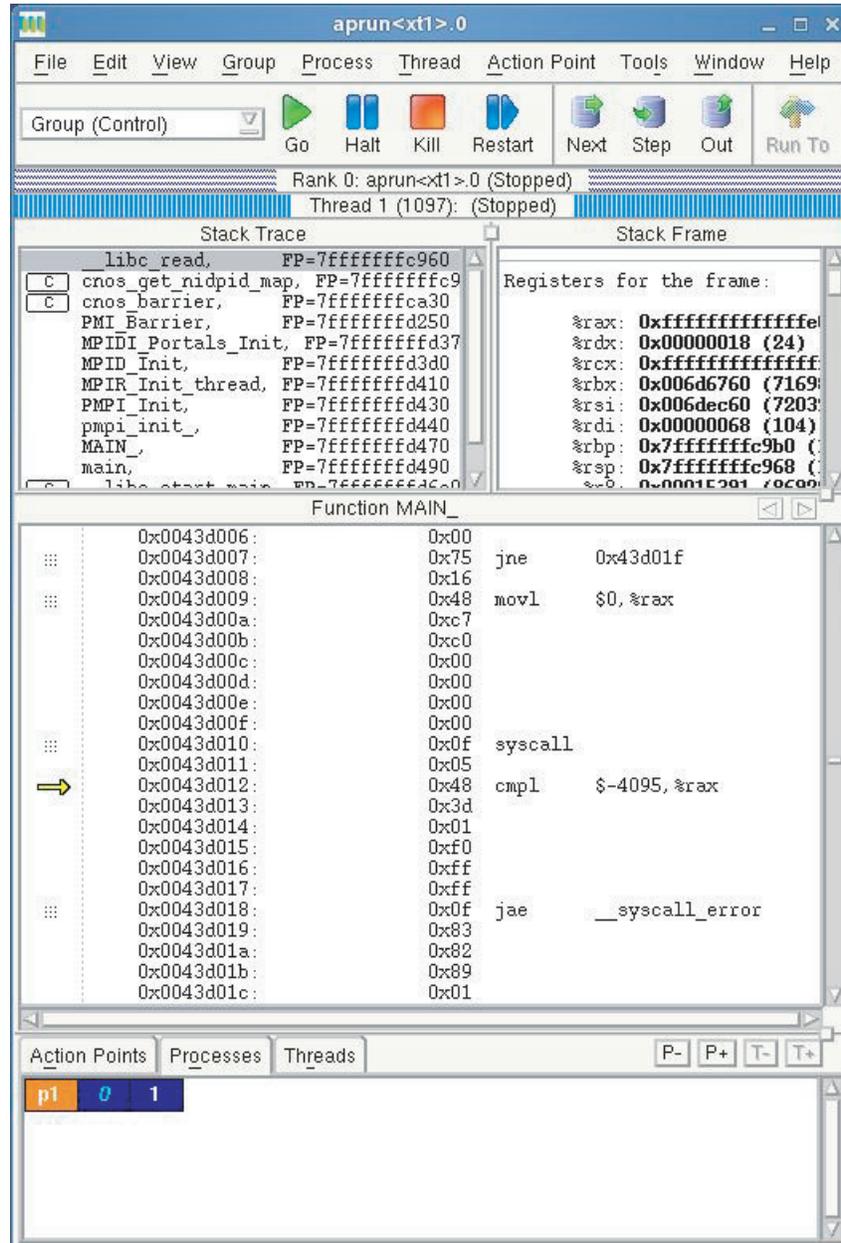


Figure 3. TotalView Process Window

To debug a Catamount application, substitute `yod` for `aprun` in the `totalview` command.

### 11.1.2 Using TotalView to Debug a Core File

To debug a core file,

1. Select **New Program** from the Process window **File** menu. A New Program window appears.

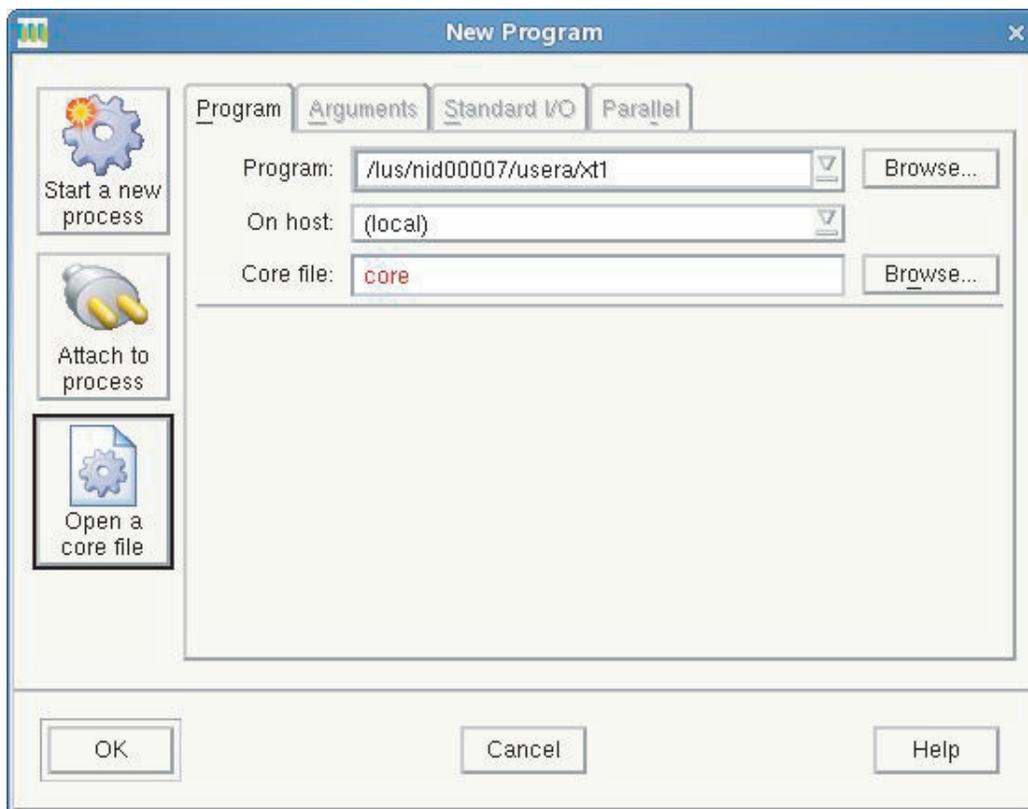


Figure 4. Debugging a Core File

2. Click the **Open a core file** icon.

3. On the **Program** tab, specify the application name in the **Program:** field and the core file name in the **Core file:** field.
4. Click **OK**

### 11.1.3 Using TotalView to Attach to a Running Process

To attach TotalView to a running process, you must be logged in to the same login node that you used to launch the process, and you must attach to the instance of `aprun` that was used to launch the process, rather than to the process itself. To do so, follow these steps:

1. Launch TotalView.

```
% totalview
```

2. In the New Program window, click the **Attach to process** icon. The list of processes currently running displays.

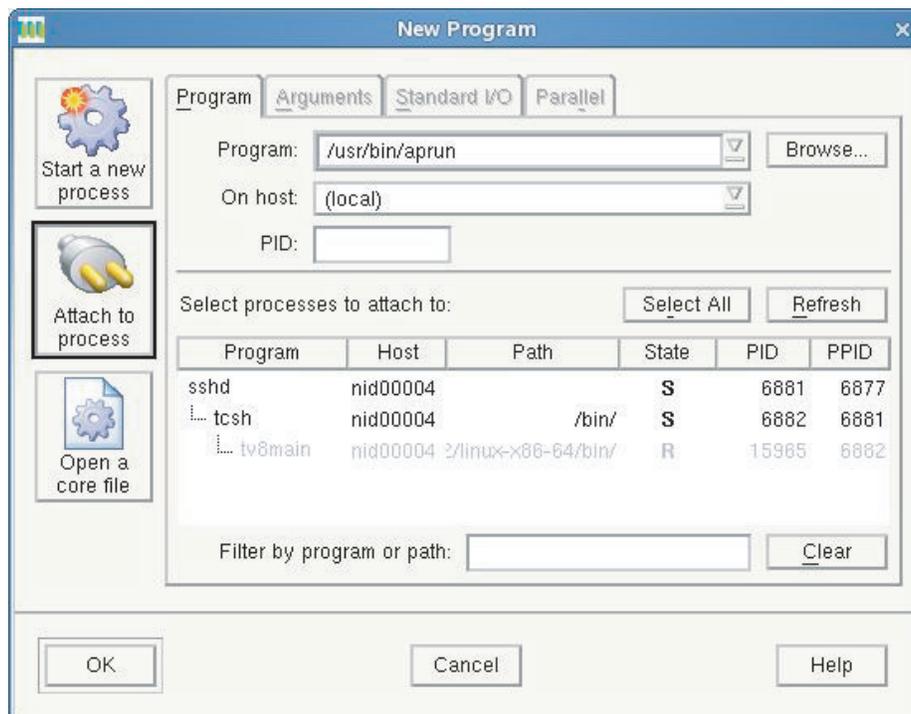


Figure 5. Attaching to a Running Process

3. Select the instance of `aprun` you want, and click **OK**. TotalView displays a Process Window showing both `aprun` and the program threads that were launched using that instance of `aprun`.

#### 11.1.4 Using TotalView to Alter Standard I/O

To change the names of the files to which TotalView will write or from which TotalView will read:

1. Launch the program using the `totalview` command. Do not specify the `stdin` file at this time.

```
% totalview aprun -a -n pes executable
```

The TotalView Root and Process windows display.

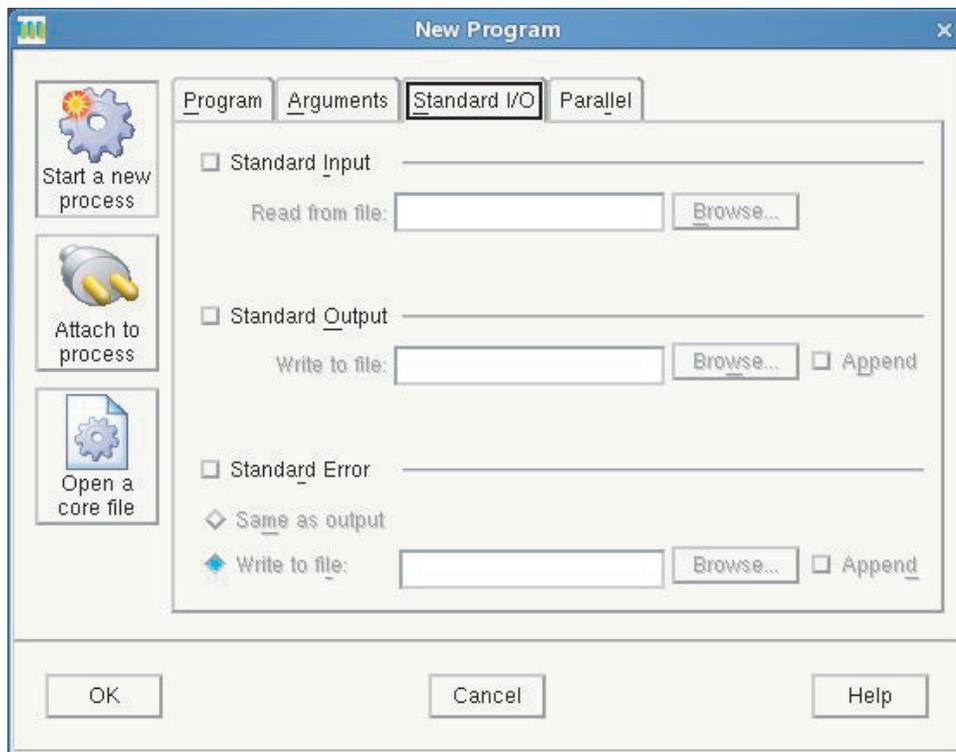


Figure 6. Altering Standard I/O

2. Select Startup Parameters in the Process window under the **Process** menu. The New Program window displays.
3. Select the **Standard I/O** tab. The **Standard Input**, **Standard Output**, and **Standard Error** fields are displayed.
4. Type the file name for **Standard Input**, **Standard Output**, or **Standard Error** field.
5. Type the desired file name and click the **OK** button.
6. On the main TotalView window, click the **Go** button to begin program execution.

#### 11.1.5 About the Limitations of TotalView on Cray XT Systems

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

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

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

## 11.2 Using the GNU Debugger

Cray XT systems support the GNU debugger, `gdb`, for debugging compute-node applications. The GNU debugger for CNL applications is `lgdb`, and the GNU debugger for Catamount applications is `xtgdb`.

### 11.2.1 Using the `lgdb` Debugger

(Deferred implementation) The `lgdb` command launches an application and `gdbserver` processes on CNL compute nodes for debugging purposes. Also, you can use `lgdb` to attach to an already running application by supplying the process ID (PID) of the `totalview` process that launched the application. When running `lgdb`, you should be in a directory that can be accessed from a remote node.

To use `lgdb`:

1. Load the `xt-lgdb` module.
2. Use the `-g` option on the `cc`, `CC`, or `ftn` command to generate debugging information.
3. Use the `lgdb` command to launch the debugger.

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

### 11.2.2 Using the `xtgdb` Debugger

The `xtgdb` command launches a single-process application on a Catamount compute node for debugging purposes. Additionally, you can use `xtgdb` to attach to an already running application by supplying the process ID and rank of the application.

To use `xtgdb`:

1. Load the `xtgdb` module.
2. Use the `-g` option on the `cc`, `CC`, or `ftn` command to generate debugging information.
3. Use the `xtgdb` command to launch the debugger.

For an example showing how to use `xtgdb` to set breakpoints in a single-process job, see [Section 15.13, page 175](#).

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

### 11.3 Troubleshooting Catamount Application Failures

The `yod` utility provides rudimentary diagnostics for the subset of Catamount calls that perform remote procedure calls (RPCs) to `yod`. [Table 7](#) lists the calls that perform RPCs to `yod`.

Table 7. RPCs to `yod`

|                     |                            |                       |                      |                       |
|---------------------|----------------------------|-----------------------|----------------------|-----------------------|
| <code>chmod</code>  | <code>fstatfs</code>       | <code>mkdir</code>    | <code>rmdir</code>   | <code>symlink</code>  |
| <code>chown</code>  | <code>fsync</code>         | <code>open</code>     | <code>setegid</code> | <code>sync</code>     |
| <code>close</code>  | <code>ftruncate</code>     | <code>pread</code>    | <code>seteuid</code> | <code>truncate</code> |
| <code>exit</code>   | <code>getdirentries</code> | <code>pwrite</code>   | <code>setgid</code>  | <code>umask</code>    |
| <code>fchmod</code> | <code>link</code>          | <code>read</code>     | <code>setuid</code>  | <code>unlink</code>   |
| <code>fchown</code> | <code>lseek</code>         | <code>readlink</code> | <code>stat</code>    | <code>utimes</code>   |
| <code>fstat</code>  | <code>lstat</code>         | <code>rename</code>   | <code>statfs</code>  | <code>write</code>    |

System calls performed solely by Catamount do not show up in the diagnostic output.

There are two ways to enable this feature:

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

**Note:** In this context the term *strace* is a misnomer. The `yod` utility does not provide the UNIX-like `strace()` function. Enabling `strace` turns on diagnostic output generated by the RPC library, which `yod` uses to service the system calls in [Table 7](#). The I/O-related system calls are for non-parallel file systems.

The `yod` command can also provide trace reports about memory allocation and deallocation. The `-tracemalloc` option provides rudimentary diagnostics for `malloc()` and `free()` calls. This information can help you pinpoint memory leaks and determine if using the GNU `malloc` library would be beneficial. For more information about the GNU `malloc` library, see [Appendix B, page 193](#).



# Analyzing Performance [12]

---

This chapter describes the Cray XT performance analysis tools.

## 12.1 Using the Performance API (PAPI)

The Performance API (PAPI) is a standard API for accessing microprocessor registers 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 high-level interface for basic measurements
- A fully programmable, low-level interface for users with more sophisticated needs

PAPI supports multiplexing under CNL. Although it is also supported under Catamount, the long time slice (~1 second) for each set of independent counters makes it impractical to use except for very long running programs.

The `pat_build` utility does not allow you to instrument a program that is also using the PAPI interface directly or indirectly (via `libhwpc`).

To use PAPI, load the PAPI module:

```
% module load xt-papi
```

For more information about PAPI, see <http://icl.cs.utk.edu/papi/>.

### 12.1.1 Using the High-level PAPI Interface

The high-level interface provides the ability to start, stop, and read specific events, one at a time. For an example of a CNL application using the PAPI high-level interface, see [Section 14.15, page 148](#). For an example of a Catamount application using the PAPI high-level interface, see [Section 15.14, page 176](#).

### 12.1.2 Using the Low-level PAPI Interface

The low-level PAPI interface deals with hardware events in groups called *event sets*. An event set maps 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 an example of a CNL application using the PAPI low-level interface, see [Section 14.16, page 149](#). For an example of a Catamount application using the PAPI low-level interface, see [Section 15.15, page 178](#).

For information about constructing an event set, see the *PAPI User Guide* and the *PAPI Programmer's Reference* manual.

For a list of supported hardware counter presets from which to construct an event set, see [Appendix C, page 199](#).

## 12.2 Using the Cray Performance Analysis Tool (CrayPat)

The Cray Performance Analysis Tool (CrayPat) helps you analyze the performance of programs. To use it:

1. Load the `xt-craypat` module.

```
% module load xt-craypat
```

**Note:** You must load the `xt-craypat` module before building even the uninstrumented version of the application.

2. Compile and link your application.

**Note:** All executable programs created with an older version of the CrayPat module loaded must be relinked in order to be instrumented with the latest version of CrayPat. The `pat_build` utility will not instrument executable files linked when older versions of the CrayPat module were loaded.

3. Use the `pat_build` command to create an instrumented version of the

application, specifying the functions to be traced through options such as `-u` and `-g mpi`.

4. Set any relevant environment variables, such as:

- `setenv PAT_RT_HWPC 1` or `export PAT_RT_HWPC=1`, which specifies the first of the predefined sets of hardware counter events.
- `setenv PAT_RT_SUMMARY 0` or `export PAT_RT_SUMMARY=0`, which specifies a full-trace data file rather than a summary. Such a file can be very large but is needed to view behavior over time with Cray Apprentice2.
- `setenv PAT_RT_EXPFILDIR dir` or `export PAT_RT_EXPFILDIR= dir` to specify a directory into which the experiment data files will be written, instead of the current working directory.

If a single data file is written, its default root name is the name of the instrumented program followed by the plus sign (+), the process ID, the minus sign (-), the physical node the application started executing upon, and one or more key letters indicating the type of the experiment (such as `program1+pat+3820-671tdt`).

If there is a data file from each process, they are written into a subdirectory with that name. For a large number of processes, it may be necessary for `PAT_RT_EXPFILMAX` to be set to 0 or the number of processes and that `PAT_RT_EXPFILDIR` be set to a directory in a Lustre file system (if the instrumented program is not invoked in such a directory). The default for a multi-PE program is to write a single data file.

5. Execute the instrumented program.

6. Use `pat_report` on the resulting data file to generate a report. The default report is a sample by function, but alternative views can be specified through options such as:

- `-O calltree`
- `-O callers`
- `-O load_balance`

The `-s pe=...` option overrides the way that per-PE data is shown in default tables and in tables specified using the `-O` option. For details, see the `pat_report(1)` man page.

These steps are illustrated in the example CrayPat programs (see [Chapter 14, page 115](#) and [Chapter 15, page 157](#)). For more information, see the man pages and the interactive `pat_help` utility.

For more information about using CrayPat, see the *Using Cray Performance Analysis Tools* guide and the `intro_craypat(1)` man page and run the `pat_help` utility. For more information about PAPI HWPC, see [Appendix C, page 199](#), the `hwpc(3)` man page, and the PAPI website at <http://icl.cs.utk.edu/papi/>.

### 12.2.1 Running Tracing and Sampling Experiments

CrayPat supports two types of experiments: tracing and sampling.

Tracing counts an event, such as the number of times an MPI call is executed. When tracing experiments are done, selected function entry points are traced and produce a data record in the run time experiment data file if the function is executed. Use the `-g tracegroup` option to instrument your program to trace all function entry point references belonging to a `tracegroup` value:

|                      |                                                                              |
|----------------------|------------------------------------------------------------------------------|
| <code>biolibs</code> | Cray Bioinformatics library routines                                         |
| <code>blas</code>    | Basic Linear Algebra Subprograms                                             |
| <code>heap</code>    | Dynamic heap                                                                 |
| <code>io</code>      | Includes <code>stdio</code> and <code>sysio</code> groups                    |
| <code>lapack</code>  | Linear Algebra PACKage                                                       |
| <code>math</code>    | ANSI math                                                                    |
| <code>mpi</code>     | MPI                                                                          |
| <code>omp</code>     | OpenMP API (not supported on Catamount)                                      |
| <code>omp-rtl</code> | OpenMP run time library (not supported on Catamount)                         |
| <code>pthread</code> | POSIX threads (not supported on Catamount)                                   |
| <code>shmem</code>   | SHMEM                                                                        |
| <code>stdio</code>   | All library functions that accept or return the <code>FILE*</code> construct |
| <code>sysio</code>   | I/O system calls                                                             |
| <code>system</code>  | System calls                                                                 |

**Note:** Only function calls at global scope can be traced. Function calls that are written in assembly language, that are inlined, or that are local to a translation unit cannot be traced.

Sampling experiments capture values from the call stack or the program counter at specified intervals or when a specified counter overflows. (Sampling experiments are also referred to as asynchronous experiments).

Supported sampling functions are:

- `samp_pc_time`, which samples the program counter at a given time interval. This returns the total program time and the absolute and relative times each program counter was recorded.
- `samp_pc_ovfl`, which samples the program counter at a given overflow of a hardware performance counter.
- `samp_cs_time`, which samples the call stack at a given time interval and returns the total program time and the absolute and relative times each call stack counter was recorded (otherwise identical to the `samp_pc_time` experiment).
- `samp_cs_ovfl`, which samples the call stack at a given overflow of a hardware performance counter (otherwise identical to the `samp_pc_ovfl` experiment).
- `samp_ru_time`, which samples system resources at a given time interval (otherwise identical to the `samp_pc_time` experiment).
- `samp_ru_ovfl`, which samples system resources at a given overflow of a hardware performance counter (otherwise identical to the `samp_pc_ovfl` experiment.)
- `samp_heap_time`, which samples dynamic heap memory management statistics at a given time interval (otherwise identical to the `samp_pc_time` experiment).
- `samp_heap_ovfl`, which samples dynamic heap memory management statistics at a given overflow of a hardware performance counter (otherwise identical to the `samp_pc_ovfl` experiment).

**Note:** Hardware counter information should not be collected during sampling by overflow. Cray recommends that you use sampling to obtain a profile and then trace the functions of interest to obtain hardware counter information for them.

## 12.3 Visualizing Performance Data

The Cray XT performance data visualization tool is Cray Apprentice2. You can run Cray Apprentice2 on a Cray XT system or Cray Apprentice2 Desktop on a standalone Linux machine. After you have used `pat_build` to instrument a program for a performance analysis experiment, executed the instrumented program, and used `pat_report` to convert the resulting data file to a Cray Apprentice2 data format, you can use Cray Apprentice2 to explore the experiment data file and generate a variety of interactive graphical reports.

To run Cray Apprentice2, load the Cray Apprentice2 module:

```
% module load apprentice2
```

Then use the `app2` command to launch Cray Apprentice2

```
% app2 [--limit tag_count | --limit_per_pe tag_count] [data_files]
```

To create a graphical representation of a CrayPat report, use the `pat_report -f` option to generate a report in `ap2` format.

For an example showing how to use Cray Apprentice2, see [Section 14.18, page 156](#).

For more information about using Cray Apprentice2, see the Cray Apprentice2 online help system and the `app2(1)` and `pat_report(1)` man pages.

# Optimizing Applications [13]

---

## 13.1 Using Compiler Optimization Options

After you have compiled and debugged your code and analyzed its performance, you can use a number of techniques to optimize performance. For details about compiler optimization and optimization reporting options, see the *PGI User's Guide*, the *Using the GNU Compiler Collection (GCC)* manual, or the *PathScale Compiler Suite User 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 through compiler driver options or to selected portions of code through the use of directives or pragmas. 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.

Source code of `matrix_multiply.f90`:

```
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
```

PGI Fortran compiler command:

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

**Optimization report:**

```
mxm:
 5, Interchange produces reordered loop nest: 7, 5, 9
 9, Generated 3 alternate loops for the inner loop
 Generated vector sse code for inner loop
 Generated 2 prefetch instructions for this loop
 Generated vector sse code for inner loop
 Generated 2 prefetch instructions for this loop
 Generated vector sse code for inner loop
 Generated 2 prefetch instructions for this loop
 Generated vector sse code for inner loop
 Generated 2 prefetch instructions for this loop
```

## 13.2 Using `aprun` Memory Affinity Options

On Cray XT5 systems, each compute node has local-NUMA-node memory and remote-NUMA-node memory. Remote-NUMA-node memory references, such as a NUMA node 0 PE accessing NUMA node 1 memory, can adversely affect performance. To give you run time controls that may optimize memory references, Cray has added `aprun` memory affinity options.

Applications can use one or both NUMA nodes of a Cray XT5 compute node. If an application is placed using one NUMA node, the other NUMA node is not used. In this case, the application processes are restricted to using local-NUMA-node memory. This memory usage policy is enforced by running the application processes within a *cpuset*. A *cpuset* is a process container that controls memory and CPU usage.

When an application is placed using both NUMA nodes, the *cpuset* includes all node memory and all CPUs. In this case, the application processes allocate local-NUMA-node memory first. If insufficient free local-NUMA-node memory is available, the allocation may be satisfied using remote-NUMA-node memory. In other words, if there is not enough NUMA node 0 memory, the allocation may be satisfied using NUMA node 1 memory. The one exception is the `-ss` (strict memory containment) option. For this option, memory accesses are restricted to local-NUMA-node memory even if both NUMA nodes are available to the application.

The `aprun` memory affinity options are:

- `-S pes_per_numa_node`
- `-sn numa_nodes_per_node`
- `-sl list_of_numa_nodes`
- `-ss`

For details, see [Section 7.1, page 61](#).

You can use these `aprun` options for each element of an MPMD application and can vary them with each MPMD element.

Only Cray XT5 compute nodes are considered for the application placement if any of the following are true:

- The `-sn` value is 2.
- The `-sl` list has more than one entry.
- The `-sl` list is NUMA node 1 (Cray XT3 and Cray XT4 systems have single-NUMA-node compute nodes, defined as NUMA node 0).
- The `-S` value along with a `-N` value requires two NUMA nodes (such as `-N 4 -S 2`).

You can use the `cselect coremask.eq.255` command to get a list of Cray XT5 compute nodes and the `aprun -L` or `qsub -lmpnodes` option to specify that list or a subset of the list. For additional information, see the `aprun(1)`, `cselect(1)`, and `qsub(1)` man pages.

### 13.3 Using `aprun` CPU Affinity Optimizations

CNL can dynamically distribute work by allowing PEs and threads to migrate from one CPU to another within a node. In some cases, moving processes from CPU to CPU increases cache misses and translation lookaside buffer (TLB) misses and therefore reduces performance. Also, there may be cases where an application runs faster by avoiding or targeting a particular CPU. The `aprun` CPU affinity options let you bind a process to a particular CPU or the CPUs on a NUMA node. These options apply to all Cray XT multicore compute nodes.

Applications are assigned to a cuset and can run only on the CPUs specified by the cuset. Also, applications can allocate memory only on memory defined by the cuset. A cuset can be a compute node (default) or, for Cray XT5 systems, a NUMA node.

The CPU affinity options are:

- `-cc cpu-list | keyword`
- (Deferred implementation) `-cp cpu_placement_file_name`

For details, see [Section 7.1, page 61](#).

These `aprun` options can be used for each element of an MPMD application and can vary with each MPMD element.

Cray XT3 and Cray XT4 systems have single-NUMA-node compute nodes. Their default CPU affinity is the same as for Cray XT5 systems — `aprun -cc cpu`.

## 13.4 Optimizing Process Placement on Multicore Nodes

Because multicore systems can run more tasks simultaneously, overall system performance can increase. The trade-offs are that each core has less local memory (because it is shared by the cores) and less system interconnection bandwidth (which is also shared).

### 13.4.1 Optimizing MPI and SHMEM Applications Running under CNL

Processes are placed in packed rank-sequential order, starting with the first node. So, for a 100-core, 50-node job running on dual-core nodes, the layout of ranks on cores is:

|      | Node 1 |   | Node 2 |   | Node 3 |   | ... | Node 50 |    |
|------|--------|---|--------|---|--------|---|-----|---------|----|
| Core | 0      | 1 | 0      | 1 | 0      | 1 | ... | 0       | 1  |
| Rank | 0      | 1 | 2      | 3 | 4      | 5 | ... | 98      | 99 |

**Note:** You can use the `yod` placement method (rank-sequential order) in CNL applications instead by setting `MPICH_RANK_REORDER_METHOD` to 0.

For CNL applications, MPI supports multiple interconnect device drivers for a single MPI job. This allows each process (rank) of an MPI job to create the most optimal messaging path to every other process in the job, based on the topology of the given ranks.

Two device drivers are supported: the SMP driver and the Portals device driver. The SMP device driver is based on shared memory and is used for communication between ranks that share a node. The Portals device driver is used for communication between ranks that span nodes.

To attain the fastest possible run time, try running your program on only one core of each node. (In this case, the other cores are allocated to your job but idle.) This allows each process to have full access to the system interconnection network.

For example, the command:

```
% aprun -n 64 -N 1 ./prog1
```

launches `prog1` on one core of each of 64 multicore nodes.

### 13.4.2 Optimizing MPI and SHMEM Applications Running under Catamount

By default, processes are placed in rank-sequential order on dual-core nodes, first on the master core (core 0) on each node and then on the subordinate core (core 1) on each node. So, for a 100-core, 50-node job, the layout of ranks on cores is:

|      | Node 1 |    | Node 2 |    | Node 3 |    | ... | Node 50 |    |
|------|--------|----|--------|----|--------|----|-----|---------|----|
| Core | 0      | 1  | 0      | 1  | 0      | 1  | ... | 0       | 1  |
| Rank | 0      | 50 | 1      | 51 | 2      | 52 | ... | 49      | 99 |

Latency times for data transfers between parallel processes can vary according to the type of process-to-core placement: master-to-master, subordinate-to-subordinate, master-to-subordinate on different nodes, and master-to-subordinate on the same node. Master-to-master transfers have the shortest latency; subordinate-to-subordinate transfers have the longest latency.

For Catamount applications, MPI and SHMEM are not aware of the processor placement topology. As a result, some applications may experience performance degradation.

To attain the fastest possible run time, try running your program on the master core of each allocated node. The subordinate cores are allocated to your job but idle.

For example, the command:

```
% yod -sz 64 prog1
```

launches `prog1` on the master core of each of 64 nodes.

The `MPICH_RANK_REORDER_METHOD` environment variable allows you to override the default rank ordering scheme and use an SMP-style placement, a folded-rank placement, or a custom rank placement. See the `intro_mpi(3)` man page for details.

# Example CNL Applications [14]

---

This chapter gives examples showing how to compile, link, and run CNL applications.

Verify that your work area is in a Lustre-mounted directory. Then use the `module list` command to verify that the correct modules are loaded. Whenever you compile and link applications to be run under CNL, you need to have the `xtpe-target-cnl` module loaded. Each following example lists the modules that have to be loaded.

## 14.1 Running a Basic Application under CNL

This example shows how to compile program `simple_cnl.c` and launch the executable.

Modules required:

```
xtpe-target-cnl
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

Create a C program, `simple_cnl.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:

```
% cc -o simple_cnl simple_cnl.c
```

Run the program:

```
% aprun -n 6 ./simple_cnl
hello from pe 0 of 6
hello from pe 5 of 6
hello from pe 4 of 6
hello from pe 3 of 6
hello from pe 2 of 6
hello from pe 1 of 6
Application 135891 resources: utime 0, stime 0
```

## 14.2 Running an MPI Application under CNL

This example shows how to compile, link, and run an MPI program. The MPI program distributes the work represented in a reduction loop, prints the subtotal for each PE, combines the results from the PEs, and prints the total.

Modules required:

```
xtpe-target-cnl
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

Create a Fortran program, `mpi_cnl.f90`:

```

program reduce
include "mpif.h"

integer n, nres, ierr

call MPI_INIT (ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD,mype,ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD,npes,ierr)

nres = 0
n = 0

do i=mype,100,npes
 n = n + i
enddo

print *, 'My PE:', mype, ' My part:',n

call MPI_REDUCE (n,nres,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)

if (mype == 0) print *,' PE:',mype,'Total is:',nres

call MPI_FINALIZE (ierr)

end

```

Compile `mpi_cnl.f90`:

```
% ftn -o mpi_cnl mpi_cnl.f90
```

Run program `mpi_cnl`:

```

% aprun -n 6 ./mpi_cnl | sort
 PE: 0 Total is: 5050
My PE: 0 My part: 816
My PE: 1 My part: 833
My PE: 2 My part: 850
My PE: 3 My part: 867
My PE: 4 My part: 884
My PE: 5 My part: 800
Application 3016865 resources: utime 0, stime 0

```

If desired, you could use this C version of the program:

```
/* program reduce */

#include <stdio.h>
#include "mpi.h"

int main (int argc, char *argv[])
{
 int i, sum, mype, npes, nres, ret;
 ret = MPI_Init (&argc, &argv);
 ret = MPI_Comm_size (MPI_COMM_WORLD, &npes);
 ret = MPI_Comm_rank (MPI_COMM_WORLD, &mype);
 nres = 0;
 sum = 0;
 for (i = mype; i <=100; i += npes) {
 sum = sum + i;
 }

 (void) printf ("My PE:%d My part:%d\n",mype, sum);
 ret = MPI_Reduce (&sum,&nres,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD);
 if (mype == 0)
 {
 (void) printf ("PE:%d Total is:%d\n",mype, nres);
 }
 ret = MPI_Finalize ();
}
```

### 14.3 Using the Cray `shmem_put` Function under CNL

This example shows how to use the `shmem_put64()` function to copy a contiguous data object from the local PE to a contiguous data object on a different PE.

Modules required:

`xtpe-target-cn1`

and one of the following:

`PrgEnv-pgi`

`PrgEnv-gnu`

`PrgEnv-pathscale`

Source code of C program (shmem\_put\_cnl.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 XT. 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 `shmem_put_cnl.c` and create executable `shmem_put_cnl`:

```
% cc -o shmem_put_cnl shmem_put_cnl.c
```

Run `shmem_put_cnl`:

```
% aprun -n 6 ./shmem_put_cnl
PE 0: Test passed.
PE 5: Test passed.
PE 4: Test passed.
PE 3: Test passed.
PE 2: Test passed.
PE 1: Test passed.
Application 137008 exit codes: 128
Application 137008 resources: utime 0, stime 0
```

## 14.4 Using the Cray `shmem_get` Function under CNL

This example shows how to use the `shmem_get()` function to copy a contiguous data object from a different PE to a contiguous data object on the local PE.

**Modules required:**

```
xtpe-target-cn1
```

**and one of the following:**

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

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

**Source code of Fortran program (shmem\_get\_cn1.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 shmem\_get\_cn1.f90 and create executable shmem\_get\_cn1:**

```
% ftn -o shmem_get_cn1 shmem_get_cn1.f90
```

Run shmem2:

```
% aprun -n 6 ./shmem_get_cn1
PE 0 computedsum= 15.00000
PE 5 computedsum= 15.00000
PE 4 computedsum= 15.00000
PE 3 computedsum= 15.00000
PE 2 computedsum= 15.00000
PE 1 computedsum= 15.00000
Application 137031 resources: utime 0, stime 0
```

## 14.5 Running a UPC Application under CNL

This example shows how to compile and run a C program that includes Unified Parallel C (UPC) functions.

Modules required:

```
xtpe-target-cn1
xt-upc
```

and one of the following:

```
PrgEnv-upc-pgi
PrgEnv-upc-gnu
```

**Note:** UPC source files must have the `upc` extension.

Source code of program `upc_cn1.upc`:

```
#include <upc.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
 int i;
 for (i = 0; i < THREADS; ++i)
 {
 upc_barrier;
 if (i == MYTHREAD)
 printf ("Hello world from thread: %d\n", MYTHREAD);
 }
 return 0;
}
```

You can use the Berkeley UPC translator or the Intrepid GCCUPC compiler to compile your program.

Compile `upc_cnl.upc` using the Berkeley UPC translator:

```
% upcc -o upc_cnl upc_cnl.upc
```

Run `upc_cnl`:

```
% upcrun -n 2 ./upc_cnl
UPCR: UPC thread 0 of 2 on nid00012 (process 0 of 2, pid=1438)
UPCR: UPC thread 1 of 2 on nid00013 (process 1 of 2, pid=1402)
Hello world from thread: 0
Hello world from thread: 1
Application 170008 resources: utime 0, stime 0
```

To compile `upc_cnl.upc` using the Intrepid GCCUPC compiler, load the `PrgEnv-upc-gnu` module and include the `-gccupc` option on the compiler command line:

```
% module swap PrgEnv-upc-pgi PrgEnv-upc-gnu
% upcc -o upc_cnl -gccupc upc_cnl.upc
% upcrun -n 2 ./upc_cnl
UPCR: UPC thread 0 of 2 on nid00012 (process 0 of 2, pid=1443)
UPCR: UPC thread 1 of 2 on nid00013 (process 1 of 2, pid=1407)
Hello world from thread: 0
Hello world from thread: 1
Application 170077 resources: utime 0, stime 0
```

## 14.6 Running a PETSc Application under CNL

This example (Copyright 1995-2004 University of Chicago) shows how to use PETSc functions to solve a linear system of partial differential equations.

**Note:** There are many ways to use the PETSc solvers. This example is intended to show the basics of compiling and running a PETSc program on a Cray XT system. It presents one simple approach and may not be the best template to use in writing user code. For issues that are not specific to Cray XT systems, you can get technical support through `petsc-users@mcs.anl.gov`.

The source code for this example includes a comment about the use of the `mpiexec` command to launch the executable. Use `aprun` instead.

**Modules required:**

xtpe-target-cn1  
 petsc

**and one of the following:**

PrgEnv-pgi  
 PrgEnv-gnu  
 PrgEnv-pathscale

**Source code of program ex2f.F:**

```

!
! Description: Solves a linear system in parallel with KSP (Fortran code).
! Also shows how to set a user-defined monitoring routine.
!
! Program usage: mpiexec -np ex2f [-help] [all PETSc options]
!
!/*T
! Concepts: KSP^basic parallel example
! Concepts: KSP^setting a user-defined monitoring routine
! Processors: n
!T*/
!
! -----
!
! program main
! implicit none
!
! -----
!
! Include files
! -----
!
! This program uses CPP for preprocessing, as indicated by the use of
! PETSc include files in the directory petsc/include/finclude. This
! convention enables use of the CPP preprocessor, which allows the use
! of the #include statements that define PETSc objects and variables.
!
! Use of the conventional Fortran include statements is also supported
! In this case, the PETSc include files are located in the directory
! petsc/include/foldinclude.
!
! Since one must be very careful to include each file no more than once
! in a Fortran routine, application programmers must explicitly list
! each file needed for the various PETSc components within their

```

```

! program (unlike the C/C++ interface).
!
! See the Fortran section of the PETSc users manual for details.
!
! The following include statements are required for KSP Fortran programs:
! Petsc.h - base PETSc routines
! PetscVec.h - vectors
! PetscMat.h - matrices
! PetscPC.h - preconditioners
! PetscKSP.h - Krylov subspace methods
! Include the following to use PETSc random numbers:
! PetscSys.h - system routines
! Additional include statements may be needed if using additional
! PETSc routines in a Fortran program, e.g.,
! PetscViewer.h - viewers
! PetscIS.h - index sets
!
#include "include/finclude/petsc.h"
#include "include/finclude/petscvec.h"
#include "include/finclude/petscmat.h"
#include "include/finclude/petscpc.h"
#include "include/finclude/petscksp.h"
#include "include/finclude/petscsys.h"
!
! - - - - -
!
! Variable declarations
!
! - - - - -
!
! Variables:
! ksp - linear solver context
! ksp - Krylov subspace method context
! pc - preconditioner context
! x, b, u - approx solution, right-hand-side, exact solution vectors
! A - matrix that defines linear system
! its - iterations for convergence
! norm - norm of error in solution
! rctx - random number generator context
!
! Note that vectors are declared as PETSc "Vec" objects. These vectors
! are mathematical objects that contain more than just an array of
! double precision numbers. I.e., vectors in PETSc are not just
! double precision x(*).
! However, local vector data can be easily accessed via VecGetArray().

```

```
! See the Fortran section of the PETSc users manual for details.
!
double precision norm
PetscInt i,j,II,JJ,m,n,its
PetscInt Istart,Iend,ione
PetscErrorCode ierr
PetscMPIInt rank,size
PetscTruth flg
PetscScalar v,one,neg_one
Vec x,b,u
Mat A
KSP ksp
PetscRandom rctx

! These variables are not currently used.
! PC pc
! PCType ptype
! double precision tol

! Note: Any user-defined Fortran routines (such as MyKSPMonitor)
! MUST be declared as external.

external MyKSPMonitor,MyKSPConverged

! -----
! Beginning of program
! -----

call PetscInitialize(PETSC_NULL_CHARACTER,ierr)
m = 3
n = 3
one = 1.0
neg_one = -1.0
ione = 1
call PetscOptionsGetInt(PETSC_NULL_CHARACTER,'-m',m,flg,ierr)
call PetscOptionsGetInt(PETSC_NULL_CHARACTER,'-n',n,flg,ierr)
call MPI_Comm_rank(PETSC_COMM_WORLD,rank,ierr)
call MPI_Comm_size(PETSC_COMM_WORLD,size,ierr)

! -----
! Compute the matrix and right-hand-side vector that define
! the linear system, Ax = b.
```

```

! -----
! Create parallel matrix, specifying only its global dimensions.
! When using MatCreate(), the matrix format can be specified at
! runtime. Also, the parallel partitioning of the matrix is
! determined by PETSc at runtime.

 call MatCreate(PETSC_COMM_WORLD,A,ierr)
 call MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,m*n,m*n,ierr)
 call MatSetFromOptions(A,ierr)

! Currently, all PETSc parallel matrix formats are partitioned by
! contiguous chunks of rows across the processors. Determine which
! rows of the matrix are locally owned.

 call MatGetOwnershipRange(A,Istart,Iend,ierr)

! Set matrix elements for the 2-D, five-point stencil in parallel.
! - Each processor needs to insert only elements that it owns
! locally (but any non-local elements will be sent to the
! appropriate processor during matrix assembly).
! - Always specify global row and columns of matrix entries.
! - Note that MatSetValues() uses 0-based row and column numbers
! in Fortran as well as in C.

! Note: this uses the less common natural ordering that orders first
! all the unknowns for $x = h$ then for $x = 2h$ etc; Hence you see $JH = II +- n$
! instead of $JJ = II +- m$ as you might expect. The more standard ordering
! would first do all variables for $y = h$, then $y = 2h$ etc.

do 10, II=Istart,Iend-1
 v = -1.0
 i = II/n
 j = II - i*n
 if (i.gt.0) then
 JJ = II - n
 call MatSetValues(A,ione,II,ione,JJ,v,INSERT_VALUES,ierr)
 endif
 if (i.lt.m-1) then
 JJ = II + n
 call MatSetValues(A,ione,II,ione,JJ,v,INSERT_VALUES,ierr)
 endif
 if (j.gt.0) then

```

```
 JJ = II - 1
 call MatSetValues(A,ione,II,ione,JJ,v,INSERT_VALUES,ierr)
 endif
 if (j.lt.n-1) then
 JJ = II + 1
 call MatSetValues(A,ione,II,ione,JJ,v,INSERT_VALUES,ierr)
 endif
 v = 4.0
 call MatSetValues(A,ione,II,ione,II,v,INSERT_VALUES,ierr)
10 continue

! Assemble matrix, using the 2-step process:
! MatAssemblyBegin(), MatAssemblyEnd()
! Computations can be done while messages are in transition,
! by placing code between these two statements.

 call MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY,ierr)
 call MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY,ierr)

! Create parallel vectors.
! - Here, the parallel partitioning of the vector is determined by
! PETSc at runtime. We could also specify the local dimensions
! if desired -- or use the more general routine VecCreate().
! - When solving a linear system, the vectors and matrices MUST
! be partitioned accordingly. PETSc automatically generates
! appropriately partitioned matrices and vectors when MatCreate()
! and VecCreate() are used with the same communicator.
! - Note: We form 1 vector from scratch and then duplicate as needed.

 call VecCreateMPI(PETSC_COMM_WORLD,PETSC_DECIDE,m*n,u,ierr)
 call VecSetFromOptions(u,ierr)
 call VecDuplicate(u,b,ierr)
 call VecDuplicate(b,x,ierr)

! Set exact solution; then compute right-hand-side vector.
! By default we use an exact solution of a vector with all
! elements of 1.0; Alternatively, using the runtime option
! -random_sol forms a solution vector with random components.

 call PetscOptionsHasName(PETSC_NULL_CHARACTER, &
& "-random_exact_sol",flg,ierr)
 if (flg .eq. 1) then
 call PetscRandomCreate(PETSC_COMM_WORLD,rctx,ierr)
```

```

 call PetscRandomSetFromOptions(rctx,ierr)
 call VecSetRandom(u,rctx,ierr)
 call PetscRandomDestroy(rctx,ierr)
 else
 call VecSet(u,one,ierr)
 endif
 call MatMult(A,u,b,ierr)

! View the exact solution vector if desired

 call PetscOptionsHasName(PETSC_NULL_CHARACTER, &
& "-view_exact_sol",flg,ierr)
 if (flg .eq. 1) then
 call VecView(u,PETSC_VIEWER_STDOUT_WORLD,ierr)
 endif

! -----
! Create the linear solver and set various options
! -----

! Create linear solver context

 call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)

! Set operators. Here the matrix that defines the linear system
! also serves as the preconditioning matrix.

 call KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN,ierr)

! Set linear solver defaults for this problem (optional).
! - By extracting the KSP and PC contexts from the KSP context,
! we can then directly call any KSP and PC routines
! to set various options.
! - The following four statements are optional; all of these
! parameters could alternatively be specified at runtime via
! KSPSetFromOptions(). All of these defaults can be
! overridden at runtime, as indicated below.

! We comment out this section of code since the Jacobi
! preconditioner is not a good general default.

! call KSPGetPC(ksp,pc,ierr)
! ptype = PCJACOBI

```

```
! call PCSetType(pc,ptype,ierr)
! tol = 1.e-7
! call KSPSetTolerances(ksp,tol,PETSC_DEFAULT_DOUBLE_PRECISION,
! & PETSC_DEFAULT_DOUBLE_PRECISION,PETSC_DEFAULT_INTEGER,ierr)

! Set user-defined monitoring routine if desired

 call PetscOptionsHasName(PETSC_NULL_CHARACTER,'-my_ksp_monitor', &
& flg,ierr)
 if (flg .eq. 1) then
 call KSPMonitorSet(ksp,MyKSPMonitor,PETSC_NULL_OBJECT, &
& PETSC_NULL_FUNCTION,ierr)
 endif

! Set runtime options, e.g.,
! -ksp_type <type> -pc_type <type> -ksp_monitor -ksp_rtol
! These options will override those specified above as long as
! KSPSetFromOptions() is called after any other customization
! routines.

 call KSPSetFromOptions(ksp,ierr)

! Set convergence test routine if desired

 call PetscOptionsHasName(PETSC_NULL_CHARACTER, &
& '-my_ksp_convergence',flg,ierr)
 if (flg .eq. 1) then
 call KSPSetConvergenceTest(ksp,MyKSPConverged, &
& PETSC_NULL_OBJECT,ierr)
 endif

!
! -----
! Solve the linear system
! -----

 call KSPSolve(ksp,b,x,ierr)

! -----
! Check solution and clean up
! -----

! Check the error
```

```

call VecAXPY(x,neg_one,u,ierr)
call VecNorm(x,NORM_2,norm,ierr)
call KSPGetIterationNumber(ksp,its,ierr)
if (rank .eq. 0) then
 if (norm .gt. 1.e-12) then
 write(6,100) norm,its
 else
 write(6,110) its
 endif
endif
100 format('Norm of error ',e10.4,' iterations ',i5)
110 format('Norm of error < 1.e-12,iterations ',i5)

! Free work space. All PETSc objects should be destroyed when they
! are no longer needed.

call KSPDestroy(ksp,ierr)
call VecDestroy(u,ierr)
call VecDestroy(x,ierr)
call VecDestroy(b,ierr)
call MatDestroy(A,ierr)

! Always call PetscFinalize() before exiting a program. This routine
! - finalizes the PETSc libraries as well as MPI
! - provides summary and diagnostic information if certain runtime
! options are chosen (e.g., -log_summary). See PetscFinalize()
! manpage for more information.

call PetscFinalize(ierr)
end

! -----
!
! MyKSPMonitor - This is a user-defined routine for monitoring
! the KSP iterative solvers.
!
! Input Parameters:
! ksp - iterative context
! n - iteration number
! rnorm - 2-norm (preconditioned) residual value (may be estimated)
! dummy - optional user-defined monitor context (unused here)
!

```

```
subroutine MyKSPMonitor(ksp,n,rnorm,dummy,ierr)

implicit none

#include "include/finclude/petsc.h"
#include "include/finclude/petscvec.h"
#include "include/finclude/petscksp.h"

KSP ksp
Vec x
PetscErrorCode ierr
PetscInt n,dummy
PetscMPIInt rank
double precision rnorm

! Build the solution vector

call KSPBuildSolution(ksp,PETSC_NULL_OBJECT,x,ierr)

! Write the solution vector and residual norm to stdout
! - Note that the parallel viewer PETSC_VIEWER_STDOUT_WORLD
! handles data from multiple processors so that the
! output is not jumbled.

call MPI_Comm_rank(PETSC_COMM_WORLD,rank,ierr)
if (rank .eq. 0) write(6,100) n
call VecView(x,PETSC_VIEWER_STDOUT_WORLD,ierr)
if (rank .eq. 0) write(6,200) n,rnorm

100 format('iteration ',i5,' solution vector:')
200 format('iteration ',i5,' residual norm ',e10.4)
ierr = 0
end

! -----
!
! MyKSPConverged - This is a user-defined routine for testing
! convergence of the KSP iterative solvers.
!
! Input Parameters:
! ksp - iterative context
! n - iteration number
! rnorm - 2-norm (preconditioned) residual value (may be estimated)
```

```
! dummy - optional user-defined monitor context (unused here)
!
 subroutine MyKSPConverged(ksp,n,rnorm,flag,dummy,ierr)

 implicit none

#include "include/finclude/petsc.h"
#include "include/finclude/petscvec.h"
#include "include/finclude/petscksp.h"

 KSP ksp
 PetscErrorCode ierr
 PetscInt n,dummy
 KSPConvergedReason flag
 double precision rnorm

 if (rnorm .le. .05) then
 flag = 1
 else
 flag = 0
 endif
 ierr = 0

end
```

Use makefile.F:

```
.SUFFIXES: .mod .o .F

Compilers, linkers and flags.

FC = ftn
LINKER = ftn
FCFLAGS =
LINKLAGS =

Fortran optimization options.

FOPTFLAGS = -O3

.F.o:
$(FC) -c ${FOPTFLAGS} ${FCFLAGS} $*.F

all : ex2f
ex2f : ex2f.o
$(LINKER) -o $@ ex2f.o
```

Create and run executable `ex2f`, including the PETSc run time option `-mat_view` to display the nonzero values of the 9x9 matrix A:

```
% make -f makefile.F
% aprun -n 2 ./ex2f -mat_view
row 0: (0, 4) (1, -1) (3, -1)
row 1: (0, -1) (1, 4) (2, -1) (4, -1)
row 2: (1, -1) (2, 4) (5, -1)
row 3: (0, -1) (3, 4) (4, -1) (6, -1)
row 4: (1, -1) (3, -1) (4, 4) (5, -1) (7, -1)
row 5: (2, -1) (4, -1) (5, 4) (8, -1)
row 6: (3, -1) (6, 4) (7, -1)
row 7: (4, -1) (6, -1) (7, 4) (8, -1)
row 8: (5, -1) (7, -1) (8, 4)
row 0: (0, 0.25) (3, -1)
row 1: (1, 0.25) (2, -1)
row 2: (1, -0.25) (2, 0.266667) (3, -1)
row 3: (0, -0.25) (2, -0.266667) (3, 0.287081)
row 0: (0, 0.25) (1, -1) (3, -1)
row 1: (0, -0.25) (1, 0.266667) (2, -1) (4, -1)
row 2: (1, -0.266667) (2, 0.267857)
row 3: (0, -0.25) (3, 0.266667) (4, -1)
row 4: (1, -0.266667) (3, -0.266667) (4, 0.288462)
Norm of error < 1.e-12, iterations 7
Application 155514 resources: utime 0, stime 12
```

## 14.7 Running an OpenMP Application under CNL

This example shows how to compile and run an OpenMP/MPI application.

Modules required:

```
xtpe-target-cnl
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

**Note:** To compile an OpenMP program using a PGI or PathScale compiler, include `-mp` on the compiler driver command line. For a GCC compiler, include `-fopenmp`.

Source code of C program `omp_cn1.c`:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sched.h>
#include <mpi.h>
#include <omp.h>

/* Borrowed from util-linux-2.13-pre7/schedutils/taskset.c */
static char *cpuset_to_cstr(cpu_set_t *mask, char *str)
{
 char *ptr = str;
 int i, j, entry_made = 0;
 for (i = 0; i < CPU_SETSIZE; i++) {
 if (CPU_ISSET(i, mask)) {
 int run = 0;
 entry_made = 1;
 for (j = i + 1; j < CPU_SETSIZE; j++) {
 if (CPU_ISSET(j, mask)) run++;
 else break;
 }
 if (!run)
 sprintf(ptr, "%d,", i);
 else if (run == 1) {
 sprintf(ptr, "%d,%d,", i, i + 1);
 i++;
 } else {
 sprintf(ptr, "%d-%d,", i, i + run);
 i += run;
 }
 }
 while (*ptr != 0) ptr++;
 }
 ptr -= entry_made;
 *ptr = 0;
 return(str);
}

int main(int argc, char *argv[])
{
 int rank, thread;
```

```

cpu_set_t coremask;
char clbuf[7 * CPU_SETSIZE], hnbuf[64];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
memset(clbuf, 0, sizeof(clbuf));
memset(hnbuf, 0, sizeof(hnbuf));
(void)gethostname(hnbuf, sizeof(hnbuf));
#pragma omp parallel private(thread, coremask, clbuf)
{
thread = omp_get_thread_num();
(void)sched_getaffinity(0, sizeof(coremask), &coremask);
cpuset_to_cstr(&coremask, clbuf);
#pragma omp barrier
printf("Hello from rank %d, thread %d, on %s. (core affinity = %s)\n",
rank, thread, hnbuf, clbuf);
}
MPI_Finalize();
return(0);
}

```

Compile and link `omp_cnl.c`:

```
% cc -mp -o omp_cnl omp_cnl.c
```

Set the OpenMP environment variable equal to the number of threads in the team:

```
% setenv OMP_NUM_THREADS 2
```

Run program `omp_cnl`:

```

% aprun -n 4 -d 2 -q ./omp_cnl | sort
gar cnl/c> aprun -n 4 -d 2 -q ./xt5hi | sort
Hello from rank 0, thread 0, on nid00317. (core affinity = 0)
Hello from rank 0, thread 1, on nid00317. (core affinity = 1)
Hello from rank 1, thread 0, on nid00317. (core affinity = 2)
Hello from rank 1, thread 1, on nid00317. (core affinity = 3)
Hello from rank 2, thread 0, on nid00317. (core affinity = 4)
Hello from rank 2, thread 1, on nid00317. (core affinity = 5)
Hello from rank 3, thread 0, on nid00317. (core affinity = 6)
Hello from rank 3, thread 1, on nid00317. (core affinity = 7)

```

The `aprun` command created four instances of `omp_cnl`; each instance of `omp_cnl` spawned an additional thread. Each thread runs on a separate CPU.

If we run the same program but do not include the `-d 2` option, each CPU runs a PE and its thread:

```
% aprun -n 4 -q ./xt5hi | sort
Hello from rank 0, thread 0, on nid00317. (core affinity = 0)
Hello from rank 0, thread 1, on nid00317. (core affinity = 0)
Hello from rank 1, thread 0, on nid00317. (core affinity = 1)
Hello from rank 1, thread 1, on nid00317. (core affinity = 1)
Hello from rank 2, thread 0, on nid00317. (core affinity = 2)
Hello from rank 2, thread 1, on nid00317. (core affinity = 2)
Hello from rank 3, thread 0, on nid00317. (core affinity = 3)
Hello from rank 3, thread 1, on nid00317. (core affinity = 3)
```

## 14.8 Running a PBS Professional Interactive Job under CNL

This example shows how to compile and run an OpenMP/MPI application (see [Section 14.7, page 135](#)) on quad-core processors using a PBS Professional interactive job.

Modules required:

```
xtpe-target-cnl
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

Use the `cnsselect` command to get a list of quad-core processors:

```
% cnsselect coremask.eq.15
28-95,128-223,256-351,384-479,512-607,640-715
```

Initiate a PBS Professional interactive session:

```
% qsub -I -l mppwidth=4 -l mppdepth=2 -l mppnodes="28-30"
```

Set the OpenMP environment variable equal to the number of threads in the team:

```
% setenv OMP_NUM_THREADS 2
```

Run `omp_cnl`:

```
% aprun -n 4 -d 2 -L28-30 ./omp_cnl
Hello from rank 0 (thread 1) on nid00512
Hello from rank 1 (thread 0) on nid00512
Hello from rank 1 (thread 1) on nid00512
Hello from rank 0 (thread 0) on nid00512
Hello from rank 3 (thread 0) on nid00513
Hello from rank 2 (thread 1) on nid00513
Hello from rank 2 (thread 0) on nid00513
Hello from rank 3 (thread 1) on nid00513
```

## 14.9 Running a PBS Professional Job Script under CNL

In this example, a PBS Professional job script requests six PEs to run program `mpi_cnl`.

Modules required:

```
xtpe-target-cnl
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

**Note:** Do not load the `xt-pbs` module. Unload it if it has been loaded.

Create `pbs_script_cnl`:

```
#!/bin/bash
#
Define the destination of this job
as the queue named "workq":
#PBS -q workq
#PBS -l mppwidth=6
Tell PBS Pro to keep both standard output and
standard error on the execution host:
#PBS -k eo
cd /lus/nid0008/user1
aprun -n 6 ./mpi_cnl
exit 0
```

Set permissions to executable:

```
% chmod +x pbs_script_cnl
```

Submit the job:

```
% qsub pbs_script_cnl
```

The `qsub` command produces a batch job log file with output from `mpi_cnl` (see [Section 14.2, page 116](#)). The job output is in a `pbs_script_cnl.onnnnn` file.

```
% cat pbs_script_cnl.o238830 | sort
Application 848571 resources: utime 0, stime 0
My PE: 0 My part: 816
My PE: 1 My part: 833
My PE: 2 My part: 850
My PE: 3 My part: 867
My PE: 4 My part: 884
My PE: 5 My part: 800
PE: 0 Total is: 5050
```

## 14.10 Running Multiple Sequential Applications under CNL

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

Modules required:

```
xtpe-target-cnl
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

**Note:** Do not load the `xt-pbs` module. Unload it if it has been loaded.

Create script `mult_seq_cnl`:

```
#!/bin/bash
```

```

#
Define the destination of this job
as the queue named "workq":
#PBS -q workq
#PBS -l mppwidth=6
Tell PBS Pro to keep both standard output and
standard error on the execution host:
#PBS -k eo
cd /lus/nid000015/user1
aprun -n 2 -q ./simple_cnl
aprun -n 3 -q ./mpi_cnl
aprun -n 6 -q ./shmem_put_cnl
aprun -n 6 -q ./shmem_get_cnl
exit 0

```

The script launches applications `simple_cnl` (see [Section 14.1, page 115](#)), `mpi_cnl` (see [Section 14.2, page 116](#)), `shmem_put_cnl` (see [Section 14.3, page 118](#)), and `shmem_get_cnl` (see [Section 14.4, page 120](#)).

Set file permission to executable:

```
% chmod +x mult_seq_cnl
```

Run the script:

```
% qsub mult_seq_cnl
```

List the output:

```

% cat mult_seq_cnl.o465713
hello from pe 0 of 2
hello from pe 1 of 2
My PE: 0 My part: 1683
My PE: 1 My part: 1717
My PE: 2 My part: 1650
PE: 0 Total is: 5050
PE 0: Test passed.
PE 1: Test passed.
PE 2: Test passed.
PE 3: Test passed.
PE 4: Test passed.
PE 5: Test passed.
PE 0 computedsum= 15.00000
PE 1 computedsum= 15.00000
PE 2 computedsum= 15.00000

```

```
PE 3 computedsum= 15.00000
PE 4 computedsum= 15.00000
PE 5 computedsum= 15.00000
```

## 14.11 Running Multiple Parallel Applications under CNL

If you are running multiple parallel applications, the number of processors must be equal to or greater than the **total** number of processors specified by calls to `aprun`. For example, in job script `mult_par_cnl`, the `-l mppwidth` value is 11 because the total of the `aprun n` values is 11.

Modules required:

```
xtpe-target-cnl
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

**Note:** Do not load the `xt-pbs` module. Unload it if it has been loaded.

Create `mult_par_cnl`:

```
#!/bin/bash
#
Define the destination of this job
as the queue named "workq":
#PBS -q workq
#PBS -l mppwidth=11
Tell PBS Pro to keep both standard output and
standard error on the execution host:
#PBS -k eo
cd /lus/nid00007/user1
aprun -n 2 ./simple_cnl &
aprun -n 3 ./mpi_cnl &
aprun -n 6 ./shmem_put_cnl &
aprun -n 6 ./shmem_get_cnl &
exit 0
```

The script launches applications `simple_cnl` (see [Section 14.1, page 115](#)), `mpi_cnl` (see [Section 14.2, page 116](#)), `shmem_put_cnl` (see [Section 14.3, page 118](#)), and `shmem_get_cnl` (see [Section 14.4, page 120](#)).

Set file permission to executable:

```
% chmod +x mult_par_cnl
```

Run the script:

```
% qsub mult_par_cnl
```

List the output:

```
% cat mult_par_cnl.o7231
hello from pe 0 of 2
hello from pe 1 of 2
Application 520255 resources: utime 0, stime 0
My PE: 0 My part: 1683
My PE: 2 My part: 1650
My PE: 1 My part: 1717
PE: 0 Total is: 5050
Application 520256 resources: utime 0, stime 0
PE 0: Test passed.
PE 5: Test passed.
PE 4: Test passed.
PE 3: Test passed.
PE 2: Test passed.
PE 1: Test passed.
Application 520258 exit codes: 64
Application 520258 resources: utime 0, stime 0
PE 0 computedsum= 15.00000
PE 5 computedsum= 15.00000
PE 4 computedsum= 15.00000
PE 3 computedsum= 15.00000
PE 2 computedsum= 15.00000
PE 1 computedsum= 15.00000
Application 520259 resources: utime 0, stime 0
```

## 14.12 Using aprun Memory Affinity Options

In some cases, remote-NUMA-node memory references can reduce the performance of Cray XT5 applications. You can use the `aprun` memory affinity options to control remote-NUMA-node memory references. For the `-S`, `-s1`, and `-sn` options, memory allocation is satisfied using local-NUMA-node memory. If there is not enough NUMA node 0 memory, NUMA node 1 memory may be used. For the `-ss`, only local-NUMA-node memory can be allocated.

### 14.12.1 Using the `aprun -s` Option

This example runs each PE on a specific NUMA node 0 CPU:

```
% aprun -n 4 -q ./xthi | sort
PE 0 nid00045 Core affinity = 0
PE 1 nid00045 Core affinity = 1
PE 2 nid00045 Core affinity = 2
PE 3 nid00045 Core affinity = 3
```

PEs 0-3 cannot access NUMA node 1 memory.

This example runs one PE on each NUMA node of nodes 45 and 70:

```
% aprun -n 4 -s 1 -q ./xthi | sort
PE 0 nid00045 Core affinity = 0
PE 1 nid00045 Core affinity = 4
PE 2 nid00070 Core affinity = 0
PE 3 nid00070 Core affinity = 4
```

### 14.12.2 Using the `aprun -s1` Option

This example runs all PEs on NUMA node 0:

```
% aprun -n 4 -s1 0 -q ./xthi | sort
PE 0 nid00045 Core affinity = 0
PE 1 nid00045 Core affinity = 1
PE 2 nid00045 Core affinity = 2
PE 3 nid00045 Core affinity = 3
```

PEs 0-3 cannot access NUMA node 1 memory.

This example runs all PEs on NUMA node 1:

```
% aprun -n 4 -sl 1 -q ./xthi | sort
PE 0 nid00045 Core affinity = 4
PE 1 nid00045 Core affinity = 5
PE 2 nid00045 Core affinity = 6
PE 3 nid00045 Core affinity = 7
```

PEs 4-7 cannot access NUMA node 0 memory.

#### 14.12.3 Using the `aprun -sn` Option

This example runs four PEs on NUMA node 0 of node 45 and four PEs on NUMA node 0 of node 70:

```
% aprun -n 8 -sn 1 -q ./xthi | sort
PE 0 nid00045 Core affinity = 0
PE 1 nid00045 Core affinity = 1
PE 2 nid00045 Core affinity = 2
PE 3 nid00045 Core affinity = 3
PE 4 nid00070 Core affinity = 0
PE 5 nid00070 Core affinity = 1
PE 6 nid00070 Core affinity = 2
PE 7 nid00070 Core affinity = 3
```

The PEs cannot access NUMA node 1 memory on either node.

#### 14.12.4 Using the `aprun -ss` Option

When `-ss` is specified, a PE can allocate only the memory local to its assigned NUMA node. The default is to allow remote-NUMA-node memory allocation. For example, by default any PE running on NUMA node 0 can allocate NUMA node 1 memory (if NUMA node 1 has been reserved for the application).

This example runs PEs 0-3 on NUMA node 0 and PEs 4-7 on NUMA node 1. PEs 0-3 cannot allocate NUMA node 1 memory, and PEs 4-7 cannot allocate NUMA node 0 memory.

```
% aprun -n 8 -sl 0,1 -ss -q ./xthi | sort
PE 0 nid00056.Core affinity = 0-3
PE 1 nid00056.Core affinity = 0-3
PE 2 nid00056.Core affinity = 0-3
PE 3 nid00056.Core affinity = 0-3
PE 4 nid00056.Core affinity = 4-7
PE 5 nid00056.Core affinity = 4-7
PE 6 nid00056.Core affinity = 4-7
PE 7 nid00056.Core affinity = 4-7
```

## 14.13 Using aprun CPU Affinity Options

The following examples show how you can use aprun CPU affinity options to bind a process to a particular CPU or the CPUs on a NUMA node.

### 14.13.1 Using the aprun `-cc cpu_list` Option

This example binds PEs to CPUs 0-4 and 7:

```
% aprun -n 6 -cc 0-4,7 -q ./xthi | sort
PE 0 nid00045 Core affinity = 0
PE 1 nid00045 Core affinity = 1
PE 2 nid00045 Core affinity = 2
PE 3 nid00045 Core affinity = 3
PE 4 nid00045 Core affinity = 4
PE 5 nid00045 Core affinity = 7
```

**Note:** You do not by default have access to both NUMA nodes of Cray XT5 compute nodes. If you requests resources that can be fulfilled by one NUMA node, your application does not have access to the other NUMA node. For example, the following command fails because the attempt to bind the PEs to CPUs 4-7 is illegal.

```
% aprun -n 4 -cc 4-7 -q ./xthi | sort
[NID 00045] Apid 651253: Affinity failure: all cpu
values out of range
```

### 14.13.2 Using the `aprun -cc keyword` Options

Processes can migrate from one CPU to another on a node. You can use the `-cc` option to bind PEs to CPUs. This example uses the `-cc cpu` (default) option to bind each PE to a CPU:

```
% aprun -n 8 -cc cpu -q ./xthi | sort
PE 0 nid00045 Core affinity = 0
PE 1 nid00045 Core affinity = 1
PE 2 nid00045 Core affinity = 2
PE 3 nid00045 Core affinity = 3
PE 4 nid00045 Core affinity = 4
PE 5 nid00045 Core affinity = 5
PE 6 nid00045 Core affinity = 6
PE 7 nid00045 Core affinity = 7
```

This example uses the `-cc numa_node` option to bind each PE to the CPUs within a NUMA node:

```
% aprun -n 8 -cc numa_node -q ./xthi | sort
PE 0 nid00045 Core affinity = 0-3
PE 1 nid00045 Core affinity = 0-3
PE 2 nid00045 Core affinity = 0-3
PE 3 nid00045 Core affinity = 0-3
PE 4 nid00045 Core affinity = 4-7
PE 5 nid00045 Core affinity = 4-7
PE 6 nid00045 Core affinity = 4-7
PE 7 nid00045 Core affinity = 4-7
```

## 14.14 Running Compute Node Commands under CNL

You can use the `aprun -b` option to run compute node commands.

The following `aprun` command runs the compute node `grep` command to find references to `MemTotal` in compute node file `/proc/meminfo`:

```
% aprun -b grep MemTotal -q /proc/meminfo
MemTotal: 8124872 kB
```

## 14.15 Using the High-level PAPI Interface under CNL

PAPI provides simple high-level interfaces for instrumenting applications written in C or Fortran. This example shows the use of the `PAPI_start_counters()` and `PAPI_stop_counters()` functions.

**Modules required:**

```
xtpe-target-cnl
xt-papi
```

**and one of the following:**

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

**Source of `papi_hl_cnl.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]);
}
```

**Compile `papi_hl_cnl.c`:**

```
% cc -o papi_hl_cnl papi_hl_cnl.c
```

Run `papi_hl_cnl`:

```
% aprun ./papi_hl_cnl
PAPI_TOT_CYC = 4020
PAPI_TOT_INS = 201
Application 520262 exit codes: 19
Application 520262 resources: utime 0, stime 0
```

## 14.16 Using the Low-level PAPI Interface under CNL

PAPI provides an advanced low-level interface for instrumenting applications. The PAPI library must be initialized before calling any of these functions; initialization can be done by issuing either a high-level function call or a call to `PAPI_library_init()`. This example shows the use of the `PAPI_create_eventset()`, `PAPI_add_event()`, `PAPI_start()`, and `PAPI_read()` functions.

Modules required:

```
xtpe-target-cnl
xt-papi
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

Source of `papi_ll_cnl.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]);
}
```

Compile `papi_ll_cnl.c`:

```
% cc -o papi_ll_cnl papi_ll_cnl.c
```

Run `papi_ll_cnl`:

```
% aprun ./papi_ll_cnl
PAPI_TOT_INS = 97
Application 520264 exit codes: 18
Application 520264 resources: utime 0, stime 0
```

## 14.17 Using CrayPat under CNL

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

**Modules required:**

```
xtpe-target-cnl
xt-craypat
```

**and one of the following:**

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

**Source code of pa1\_cnl.f90:**

```
program main
include 'mpif.h'

 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 pa2\_cnl.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 pa2\_cnl.c and pa1\_cnl.f90 and create executable perf\_cnl:**

```
% cc -c pa2_cnl.c
% ftn -o perf_cnl pa1_cnl.f90 pa2_cnl.o
```

Run `pat_build` to generate instrumented program `perf_cnl+pat`:

```
% pat_build -u -g mpi perf_cnl perf_cnl+pat
INFO: A trace intercept routine was created for the function 'MAIN_'.
INFO: A trace intercept routine was created for the function 'work_'.
```

The `tracegroup` (`-g` option) is `mpi`.

Run `perf_cnl+pat`:

```
% aprun -n 4 ./perf_cnl+pat | sort
CrayPat/X: Version 4.3.2 Revision 2055 09/08/08 14:21:44
Experiment data directory written:
/lus/nid00011/user1/cnl/fortran/perf_cnl+pat+19779-478tdt
hello from pe 0 of 4
hello from pe 1 of 4
hello from pe 2 of 4
hello from pe 3 of 4
Application 3014795 resources: utime 0, stime 0
PE 1: The answer is: 42
PE 1: sizeof(long) = 8
```

**Note:** When executed, the instrumented executable creates directory `progname+pat+PIDkeyletters`, where `.PID` is the process ID that was assigned to the instrumented program at run time.

Run `pat_report` to generate reports `perf_cnl.rpt1` (using default `pat_report` options) and `perf_cnl.rpt2` (using the `-O calltree` option).

```
% pat_report perf_cnl+pat+19756-478tdt > perf_cnl.rpt1
Data file 4/4: [.....]
% pat_report -O calltree perf_cnl+pat+19756-478tdt > perf_cnl.rpt2
Data file 4/4: [.....]
% pat_report -O calltree -f ap2 perf_cnl+pat+19756-478tdt
Output redirected to: perf_cnl+pat+19756-478tdt.ap2
```

**Note:** The `-f ap2` option is used to create a `*.ap2` file for input to Cray Apprentice2 (see [Section 14.18, page 156](#)).

List `perf_cnl.rpt1`:

```
CrayPat/X: Version 4.3.2 Revision 2055 (xf 1985) 09/08/08 14:21:44

Number of PEs (MPI ranks): 4

Number of Threads per PE: 1
```

Number of Cores per Processor: 2

<snip>

Table 1: Profile by Function Group and Function

| Time % | Time     | Imb. Time | Imb. Time % | Calls | Experiment=1<br>Group<br>Function<br>PE='HIDE' |
|--------|----------|-----------|-------------|-------|------------------------------------------------|
| 100.0% | 0.000494 | --        | --          | 257   | Total                                          |
| 99.2%  | 0.000490 | --        | --          | 253   | USER                                           |
| 89.8%  | 0.000444 | 0.000353  | 59.1%       | 1     | MAIN_                                          |
| 7.7%   | 0.000038 | 0.000006  | 17.2%       | 1     | exit                                           |

<snip>

Table 2: Load Balance with MPI Sent Message Stats

| Time % | Time     | Avg Sent<br>Msg Size | Experiment=1<br>Group<br>PE |
|--------|----------|----------------------|-----------------------------|
| 100.0% | 0.000582 | --                   | Total                       |
| 98.9%  | 0.000576 | --                   | USER                        |
| 40.0%  | 0.000932 | --                   | pe.0                        |
| 31.8%  | 0.000741 | --                   | pe.2                        |
| 14.1%  | 0.000328 | --                   | pe.1                        |
| 13.0%  | 0.000304 | --                   | pe.3                        |
| 1.1%   | 0.000006 | --                   | MPI                         |
| 0.3%   | 0.000007 | --                   | pe.2                        |
| 0.3%   | 0.000007 | --                   | pe.0                        |
| 0.2%   | 0.000005 | --                   | pe.1                        |

```
|| 0.2% | 0.000005 | -- |pe.3
|=====
```

<snip>

Table 5: Program Wall Clock Time, Memory High Water Mark

| Process<br>Time | Process<br>HiMem<br>(MBytes) | Experiment=1<br>PE |
|-----------------|------------------------------|--------------------|
| 0.027873        | 82                           | Total              |
| -----           |                              |                    |
| 0.028053        | 81.590                       | pe.2               |
| 0.028043        | 81.594                       | pe.0               |
| 0.027961        | 81.574                       | pe.3               |
| 0.027436        | 81.578                       | pe.1               |
| =====           |                              |                    |

=====  
 ===== Additional details =====

Experiment: trace

<snip>

Estimated minimum overhead per call of a traced function,  
 which was subtracted from the data shown in this report  
 (for raw data, use the option: -s overhead=include):  
 Time 0.599 microseconds

Number of traced functions: 98  
 (To see the list, specify: -s traced\_functions=show)

## List perf\_cnl.rpt2:

CrayPat/X: Version 4.3.2 Revision 2055 (xf 1985) 09/08/08 14:21:44

Number of PEs (MPI ranks): 4

Number of Threads per PE: 1

Number of Cores per Processor: 2

<snip>

Table 1: Function Calltree View

| Time % | Time     | Calls | Experiment=1<br>Calltree<br>PE='HIDE' |
|--------|----------|-------|---------------------------------------|
| 100.0% | 0.000536 | 657   | Total                                 |
| 91.4%  | 0.000490 | 256   | MAIN_                                 |
| 82.7%  | 0.000444 | 1     | MAIN_(exclusive)                      |
| 7.1%   | 0.000038 | 1     | exit                                  |

==== Additional details =====

Experiment: trace

<snip>

Estimated minimum overhead per call of a traced function,  
which was subtracted from the data shown in this report  
(for raw data, use the option: -s overhead=include):  
Time 0.599 microseconds

Number of traced functions: 98

(To see the list, specify: -s traced\_functions=show)

## 14.18 Using Cray Apprentice2 under CNL

In the CrayPat example (Section 14.17, page 150), we ran the instrumented program `perf_cnl` and generated file `perf_cnl+pat+19756-478tdt.ap2`.

To view this Cray Apprentice2 file, first load the `apprentice2` module.

```
% module load apprentice2
```

Then launch Cray Apprentice2:

```
% ap2 perf_cnl+pat+19756-478tdt.ap2
```

We display the results in call-graph form:

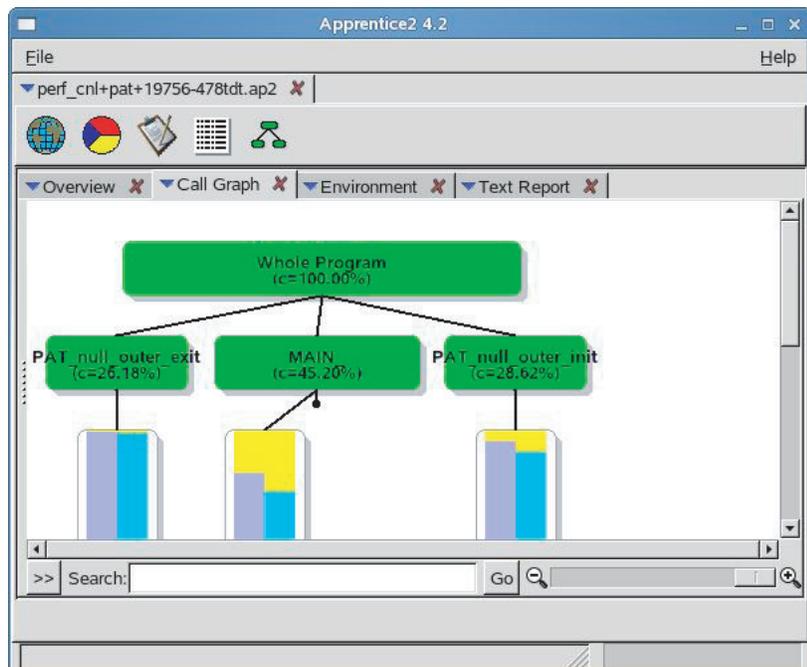


Figure 7. Cray Apprentice2 Display

# Example Catamount Applications [15]

---

This chapter gives examples showing how to compile, link, and run Catamount applications. Use the `module list` command to verify that the correct modules are loaded. If the `xtpe-target-cn1` module is loaded, use:

```
% module swap xtpe-target-cn1 xtpe-target-catamount
```

Each following example lists the additional modules that have to be loaded.

## 15.1 Running a Basic Application under Catamount

This example shows how to use the PGI C compiler to compile an MPI program and `yod` to launch the executable.

Modules required:

```
PrgEnv-pgi
```

and one of the following:

```
PrgEnv-pgi
```

```
PrgEnv-gnu
```

```
PrgEnv-pathscales
```

Create a C program, `simple_qk.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:

```
% cc -o simple_qk simple_qk.c
```

Run the program:

```
% yod -sz 6 simple_qk
hello from pe 0 of 6
hello from pe 3 of 6
hello from pe 5 of 6
hello from pe 4 of 6
hello from pe 2 of 6
hello from pe 1 of 6
```

## 15.2 Running an MPI Application under Catamount

This example shows how to compile, link, and run an MPI program. The MPI program distributes the work represented in a reduction loop, prints the subtotal for each PE, combines the results from the PEs, and prints the total.

Module required:

```
xtpe-target-catamount
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

Create a Fortran program, `mpi_qk.f90`:

```

program reduce
include "mpif.h"

integer n, nres, ierr

call MPI_INIT (ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD,mype,ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD,npes,ierr)

nres = 0
n = 0

do i=mype,100,npes
 n = n + i
enddo

print *, 'My PE:', mype, ' My part:',n

call MPI_REDUCE (n,nres,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)

if (mype == 0) print *,' PE:',mype,'Total is:',nres

call MPI_FINALIZE (ierr)

end

```

Compile `mpi_qk.f90` and create executable `mpi_qk`:

```
% ftn -o mpi_qk mpi_qk.f90
```

Run the program:

```

% yod -sz 6 mpi_qk
My PE: 0 My part: 816
My PE: 3 My part: 867
My PE: 5 My part: 800
My PE: 4 My part: 884
My PE: 2 My part: 850
My PE: 1 My part: 833
PE: 0 Total is: 5050

```

If desired, you could use this C version of the program:

```
/* program reduce */

#include <stdio.h>
#include "mpi.h"

int main (int argc, char *argv[])
{
 int i, sum, mype, npes, nres, ret;
 ret = MPI_Init (&argc, &argv);
 ret = MPI_Comm_size (MPI_COMM_WORLD, &npes);
 ret = MPI_Comm_rank (MPI_COMM_WORLD, &mype);
 nres = 0;
 sum = 0;
 for (i = mype; i <=100; i += npes) {
 sum = sum + i;
 }

 (void) printf ("My PE:%d My part:%d\n",mype, sum);
 ret = MPI_Reduce (&sum,&nres,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD);
 if (mype == 0)
 {
 (void) printf ("PE:%d Total is:%d\n",mype, nres);
 }
 ret = MPI_Finalize ();
}
```

### 15.3 Using the Cray `shmem_put` Function under Catamount

This example shows how to use the `shmem_put64()` function to copy a contiguous data object from the local PE to a contiguous data object on a different PE.

Module required:

`xtpe-target-catamount`

and one of the following:

`PrgEnv-pgi`

`PrgEnv-gnu`

`PrgEnv-pathscale`

Source code of C program (shmem\_put\_qk.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 XT. 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 `shmem_put_qk.c` and create executable `shmem_put_qk`:

```
% cc -o shmem_put_qk shmem_put_qk.c
```

Run `shmem_put_qk`:

```
% yod -sz 4 shmem_put_qk
PE 1: Test passed.
PE 0: Test passed.
PE 2: Test passed.
PE 3: Test passed.
```

## 15.4 Using the Cray `shmem_get` Function under Catamount

This example shows how to use the `shmem_get` function to copy a contiguous data object from a different PE to a contiguous data object on the local PE.

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

**Module required:**

```
xtpe-target-catamount
```

**and one of the following:**

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

**Source code of Fortran program (shmem\_get\_qk.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 shmem\_get\_qk.f90 and create executable shmem2:**

```
% ftn -o shmem_get_qk shmem_get_qk.f90
```

Run `shmem_get_qk`:

```
% yod -sz 6 shmem_get_qk
PE 1 computedsum= 15.00000
PE 0 computedsum= 15.00000
PE 2 computedsum= 15.00000
PE 3 computedsum= 15.00000
PE 5 computedsum= 15.00000
PE 4 computedsum= 15.00000
```

## 15.5 Running a UPC Application under Catamount

This example shows how to compile and run a C program that includes Unified Parallel C (UPC) functions.

Modules required:

```
xtpe-target-catamount
xt-upc
```

and one of the following:

```
PrgEnv-upc-pgi
PrgEnv-upc-gnu
```

**Note:** UPC source files must have the `upc` extension.

Source code of program `upc_qk.upc`:

```
#include <upc.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
 int i;
 for (i = 0; i < THREADS; ++i)
 {
 upc_barrier;
 if (i == MYTHREAD)
 printf ("Hello world from thread: %d\n", MYTHREAD);
 }
 return 0;
}
```

You can use the Berkeley UPC translator or the Intrepid GCCUPC compiler to compile your program.

Compile `upc_qk.upc` using the Berkeley UPC translator:

```
% upcc -o upc_qk upc_qk.upc
```

Run `upc_qk`:

```
% upcrun -n 2 ./upc_qk
UPCR: UPC thread 0 of 2 on nid00012 (process 0 of 2, pid=1448)
UPCR: UPC thread 1 of 2 on nid00013 (process 1 of 2, pid=1412)
Hello world from thread: 0
Hello world from thread: 1
Application 170078 resources: utime 0, stime 0
```

To compile `upc_cnl.upc` using the Intrepid GCCUPC compiler, you need to have the `PrgEnv-upc-gnu` module loaded, and you need to include the `-gccupc` option on the compiler command line:

```
% module swap PrgEnv-upc-pgi PrgEnv-upc-gnu
% upcc -o upc_cnl -gccupc upc_cnl.upc
% upcrun -n 2 ./upc_cnl
UPCR: UPC thread 0 of 2 on nid00012 (process 0 of 2, pid=1443)
UPCR: UPC thread 1 of 2 on nid00013 (process 1 of 2, pid=1407)
Hello world from thread: 0
Hello world from thread: 1
Application 170077 resources: utime 0, stime 0
```

## 15.6 Using `dclock()` to Calculate Elapsed Time under Catamount

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

Module required:

```
xtpe-target-catamount
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

Source code of `dclock_qk.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("\nElapsed time = %f\n", elapsed_time);
}
```

Compile `dclock_qk.c` and create executable `dclock`:

```
% cc -o dclock_qk dclock_qk.c
```

Run `dclock_qk`:

```
% yod -sz 1 dclock_qk
```

```
Elapsed time = 5.000007
```

## 15.7 Specifying a Buffer for I/O under Catamount

An important consideration for C++ I/O in Catamount applications is that the `endl` function causes the data in the buffer to be flushed. In most cases, the `endl` function is used to output a new line, so an `endl` function usually can be replaced in the code by specifying a newline character. In this example, `endl` is redefined to be `'\n'`. If a flush is needed, you can include a call to the `flush()` member function.

Module required:

```
xtpe-target-catamount
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

**Source code of io1\_qk.C**

```

#include <iostream>
#include <catamount/dclock.h>

using namespace std;

#define endl '\n'

int main(int argc, char ** argv) {
 double start, end;
 char *buffer;

 buffer = (char *)malloc(sizeof(char)*12000);
 cout.rdbuf()->pubsetbuf(buffer,12000);
 start = dclock();
 for (int i = 0; i < 1000; i++) {
 cout << "line: " << i << endl;
 }
 end = dclock();
 cout.flush(); // Force a flush of data (not necessary)
 cerr << "Time to write using buffer = " << end - start << endl;

 return 0;
}

```

**Compile io1\_qk.C:**

```
% CC -o io1_qk io1_qk.C
```

**Run io1\_qk, directing output to file tmp:**

```
% yod io1_qk > tmp
```

```
% cat tmp
```

```
Time to write using buffer = 0.000599465
```

**15.8 Changing the Default Buffer Size for I/O-to-file Streams under Catamount**

This example uses a default buffer and a modified buffer to write data and prints the time-to-write value for each process.

**Module required:**`xtpe-target-catamount`**and one of the following:**`PrgEnv-pgi``PrgEnv-gnu``PrgEnv-pathscales`**Source code of io2\_qk.C**

```
#include <iostream>
#include <fstream>
#include <catamount/dclock.h>
using namespace std;
#define endl '\n'

char data[] = " 2345678901234567890123456789 \
0123456789012345678901234567890";

int main(int argc, char ** argv) {
 double start, end;
 char *buffer;

 // Use default buffer
 ofstream data1("output1");
 start = dclock();
 for (int i = 0; i < 10000; i++) {
 data1 << "line: " << i << data << endl;
 }
 end = dclock();
 data1.flush(); // Force a flush of data (not necessary)
 cerr << "Time to write using default buffer = " \
 << end - start << endl ;

 // Set up a buffer
 ofstream data2("output2");
 buffer = (char *)malloc(sizeof(char)*500000);
 data2.rdbuf()->pubsetbuf(buffer,500000);
 start = dclock();
 for (int i = 0; i < 10000; i++) {
 data2 << "line: " << i << data << endl;
 }
 end = dclock();
```

```
data2.flush(); // Force a flush of data (not necessary)
cerr << "Time to write with program buffer = " \
<< end - start << endl ;

return 0;
}
```

Compile io2\_qk.C:

```
% CC -o io2_qk io2_qk.C
```

Run io2\_qk:

```
% yod -sz 1 io2_qk
```

```
Time to write using default buffer = 0.012211
```

```
Time to write with program buffer = 0.0211126
```

## 15.9 Improving the Performance of stdout under Catamount

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:

Module required:

```
xtpe-target-catamount
```

and one of the following:

```
PrgEnv-pgi
```

```
PrgEnv-gnu
```

```
PrgEnv-pathscales
```

Source code of C program (setvbuf\_qk.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, _IOFBUF, bsize);
 }

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

 exit(0);
}
```

Compile setvbuf\_qk.c and create executable setvbuf\_qk:

```
% cc -o setvbuf_qk setvbuf_qk.c
```

Run setvbuf\_qk:

```
% yod -sz 1 setvbuf_qk > tmp
% more tmp
this is line 0
this is line 1
this is line 2
this is line 3
...
this is line 1020
this is line 1021
this is line 1022
this is line 1023
```

## 15.10 Running a PBS Professional Job Script under Catamount

This example of a job script, `pbs_script_qk`, requests four processors to run application `mpi_qk` (see [Section 15.2, page 158](#)).

Modules required:

```
xtpe-target-catamount
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

Do not load the `xt-pbs` module. Unload it if it has been loaded.

Create `pbs_script_qk`.

```
% cat pbs_script_qk
#!/bin/bash
#
Define the destination of this job
as the queue named "workq":
#PBS -q workq
#PBS -l mppwidth=4
Tell PBS Pro to keep both standard output and
standard error on the execution host:
#PBS -k eo
yod -sz 4 mpi_qk
exit 0
```

Set permissions to executable:

```
% chmod +x pbs_script_qk
```

Submit the job:

```
% qsub pbs_script_qk
```

The `qsub` command produces a batch job log file with output from `program1`. The job log file has the form `pbs_script_qk.onnnnnn`.

```
% cat pbs_script_qk.o278419
My PE: 0 My part: 1300
My PE: 2 My part: 1250
My PE: 1 My part: 1225
My PE: 3 My part: 1275
PE: 0 Total is: 5050
```

## 15.11 Running Multiple Sequential Applications under Catamount

To run multiple sequential applications, the number of processors you specify as an argument to `qsub` must be equal to or greater than the **largest number** of processors required by an invocation of `yod` in your script. For example, in job script `mult_seq_qk`, the `-l mppwidth` is 4 because the largest `yod sz` value is 4.

Modules required:

```
xtpe-target-catamount
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

**Note:** Do not load the `xt-pbs` module. Unload it if it has been loaded.

Create script `mult_seq_qk`:

```
#!/bin/bash
#
Define the destination of this job
as the queue named "workq":
#PBS -q workq
#PBS -l mppwidth=4
Tell PBS Pro to keep both standard output and
standard error on the execution host:
#PBS -k eo
yod -sz 2 simple_qk
yod -sz 3 mpi_qk
yod -sz 4 shmem_put_qk
yod -sz 2 shmem_get_qk
exit 0
```

The script launches applications `simple_qk` (see [Section 15.1, page 157](#)), `mpi_qk` (see [Section 15.2, page 158](#)), `shmem_put_qk` (see [Section 15.3, page 160](#)), and `shmem_get_qk` (see [Section 15.4, page 162](#)).

Set file permissions to executable:

```
% chmod +x mult_seq_qk
```

Run the script:

```
% qsub mult_seq_qk
```

List the output:

```
% more mult_seq_qk.o278425
hello from pe 1 of 2
hello from pe 0 of 2
My PE: 0 My part: 1683
My PE: 2 My part: 1650
My PE: 1 My part: 1717
PE: 0 Total is: 5050
PE 1: Test passed.
PE 0: Test passed.
PE 2: Test passed.
PE 3: Test passed.
PE 0 computedsum= 1.000000
PE 1 computedsum= 1.000000
```

## 15.12 Running Multiple Parallel Applications under Catamount

If you are running multiple parallel applications, the number of processors must be equal to or greater than the **total** number of processors specified by calls to `yod`. For example, in job script `mult_par_qk`, the `-l mppwidth` value is 11 because the total of the `yod -sz` values is 11.

Modules required:

```
xtpe-target-catamount
pbs
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

Do not load the `xt-pbs` module. Unload it if it has been loaded.

The following example shows how to use a job script, the `qsub` command, and `yod` commands to run a job that launches four applications.

Create script `mult_par_qk`:

```
#!/bin/bash
#
Define the destination of this job
as the queue named "workq":
#PBS -q workq
#PBS -l mppwidth=11
Tell PBS Pro to keep both standard output and
standard error on the execution host:
#PBS -k eo
yod -sz 2 simple_qk &
yod -sz 3 mpi_qk &
yod -sz 4 shmem_put_qk &
yod -sz 2 shmem_get_qk &
exit 0
```

The script launches applications `simple_qk` (see [Section 15.1, page 157](#)), `mpi_qk` (see [Section 15.2, page 158](#)), `shmem_put_qk` (see [Section 15.3, page 160](#)), and `shmem_get_qk` (see [Section 15.4, page 162](#)).

Set file permissions to executable:

```
% chmod +x mult_par_qk
```

Run the script:

```
% qsub mult_par_qk
```

List the output:

```
% cat mult_par_qk.o13422
hello from pe 0 of 2
hello from pe 1 of 2
PE 1: sizeof(long) = 8
PE 1: The answer is: 42
PE 0 : The answer is: -1184
PE 0: Test passed.
PE 3: Test passed.
PE 2: Test passed.
PE 1: Test passed.
PE 0 computedsum= 1.000000
PE 1 computedsum= 1.000000
```

### 15.13 Using `xtgdb` under Catamount

This example uses the GNU debugger, `xtgdb`, to debug a program.

Modules required:

```
xtpe-target-catamount
xtgdb
PrgEnv-gnu
```

Compile program `hi_qk.c`:

```
% cc -g hi_qk.c
```

Initiate a PBS Professional interactive session:

```
% qsub -I
```

Run `xtgdb`:

```
% xtgdb yod a.out
 Debugging a.out
 Target port is 33381

Please wait while connecting to catamount...

target remote :33381
Remote debugging using :33381
0x000000000200001 in _start ()
```

Set breakpoints, resume execution, and quit the `gdb` session:

```
(gdb) b main

Breakpoint 3 at 0x205674: file hi.c, line 3.

(gdb) c

Continuing.

Breakpoint 3, main () at hi.c:3
3 printf("hello.c\n");

(gdb) c

Continuing.
hello.c

Program exited with code 0377.

(gdb) quit

Done
```

## 15.14 Using the High-level PAPI Interface under Catamount

PAPI provides simple high-level interfaces for instrumenting applications written in C or Fortran. This example shows the use of the `PAPI_start_counters()` and `PAPI_stop_counters()` functions.

**Modules required:**

```
xtpe-target-catamount
xt-papi
```

**and one of the following:**

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

**Source code of papi\_hl\_qk.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]);
}
```

**Compile papi\_hl\_qk.c:**

```
% cc -o papi_hl_qk papi_hl_qk.c
```

**Run papi\_hl\_qk:**

```
% yod papi_hl_qk
PAPI_TOT_CYC = 2816
PAPI_TOT_INS = 277
```

## 15.15 Using the Low-level PAPI Interface under Catamount

PAPI provides an advanced low-level interface for instrumenting applications. The PAPI library must be initialized before calling any of these functions; initialization can be done by issuing either a high-level function call or a call to `PAPI_library_init()`. This example shows the use of the `PAPI_create_eventset()`, `PAPI_add_event()`, `PAPI_start()`, and `PAPI_read()` functions.

### Modules required:

```
xtpe-target-catamount
xt-papi
```

### and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscales
```

### Source code of `papi_ll_qk.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]);
}

```

Compile `papi_ll_qk.c`:

```
% cc -o papi_ll_qk papi_ll_qk.c
```

Run `papi_ll_qk`:

```
% yod papi_ll_qk
PAPI_TOT_INS = 170
```

## 15.16 Using CrayPat under Catamount

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

Modules required:

```
xtpe-target-catamount
xt-craypat
```

and one of the following:

```
PrgEnv-pgi
PrgEnv-gnu
PrgEnv-pathscale
```

Compile the sample program `pa1_qk.f90` and the routine it calls, `pa2_qk.c`.

Source code of `pa1_qk.f90`:

```
program main
```

```
include 'mpif.h'

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 pa2\_qk.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 pa1\_qk.f90 and pa2\_qk.c and create executable pa\_qk:

```
% cc -c pa2_qk.c
% ftn -o pa_qk pa1_qk.f90 pa2_qk.o
```

Run pat\_build to generate instrumented program pa\_qk+pat:

```
% pat_build -u -g mpi pa_qk pa_qk+pat
INFO: A trace intercept routine was created for the function 'MAIN_'.
INFO: A trace intercept routine was created for
the function 'work_'.
```

The tracegroup (-g option) is mpi.

Run pa\_qk+pat:

```
% yod -sz 4 pa_qk+pat
CrayPat/X: Version 4.2 Revision 1640 04/22/08 16:22:24
hello from pe 1 of 4
hello from pe 3 of 4
hello from pe 2 of 4
hello from pe 0 of 4
PE 1: sizeof(long) = 8
PE 1: The answer is: 42
Experiment data file written:
/lus/nid00007/user1/catamount/fortran/pa_qk+pat+18-16td.xf
```

**Note:** When executed, the instrumented executable creates directory *progname+pat+PIDkeyletters* that contains one or more data files with a *.xf* suffix. *PID* is the process ID that was assigned to the instrumented program at run time.

Run `pat_report` to generate reports `pa_qk.rpt1` (using default `pat_report` options) and `pa_qk.rpt2` (using the `-O calltree` option).

```
% pat_report pa_qk+pat+87td.xf > pa_qk.rpt1
pat_report: Creating file: pa_qk+pat+18-16td.ap2
Data file 1/1: [.....]
% pat_report -O calltree pa_qk+pat+87td.xf > pa_qk.rpt2
pat_report: Using existing file: pa_qk+pat+18-16td.ap2
Data file 1/1: [.....]
```

List `pa_qk.rpt1`:

```
CrayPat/X: Version 4.2 Revision 1640 (xf 1609) 04/22/08 16:22:24

Experiment: trace

Experiment data file:
 /lus/nid00007/user1/catamount/fortran/pa_qk+pat+18-16td.xf (RTS)

Current path to data file:
 /lus/nid00007/user1/catamount/fortran/pa_qk+pat+18-16td.ap2 (RTS)

Original program: /lus/nid00007/user1/catamount/fortran/pa_qk

Instrumented with: pat_build -u -g mpi pa_qk pa_qk+pat

Instrumented program: /lus/nid00007/user1/catamount/fortran/pa_qk+pat
```

Program invocation: pa\_qk+pat

Number of PEs: 4

Number of Threads per PE: 1

Number of Cores per Processor: 1

Exit Status: 0 PEs: 0-3

Runtime environment variables:

```
MPICHBASEDIR=/opt/xt-mpt/2.1.12/mpich2-64
MPICH_DIR=/opt/xt-mpt/2.1.12/mpich2-64/P2
MPICH_DIR_FTN_DEFAULT64=/opt/xt-mpt/2.1.12/mpich2-64/P2W
```

Report time environment variables:

```
CRAYPAT_ROOT=/opt/xt-tools/craypat/4.2/v23/cpatx
```

<snip>

Estimated minimum overhead per call of a traced function, which was subtracted from the data shown in this report (for raw data, use the option: -s overhead=include):

```
Time 0.710 microseconds
```

Number of traced functions: 54

(To see the list, specify: -s traced\_functions=show)

<snip>

Table 1: Profile by Function Group and Function

| Time % | Time     | Imb. Time | Imb. Time % | Calls | Group | Function  |
|--------|----------|-----------|-------------|-------|-------|-----------|
|        |          |           |             |       |       | PE='HIDE' |
| 100.0% | 0.000417 | --        | --          | 257   | Total |           |
| -----  |          |           |             |       |       |           |
| 99.6%  | 0.000415 | --        | --          | 253   | USER  |           |
| -----  |          |           |             |       |       |           |
| 76.0%  | 0.000317 | 0.000060  | 21.2%       | 1     | MAIN_ |           |
| 18.1%  | 0.000076 | 0.000209  | 97.9%       | 250   | work_ |           |
| 5.2%   | 0.000022 | 0.000000  | 1.8%        | 1     | exit  |           |
| =====  |          |           |             |       |       |           |

<snip>

Table 2: Heap Usage at Start and End of Main Program

| MB Heap<br>Used at<br>Start | MB Heap<br>Free at<br>Start | Heap<br>Not<br>Freed<br>MB | Max Free<br>Object at<br>End | PE    |
|-----------------------------|-----------------------------|----------------------------|------------------------------|-------|
| 79.953                      | 1854.046                    | 12.104                     | 1841.940                     | Total |
| 79.970                      | 1854.030                    | 12.100                     | 1841.928                     | pe.0  |
| 79.948                      | 1854.052                    | 12.106                     | 1841.943                     | pe.1  |
| 79.948                      | 1854.052                    | 12.104                     | 1841.946                     | pe.3  |
| 79.948                      | 1854.052                    | 12.106                     | 1841.943                     | pe.2  |

<snip>

Table 3: Program Wall Clock Time

| Process<br>Time | PE    |
|-----------------|-------|
| 0.261440        | Total |
| 0.285436        | pe.1  |
| 0.269482        | pe.0  |
| 0.253405        | pe.2  |
| 0.237437        | pe.3  |

List pa\_qk.rpt2:

CrayPat/X: Version 4.2 Revision 1640 (xf 1609) 04/22/08 16:22:24

Experiment: trace

Experiment data file:

/lus/nid00007/user1/catamount/fortran/pa\_qk+pat+18-16td.xf (RTS)

Current path to data file:

/lus/nid00007/user1/catamount/fortran/pa\_qk+pat+18-16td.ap2 (RTS)

Original program: /lus/nid00007/user1/catamount/fortran/pa\_qk

Instrumented with: pat\_build -u -g mpi pa\_qk pa\_qk+pat

Instrumented program: /lus/nid00007/user1/catamount/fortran/pa\_qk+pat

Program invocation: pa\_qk+pat

Number of PEs: 4

Number of Threads per PE: 1

Number of Cores per Processor: 1

Exit Status: 0 PEs: 0-3

Runtime environment variables:

```
MPICHBASEDIR=/opt/xt-mpt/2.1.12/mpich2-64
MPICH_DIR=/opt/xt-mpt/2.1.12/mpich2-64/P2
MPICH_DIR_FTN_DEFAULT64=/opt/xt-mpt/2.1.12/mpich2-64/P2W
```

Report time environment variables:

```
CRAYPAT_ROOT=/opt/xt-tools/craypat/4.2/v23/cpatx
```

<snip>

Estimated minimum overhead per call of a traced function, which was subtracted from the data shown in this report (for raw data, use the option: -s overhead=include):

```
Time 0.710 microseconds
```

Number of traced functions: 54

(To see the list, specify: -s traced\_functions=show)

<snip>

Table 1: Function Calltree View

```
Time % | Time | Calls | Calltree
Percentages at each level are of the Total for the program.
(For percentages relative to next level up, specify:
-s percent=r[relative])
```

Table 1: Function Calltree View

```
Time % | Time | Calls | Calltree
 | | | PE='HIDE'
-----|-----|-----|-----
100.0% | 0.000437 | 657 | Total
-----|-----|-----|-----
| 95.5% | 0.000417 | 257 | main
```

```

| |-----
| | 90.2% | 0.000394 | 255 |MAIN_
| |-----
3| | 72.5% | 0.000317 | 1 |MAIN_(exclusive)
3| | 17.3% | 0.000076 | 250 |work_
| |=====
| | 5.0% | 0.000022 | 1 |exit
| |=====
| | 3.1% | 0.000014 | 200 |__do_global_ctors
| | 1.5% | 0.000006 | 200 |exit
| |=====

```



# glibc Functions Supported in CNL [A]

---

The glibc functions and system calls supported in CNL are listed in [Table 8](#). For further information, see the man pages.

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

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

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

Table 8. Supported glibc Functions for CNL

---

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

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

|                   |                |                  |              |
|-------------------|----------------|------------------|--------------|
| getdirenties      | getdomainname  | getegid          | getenv       |
| geteuid           | getfsent       | getfsfile        | getfsspec    |
| getgid            | gethostname    | getline          | getlogin     |
| getlogin_r        | getmntent      | getopt           | getopt_long  |
| getopt_long_only  | getpagesize    | getpass          | getpid       |
| getprotoent       | getprotobyname | getprotobynumber |              |
| getrlimit         | getrusage      | gettext          | gettimeofday |
| getttyent         | getttynam      | getuid           | getusershell |
| getw              | getwc          | getwc_unlocked   | getwchar     |
| getwchar_unlocked | gmtime         | gmtime_r         | gsignal      |
| hasmntopt         | hcreate        | hcreate_r        | hdestroy     |
| hsearch           | iconv          | iconv_close      | iconv_open   |
| imaxabs           | index          | initstate        | insque       |
| ioctl             | isalnum        | isalpha          | isascii      |
| isblank           | iscntrl        | isdigit          | isgraph      |
| isinf             | islower        | isnan            | isprint      |
| ispunct           | isspace        | isupper          | iswalnum     |
| iswalpha          | iswblank       | iswcntrl         | iswctype     |
| iswdigit          | iswgraph       | iswlower         | iswprint     |
| iswpunct          | iswspace       | iswupper         | iswxdigit    |
| isxdigit          | jrand48        | kill             | l64a         |
| labs              | lcong48        | ldexp            | lfind        |
| link              | llabs          | localeconv       | localtime    |
| localtime_r       | lockf          | longjmp          | lrand48      |
| lsearch           | lseek          | lstat            | malloc       |
| mblen             | mbrlen         | mbrtowc          | mbsinit      |
| mbsnrtowcs        | mbsrtowcs      | mbstowcs         | mbtowc       |
| memccpy           | memchr         | memcmp           | memcpy       |
| memfrob           | memmem         | memmove          | memchr       |
| memset            | mkdir          | mkdtemp          | mknod        |

|                |             |                   |            |
|----------------|-------------|-------------------|------------|
| mkstemp        | mktime      | modf              | modff      |
| modfl          | mrnd48      | nanosleep         | ngettext   |
| nl_langinfo    | nrnd48      | on_exit           | open       |
| opendir        | passwd2des  | pclose            | perror     |
| pread          | printf      | psignal           | putc       |
| putc_unlocked  | putchar     | putchar_unlocked  | putenv     |
| putpwent       | puts        | putw              | putwc      |
| putwc_unlocked | putwchar    | putwchar_unlocked | pwrite     |
| qecvt          | qecvt_r     | qfcvt             | qfcvt_r    |
| qgcvt          | qsort       | raise             | rand       |
| random         | re_comp     | re_exec           | read       |
| readdir        | readlink    | readv             | realloc    |
| realpath       | regcomp     | regerror          | regexec    |
| regfree        | registerrpc | remove            | remque     |
| rename         | rewind      | rewinddir         | rindex     |
| rmdir          | scandir     | scanf             | seed48     |
| seekdir        | setbuf      | setbuffer         | setegid    |
| setenv         | seteuid     | setfsent          | setgid     |
| setitimer      | setjmp      | setlinebuf        | setlocale  |
| setlogmask     | setmntent   | setrlimit         | setstate   |
| settyent       | setuid      | setusershell      | setvbuf    |
| sigaction      |             | sigaddset         | sigdelset  |
| sigemptyset    | sigfillset  | sigismember       | siglongjmp |
| signal         | sigpending  | sigprocmask       | sigsuspend |
| sleep          | snprintf    | sprintf           | srnd       |
| srnd48         | srandom     | sscanf            | ssignal    |
| stat           | stpncpy     | stpncpy           | strcascmp  |
| strcat         | strchr      | strcmp            | strcoll    |
| strcpy         | strcspn     | strdup            | strerror   |
| strerror_r     | strfmon     | strfry            | strftime   |

|          |             |             |               |
|----------|-------------|-------------|---------------|
| strlen   | strncasecmp | strncat     | strncmp       |
| strncpy  | strndup     | strnlen     | strpbrk       |
| strptime | strchr      | strsep      | strsignal     |
| strspn   | strstr      | strtod      | strtof        |
| strtok   | strtok_r    | strtol      | strtold       |
| strtoll  | strtoq      | strtoul     | strtoull      |
| strtouq  | strverscmp  | strxfrm     | svcfld_create |
| swab     | swprintf    | symlink     | syscall       |
| sysconf  | tdelete     | telldir     | textdomain    |
| tfind    | time        | timegm      | timelocal     |
| timezone | tmpfile     | toascii     | tolower       |
| toupper  | towctrans   | towlower    | towupper      |
| truncate | tsearch     | ttyslot     | twalk         |
| tzname   | tzset       | umask       | umount        |
| uname    | ungetc      | ungetwc     | unlink        |
| unsetenv | usleep      | utime       | vasprintf     |
| vdprintf | verr        | verrx       | versionsort   |
| vfork    | vfprintf    | vfscanf     | vfwprintf     |
| vprintf  | vscanf      | vsnprintf   | vsprintf      |
| vsscanf  | vswprintf   | vwarn       | vwarnx        |
| vwprintf | warn        | warnx       | wcpcpy        |
| wcpcpy   | wcrtomb     | wcscasecmp  | wcscat        |
| wcschr   | wscmp       | wscopy      | wcscspn       |
| wcsdup   | wcslen      | wcsncasecmp | wcsncat       |
| wcsncmp  | wcsncpy     | wcsnlen     | wcsnrtoombs   |
| wcspbrk  | wcsrchr     | wcsrtombs   | wcsspn        |
| wcsstr   | wcstok      | wcstombs    | wcswidth      |
| wctob    | wctomb      | wctrans     | wctype        |
| wcwidth  | wmemchr     | wmemcmp     | wmemcpy       |

wmemmove  
writev

wmemset  
xdecrypt

wprintf  
xencrypt

write

---

# glibc Functions Supported in Catamount [B]

---

The Catamount port of glibc supports the functions listed in [Table 9](#). For further information, see the man pages.

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

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

The Cray XT system supports two implementations of `malloc()` for compute nodes running Catamount: GNU `malloc` and Catamount `malloc`. If your code makes generous use of `malloc()`, `alloc()`, `realloc()`, or automatic arrays, you may notice improvements in scaling by loading the GNU `malloc` module and relinking.

To use GNU `malloc`, load the `gmalloc` module:

```
% module load gmalloc
```

Entry points in `libgmalloc.a` (GNU `malloc`) are referenced before those in `libc.a` (Catamount `malloc`).

Table 9. Supported glibc Functions for Catamount

---

|                              |                           |                             |                          |
|------------------------------|---------------------------|-----------------------------|--------------------------|
| <code>a64l</code>            | <code>abort</code>        | <code>abs</code>            | <code>access</code>      |
| <code>addmntent</code>       | <code>alarm</code>        | <code>alphasort</code>      | <code>argz_add</code>    |
| <code>argz_add_sep</code>    | <code>argz_append</code>  | <code>argz_count</code>     | <code>argz_create</code> |
| <code>argz_create_sep</code> | <code>argz_delete</code>  | <code>argz_extract</code>   | <code>argz_insert</code> |
| <code>argz_next</code>       | <code>argz_replace</code> | <code>argz_stringify</code> | <code>asctime</code>     |

|                 |                   |                         |                 |
|-----------------|-------------------|-------------------------|-----------------|
| asctime_r       | asprintf          | atexit                  | atof            |
| atoi            | atol              | atoll                   | basename        |
| bcmp            | bcopy             | bind_textdomain_codeset | bindtextdomain  |
| bsearch         | btowc             | bzero                   | calloc          |
| catclose        | catgets           | catopen                 | cbc_crypt       |
| chdir           | chmod             | chown                   | clearenv        |
| clearerr        | clearerr_unlocked | close                   | closedir        |
| confstr         | copysign          | copysignf               | copysignl       |
| creat           | ctime             | ctime_r                 | daemon          |
| daylight        | dcgettext         | dcngettext              | des_setparity   |
| dgettext        | difftime          | dirfd                   | dirname         |
| div             | dngettext         | dprintf                 | drand48         |
| dup             | dup2              | dysize                  | ecb_crypt       |
| ecvt            | ecvt_r            | endsent                 | endmntent       |
| endttyent       | endusershell      | envz_add                | envz_entry      |
| envz_get        | envz_merge        | envz_remove             | envz_strip      |
| erand48         | err               | errx                    | exit            |
| fchmod          | fchown            | fclose                  | fcloseall       |
| fcntl           | fcvt              | fcvt_r                  | fdatasync       |
| fdopen          | feof              | feof_unlocked           | ferror          |
| ferror_unlocked | fflush            | fflush_unlocked         | ffs             |
| ffsl            | ffsll             | fgetc                   | fgetc_unlocked  |
| fgetgrent       | fgetpos           | fgetpwent               | fgets           |
| fgets_unlocked  | fgetwc            | fgetwc_unlocked         | fgetws          |
| fgetws_unlocked | fileno            | fileno_unlocked         | finite          |
| flockfile       | fnmatch           | fopen                   | fprintf         |
| fputc           | fputc_unlocked    | fputs                   | fputs_unlocked  |
| fputwc          | fputwc_unlocked   | fputws                  | fputws_unlocked |
| fread           | fread_unlocked    | free                    | freopen         |
| frexp           | fscanf            | fseek                   | fseeko          |

|                   |                 |                |                      |
|-------------------|-----------------|----------------|----------------------|
| fsetpos           | fstat           | fsync          | ftell                |
| ftello            | ftime           | ftok           | ftruncate            |
| ftrylockfile      | funlockfile     | fwide          | fwprintf             |
| fwrite            | fwrite_unlocked | gcvt           | get_current_dir_name |
| getc              | getc_unlocked   | getchar        | getchar_unlocked     |
| getcwd            | getdate         | getdate_r      | getdelim             |
| getdirent         | getdomainname   | getegid        | getenv               |
| geteuid           | getfsent        | getfsfile      | getfsspec            |
| getgid            | gethostname     | getline        | getlogin             |
| getlogin_r        | getmntent       | getopt         | getopt_long          |
| getopt_long_only  | getpagesize     | getpass        | getpid               |
| getrlimit         | getrusage       | gettext        | gettimeofday         |
| gettyent          | gettyname       | getuid         | getusershell         |
| getw              | getwc           | getwc_unlocked | getwchar             |
| getwchar_unlocked | gmtime          | gmtime_r       | gsignal              |
| hasmntopt         | hcreate         | hcreate_r      | hdestroy             |
| hsearch           | iconv           | iconv_close    | iconv_open           |
| imaxabs           | index           | initstate      | insque               |
| ioctl             | isalnum         | isalpha        | isascii              |
| isblank           | iscntrl         | isdigit        | isgraph              |
| isinf             | islower         | isnan          | isprint              |
| ispunct           | isspace         | isupper        | iswalnum             |
| iswalpha          | iswblank        | iswcntrl       | iswctype             |
| iswdigit          | iswgraph        | iswlower       | iswprint             |
| iswpunct          | iswspace        | iswupper       | iswxdigit            |
| isxdigit          | jrand48         | kill           | l64a                 |
| labs              | lcong48         | ldexp          | lfind                |
| link              | llabs           | localeconv     | localtime            |
| localtime_r       | lockf           | longjmp        | lrand48              |
| lsearch           | lseek           | lstat          | malloc               |

|                |             |                   |            |
|----------------|-------------|-------------------|------------|
| mblen          | mbrlen      | mbrtowc           | mbsinit    |
| mbsnrtowcs     | mbsrtowcs   | mbstowcs          | mbtowc     |
| memccpy        | memchr      | memcmp            | memcpy     |
| memfrob        | memmem      | memmove           | memrchr    |
| memset         | mkdir       | mkdtemp           | mknod      |
| mkstemp        | mktime      | modf              | modff      |
| modfl          | rand48      | nanosleep         | ngettext   |
| nl_langinfo    | rand48      | on_exit           | open       |
| opendir        | passwd2des  | pclose            | perror     |
| pread          | printf      | psignal           | putc       |
| putc_unlocked  | putchar     | putchar_unlocked  | putenv     |
| putpwent       | puts        | putw              | putwc      |
| putwc_unlocked | putwchar    | putwchar_unlocked | pwrite     |
| qecvt          | qecvt_r     | qfcvt             | qfcvt_r    |
| qgcvt          | qsort       | raise             | rand       |
| random         | re_comp     | re_exec           | read       |
| readdir        | readlink    | readv             | realloc    |
| realpath       | regcomp     | regerror          | regexec    |
| regfree        | registerrpc | remove            | remque     |
| rename         | rewind      | rewinddir         | rindex     |
| rmdir          | scandir     | scanf             | seed48     |
| seekdir        | setbuf      | setbuffer         | setegid    |
| setenv         | seteuid     | setfsent          | setgid     |
| setitimer      | setjmp      | setlinebuf        | setlocale  |
| setlogmask     | setmntent   | setrlimit         | setstate   |
| setttyent      | setuid      | setusershell      | setvbuf    |
| sigaction      |             | sigaddset         | sigdelset  |
| sigemptyset    | sigfillset  | sigismember       | siglongjmp |
| signal         | sigpending  | sigprocmask       | sigsuspend |
| sleep          | sprintf     | sprintf           | srand      |

---

|            |             |             |               |
|------------|-------------|-------------|---------------|
| rand48     | srandom     | sscanf      | ssignal       |
| stat       | stpcpy      | stpncpy     | strcasecmp    |
| strcat     | strchr      | strcmp      | strcoll       |
| strcpy     | strcspn     | strdup      | strerror      |
| strerror_r | strfmon     | strfry      | strftime      |
| strlen     | strncasecmp | strncat     | strncmp       |
| strncpy    | strndup     | strnlen     | strpbrk       |
| strptime   | strrchr     | strsep      | strsignal     |
| strspn     | strstr      | strtod      | strtof        |
| strtok     | strtok_r    | strtol      | strtold       |
| strtoll    | strtoq      | strtoul     | strtoull      |
| strtouq    | strverscmp  | strxfrm     | svcfld_create |
| swab       | swprintf    | symlink     | syscall       |
| sysconf    | tdelete     | telldir     | textdomain    |
| tfind      | time        | timegm      | timelocal     |
| timezone   | tmpfile     | toascii     | tolower       |
| toupper    | towctrans   | towlower    | towupper      |
| truncate   | tsearch     | ttyslot     | twalk         |
| tzname     | tzset       | umask       | umount        |
| uname      | ungetc      | ungetwc     | unlink        |
| unsetenv   | usleep      | utime       | vasprintf     |
| vdprintf   | verr        | verrx       | versionsort   |
| vfork      | vfprintf    | vfscanf     | vfwprintf     |
| vprintf    | vscanf      | vsnprintf   | vsprintf      |
| vsscanf    | vswprintf   | vwarn       | vwarnx        |
| vwprintf   | warn        | warnx       | wcpcpy        |
| wcpcpy     | wcrtomb     | wcscasecmp  | wcscat        |
| wcschr     | wcscmp      | wcscopy     | wcscspn       |
| wcsdup     | wcslen      | wcsncasecmp | wcsncat       |
| wcsncmp    | wcsncpy     | wcsnlen     | wcsnrtombs    |

|          |          |           |          |
|----------|----------|-----------|----------|
| wcspbrk  | wcsrchr  | wcsrtombs | wcsspn   |
| wcsstr   | wcstok   | wcstombs  | wcswidth |
| wctob    | wctomb   | wctrans   | wctype   |
| wcwidth  | wmemchr  | wmemcmp   | wmemcpy  |
| wmemmove | wmemset  | wprintf   | write    |
| writev   | xdecrypt | xencrypt  |          |

---

# PAPI Hardware Counter Presets [C]

---

[Table 10](#) describes the hardware counter presets that are available on the Cray XT system. Use these presets to construct an event set as described in [Section 12.1.2](#), page 104.

Table 10. PAPI Presets

| <b>Name</b> | <b>Supported on Cray XT</b>                     | <b>Derived from multiple counters?</b> | <b>Description</b>               |
|-------------|-------------------------------------------------|----------------------------------------|----------------------------------|
| PAPI_L1_DCM | Yes                                             | No                                     | Level 1 data cache misses        |
| PAPI_L1_ICM | Yes                                             | No                                     | Level 1 instruction cache misses |
| PAPI_L2_DCM | Yes                                             | No                                     | Level 2 data cache misses        |
| PAPI_L2_ICM | Yes                                             | No                                     | Level 2 instruction cache misses |
| PAPI_L3_DCM | No for single- and dual-core; YES for quad-core | No                                     | Level 3 data cache misses        |
| PAPI_L3_ICM | No for single- and dual-core; YES for quad-core | No                                     | Level 3 instruction cache misses |
| PAPI_L1_TCM | Yes                                             | Yes                                    | Level 1 cache misses             |
| PAPI_L2_TCM | Yes                                             | No                                     | Level 2 cache misses             |
| PAPI_L3_TCM | No for single- and dual-core; YES for quad-core | No                                     | Level 3 cache misses             |

| <b>Name</b>  | <b>Supported on Cray XT</b>                     | <b>Derived from multiple counters?</b> | <b>Description</b>                                 |
|--------------|-------------------------------------------------|----------------------------------------|----------------------------------------------------|
| PAPI_CA_SNP  | No                                              | No                                     | Requests for a snoop                               |
| PAPI_CA_SHR  | No                                              | No                                     | Requests for exclusive access to shared cache line |
| PAPI_CA_CLN  | No                                              | No                                     | Requests for exclusive access to clean cache line  |
| PAPI_CA_INV  | No                                              | No                                     | Requests for cache line invalidation               |
| PAPI_CA_ITV  | No                                              | No                                     | Requests for cache line intervention               |
| PAPI_L3_LDM  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 load misses                                |
| PAPI_L3_STM  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 store misses                               |
| PAPI_BRU_IDL | No                                              | No                                     | Cycles branch units are idle                       |
| PAPI_FXU_IDL | No                                              | No                                     | Cycles integer units are idle                      |
| PAPI_FPU_IDL | No                                              | No                                     | Cycles floating-point units are idle               |
| PAPI_LSU_IDL | No                                              | No                                     | Cycles load/store units are idle                   |
| PAPI_TLB_DM  | Yes                                             | No                                     | Data translation lookaside buffer misses           |
| PAPI_TLB_IM  | Yes                                             | No                                     | Instruction translation lookaside buffer misses    |
| PAPI_TLB_TL  | Yes                                             | Yes                                    | Total translation lookaside buffer misses          |
| PAPI_L1_LDM  | Yes                                             | No                                     | Level 1 load misses                                |

| <b>Name</b>  | <b>Supported on Cray XT</b>                     | <b>Derived from multiple counters?</b> | <b>Description</b>                         |
|--------------|-------------------------------------------------|----------------------------------------|--------------------------------------------|
| PAPI_L1_STM  | Yes                                             | No                                     | Level 1 store misses                       |
| PAPI_L2_LDM  | Yes                                             | No                                     | Level 2 load misses                        |
| PAPI_L2_STM  | Yes                                             | No                                     | Level 2 store misses                       |
| PAPI_BTAC_M  | No                                              | No                                     | Branch target address cache misses         |
| PAPI_PRF_DM  | No                                              | No                                     | Data prefetch cache misses                 |
| PAPI_L3_DCH  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 data cache hits                    |
| PAPI_TLB_SD  | No                                              | No                                     | Translation lookaside buffer shutdowns     |
| PAPI_CSR_FAL | No                                              | No                                     | Failed store conditional instructions      |
| PAPI_CSR_SUC | No                                              | No                                     | Successful store conditional instructions  |
| PAPI_CSR_TOT | No                                              | No                                     | Total store conditional instructions       |
| PAPI_MEM_SCY | Yes                                             | No                                     | Cycles Stalled Waiting for memory accesses |
| PAPI_MEM_RCY | No                                              | No                                     | Cycles Stalled Waiting for memory reads    |
| PAPI_MEM_WCY | No                                              | No                                     | Cycles Stalled Waiting for memory writes   |
| PAPI_STL_ICY | Yes                                             | No                                     | Cycles with no instruction issue           |
| PAPI_FUL_ICY | No                                              | No                                     | Cycles with maximum instruction issue      |
| PAPI_STL_CCY | No                                              | No                                     | Cycles with no instructions completed      |

| <b>Name</b>  | <b>Supported on Cray XT</b> | <b>Derived from multiple counters?</b> | <b>Description</b>                                  |
|--------------|-----------------------------|----------------------------------------|-----------------------------------------------------|
| PAPI_FUL_CCY | No                          | No                                     | Cycles with maximum instructions completed          |
| PAPI_HW_INT  | Yes                         | No                                     | Hardware interrupts                                 |
| PAPI_BR_UCN  | Yes                         | No                                     | Unconditional branch instructions                   |
| PAPI_BR_CN   | Yes                         | No                                     | Conditional branch instructions                     |
| PAPI_BR_TKN  | Yes                         | No                                     | Conditional branch instructions taken               |
| PAPI_BR_NTK  | Yes                         | Yes                                    | Conditional branch instructions not taken           |
| PAPI_BR_MSP  | Yes                         | No                                     | Conditional branch instructions mis-predicted       |
| PAPI_BR_PRC  | Yes                         | Yes                                    | Conditional branch instructions correctly predicted |
| PAPI_FMA_INS | No                          | No                                     | FMA instructions completed                          |
| PAPI_TOT_IIS | No                          | No                                     | Instructions issued                                 |
| PAPI_TOT_INS | Yes                         | No                                     | Instructions completed                              |
| PAPI_INT_INS | No                          | No                                     | Integer instructions                                |
| PAPI_FP_INS  | Yes                         | No                                     | Floating-point instructions                         |
| PAPI_LD_INS  | No                          | No                                     | Load instructions                                   |
| PAPI_SR_INS  | No                          | No                                     | Store instructions                                  |
| PAPI_BR_INS  | Yes                         | No                                     | Branch instructions                                 |
| PAPI_VEC_INS | Yes                         | No                                     | Vector/SIMD instructions                            |
| PAPI_FLOPS   | Yes                         | Yes                                    | Floating-point instructions per second              |
| PAPI_RES_STL | Yes                         | No                                     | Cycles stalled on any resource                      |
| PAPI_FP_STAL | Yes                         | No                                     | Cycles in the floating-point unit(s) are stalled    |
| PAPI_TOT_CYC | Yes                         | No                                     | Total cycles                                        |
| PAPI_IPS     | Yes                         | Yes                                    | Instructions per second                             |

| <b>Name</b>  | <b>Supported on Cray XT</b>                     | <b>Derived from multiple counters?</b> | <b>Description</b>                     |
|--------------|-------------------------------------------------|----------------------------------------|----------------------------------------|
| PAPI_LST_INS | No                                              | No                                     | Load/store instructions completed      |
| PAPI_SYC_INS | No                                              | No                                     | Synchronization instructions completed |
| PAPI_L1_DCH  | Yes                                             | Yes                                    | Level 1 data cache hits                |
| PAPI_L2_DCH  | Yes                                             | No                                     | Level 2 data cache hits                |
| PAPI_L1_DCA  | Yes                                             | No                                     | Level 1 data cache accesses            |
| PAPI_L2_DCA  | Yes                                             | No                                     | Level 2 data cache accesses            |
| PAPI_L3_DCA  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 data cache accesses            |
| PAPI_L1_DCR  | No                                              | No                                     | Level 1 data cache reads               |
| PAPI_L2_DCR  | Yes                                             | No                                     | Level 2 data cache reads               |
| PAPI_L3_DCR  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 data cache reads               |
| PAPI_L1_DCW  | No                                              | No                                     | Level 1 data cache writes              |
| PAPI_L2_DCW  | Yes                                             | No                                     | Level 2 data cache writes              |
| PAPI_L3_DCW  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 data cache writes              |
| PAPI_L1_ICH  | No                                              | No                                     | Level 1 instruction cache hits         |
| PAPI_L2_ICH  | No                                              | No                                     | Level 2 instruction cache hits         |

| <b>Name</b> | <b>Supported on Cray XT</b>                     | <b>Derived from multiple counters?</b> | <b>Description</b>                 |
|-------------|-------------------------------------------------|----------------------------------------|------------------------------------|
| PAPI_L3_ICH | No for single- and dual-core; YES for quad-core | No                                     | Level 3 instruction cache hits     |
| PAPI_L1_ICA | Yes                                             | No                                     | Level 1 instruction cache accesses |
| PAPI_L2_ICA | Yes                                             | No                                     | Level 2 instruction cache accesses |
| PAPI_L3_ICA | No for single- and dual-core; YES for quad-core | No                                     | Level 3 instruction cache accesses |
| PAPI_L1_ICR | Yes                                             | No                                     | Level 1 instruction cache reads    |
| PAPI_L2_ICR | No                                              | No                                     | Level 2 instruction cache reads    |
| PAPI_L3_ICR | No for single- and dual-core; YES for quad-core | No                                     | Level 3 instruction cache reads    |
| PAPI_L1_ICW | No                                              | No                                     | Level 1 instruction cache writes   |
| PAPI_L2_ICW | No                                              | No                                     | Level 2 instruction cache writes   |
| PAPI_L3_ICW | No for single- and dual-core; YES for quad-core | No                                     | Level 3 instruction cache writes   |
| PAPI_L1_TCH | No                                              | No                                     | Level 1 total cache hits           |
| PAPI_L2_TCH | No                                              | No                                     | Level 2 total cache hits           |

| <b>Name</b>  | <b>Supported on Cray XT</b>                     | <b>Derived from multiple counters?</b> | <b>Description</b>                   |
|--------------|-------------------------------------------------|----------------------------------------|--------------------------------------|
| PAPI_L3_TCH  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 total cache hits             |
| PAPI_L1_TCA  | Yes                                             | Yes                                    | Level 1 total cache accesses         |
| PAPI_L2_TCA  | No                                              | No                                     | Level 2 total cache accesses         |
| PAPI_L3_TCA  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 total cache accesses         |
| PAPI_L1_TCR  | No                                              | No                                     | Level 1 total cache reads            |
| PAPI_L2_TCR  | No                                              | No                                     | Level 2 total cache reads            |
| PAPI_L3_TCR  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 total cache reads            |
| PAPI_L1_TCW  | No                                              | No                                     | Level 1 total cache writes           |
| PAPI_L2_TCW  | No                                              | No                                     | Level 2 total cache writes           |
| PAPI_L3_TCW  | No for single- and dual-core; YES for quad-core | No                                     | Level 3 total cache writes           |
| PAPI_FML_INS | Yes                                             | No                                     | Floating-point multiply instructions |
| PAPI_FAD_INS | Yes                                             | No                                     | Floating-point add instructions      |

---

| <b>Name</b>  | <b>Supported<br/>on<br/>Cray XT</b> | <b>Derived<br/>from<br/>multiple<br/>counters?</b> | <b>Description</b>                                                                                                   |
|--------------|-------------------------------------|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| PAPI_FDV_INS | No                                  | No                                                 | Floating-point divide instructions                                                                                   |
| PAPI_FSQ_INS | No                                  | No                                                 | Floating-point square root instructions                                                                              |
| PAPI_FNV_INS | Yes                                 | Yes                                                | Floating-point inverse instructions. This event is available only if you compile with the <code>-DDEBUG</code> flag. |

---

# MPI Error Messages [D]

---

Table 11 lists the MPI error messages you may encounter and suggested workarounds.

Table 11. MPI Error Messages

| Message                                                                                                                                               | Description                                                                                                                | Workaround                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Segmentation fault in<br>MPID_Init()                                                                                                                  | The application is using all the memory on the node and not leaving enough for MPI's internal data structures and buffers. | Reduce the amount of memory used for MPI buffering by setting the environment variable <code>MPICH_UNEX_BUFFER_SIZE</code> to something greater than 60 MB. If the application uses scalable data distribution, run your application at higher process counts.                                                                                                                                                                                            |
| MPIDI_PortalsU_Request_PUPE(323):<br>exhausted unexpected<br>receive queue buffering<br>increase via env. var.<br><code>MPICH_UNEX_BUFFER_SIZE</code> | The application is sending too many short, unexpected messages to a particular receiver.                                   | Increase the amount of memory for MPI buffering using the <code>MPICH_UNEX_BUFFER_SIZE</code> environment variable or decrease the short message threshold using the <code>MPICH_MAX_SHORT_MSG_SIZE</code> variable (default is 128 KB). The default for <code>MPICH_UNEX_BUFFER_SIZE</code> is 60,000,000 bytes. The <code>MPICH_UNEX_BUFFER_SIZE</code> environment variable specifies the entire amount of buffer space for short unexpected messages. |

| <b>Message</b>                                                                                                                                                                                                         | <b>Description</b>                                                                                                                                                                                                                                                                                                     | <b>Workaround</b>                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>pe_rank MPIDI_Portals_Progress:</code><br>dropped event on<br>unexpected receive queue,<br>increase <code>pe_rank</code> queue size by<br>setting the environment variable<br><code>MPICH_PTL_UNEX_EVENTS</code> | You have used up all the<br>space allocated for event<br>queue entries associated with<br>the unexpected messages<br>queue. The default size is<br>20,480 bytes.                                                                                                                                                       | You can increase the size of<br>the unexpected messages<br>event queue by setting<br>the environment variable<br><code>MPICH_PTL_UNEX_EVENTS</code><br>to a value higher than 20,480<br>bytes. |
| <code>pe_rank MPIDI_Portals_Progress:</code><br>dropped event on "other"<br>queue, increase <code>pe_rank</code><br>queue size by setting<br>the environment variable<br><code>MPICH_PTL_OTHER_EVENTS</code>           | You have used up all the<br>space allocated for the event<br>queue entries associated with<br>the "other" queue. This can<br>happen if the application is<br>posting many non-blocking<br>sends of large messages, or<br>many MPI-2 RMA operations<br>are posted in a single epoch.<br>The default size is 2048 bytes. | You can increase the size<br>of the queue by setting<br>the environment variable<br><code>MPICH_PTL_OTHER_EVENTS</code><br>to a value higher than<br>2048 bytes.                               |

---

# ALPS Error Messages [E]

---

This appendix documents common ALPS error messages. Other messages are generated only if a system error occurs. For all ALPS messages not described here, see your system administrator.

These messages are generated by the placement scheduler during application placement and are forwarded to the user through `aprun`.

Messages that begin with `[NID node_id]` come from the application shepherds on the compute nodes and are prefixed with a node ID to indicate which compute node sent the message. When general application failures occur, typically only one message appears from an arbitrary NID assigned to the application. This is done to prevent flooding the user with potentially thousands of identical messages if the application fails globally.

Table 12. ALPS Error Messages

| Error                                                                                                         | Description                                                                                                                                                                                                                                |
|---------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| no XT nodes are configured up                                                                                 | A request for the named type of compute node cannot be satisfied because there are no nodes of that type currently available.                                                                                                              |
| memory request exceeds 1048575 megabytes                                                                      | The <code>aprun -m</code> value exceeds the indicated amount. This is probably a mistake in units by the user because the value far exceeds any compute node memory size possible to install.                                              |
| Request exceeds max<br>[CPUs   memory   nodes]<br>In user NIDs request exceeds<br>max [CPUs   memory   nodes] | The allocation request requires more of the named resource than the configuration can deliver at this time. The second message will appear instead of the first if the user has specified the NIDs using the <code>aprun -L</code> option. |
| At least one command's user NID<br>list is short                                                              | If the <code>aprun -L</code> option is used, the NID list must have at least as many NID values as the number of nodes the application requires.                                                                                           |
| nid <i>node_id</i> appears more than<br>once in user's nid list                                               | The user has specified an NID list, but the list has at least one duplicate NID.                                                                                                                                                           |
| [NID <i>node_id</i> ] Apid <i>app_id</i> /proc<br>readdir timeout alarm occurred.<br>Application aborted.     | A problem on the node prevented the shepherd responsible for the application to read information from <code>/proc</code> as it must. Report this to the system administrator.                                                              |

| Error                                                                                                                                                    | Description                                                                                                                                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [NID <i>node_id</i> ] Apid <i>app_id</i> :<br>cannot execute: <i>reason</i>                                                                              | A large number of reasons can appear, but the most likely is <code>exec failed</code> , which usually means the <code>a.out</code> file is corrupted or is the wrong instruction set to run on this compute node.                                                                                          |
| [NID <i>node_id</i> ] Apid <i>app_id</i><br>killed. Received node failed<br>or unavailable event for nid<br><i>node_id</i>                               | The system monitoring software has detected an unrecoverable error on the named NID. Notification has been delivered to this NID for handling. The application must be killed because one or more of the compute nodes on which it is running have failed.                                                 |
| aprun: Exiting due to errors.<br>Launch aborted                                                                                                          | Typically, this is the final message from <code>aprun</code> before it terminates when an error has been detected. More detailed messages should precede this one.                                                                                                                                         |
| aprun: Apid <i>app_id</i> close of the<br>compute node connection [before<br>  after] app startup barrier                                                | The compute node to which <code>aprun</code> is connected has dropped its socket connection to <code>aprun</code> without warning. This usually means the application or a compute node has failed in some way that prevents normal error messages from being created or delivered to <code>aprun</code> . |
| aprun: Application <i>app_id</i><br>exit codes: <i>one to four values</i><br>aprun: Application <i>app_id</i> exit<br>signals: <i>one to four values</i> | If an application terminates with nonzero exit codes or has internally generated a signal (such as a memory address error), the first four of the values detected are reported with these messages. Both messages will appear if both nonzero exit codes and signals have occurred in the application.     |
| aprun: Application <i>app_id</i><br>resources: utime <i>uuu</i> , stime <i>sss</i>                                                                       | When the application terminates the accumulated user time ( <code>utime</code> ) and system time ( <code>stime</code> ) are forwarded to <code>aprun</code> and reported with this message.                                                                                                                |

---

# yod Error Messages [F]

Table 13 describes yod error messages.

Table 13. yod Error Messages

| Error                | Number | Description                                                                                 |
|----------------------|--------|---------------------------------------------------------------------------------------------|
| ERR_NO_MEMORY        | 1      | Out of memory in yod.                                                                       |
| ERR_USAGE            | 2      | Command-line usage error.                                                                   |
| ERR_HOST_INIT        | 3      | Error in host_cmd_init due to out of memory or portals. yod internal initialization failed. |
| ERR_MESH_ALLOC       | 8      | Call to mesh_alloc failed. Error during mesh initialization.                                |
| ERR_LOAD             | 9      | Load error. Cannot load program.                                                            |
| ERR_ABORT            | 10     | User aborted yod during load of program.                                                    |
| LD_ERR_SEND          | 10     | Error while sending data to children in fan-out tree.                                       |
| LD_ERR_NO_HEAP       | 10     | Error allocating heap memory on node.                                                       |
| LD_ERR_TARGET_LENGTH | 10     | Target supplied location too small for message to be sent.                                  |
| ERR_LOAD_FILE        | 13     | Load-file error. Error in use of heterogeneous load file.                                   |
| ERR_YOD_USAGE        | 14     | General yod usage error.                                                                    |
| ERR_KILL             | 23     | Application was killed. yod got killed after load.                                          |
| ERR_TARGET           | 26     | Invalid target option; valid targets are linux and catamount.                               |
| ERR_TIME_LIMIT       | 27     | yod time limit expired.                                                                     |
| ERR_PREMATURE_EXIT   | 28     | yod received CMD_EXIT too soon. A process exited prematurely.                               |
| ERR_ALARM            | 29     | Load time-out. Alarm signal.                                                                |
| ERR_RCA              | 30     | RCA register failed.                                                                        |
| LD_ERR_ABORTED       | 100    | Aborted load.                                                                               |
| LD_ERR_START         | 100    | First load error.                                                                           |

| <b>Error</b>              | <b>Number</b> | <b>Description</b>                                    |
|---------------------------|---------------|-------------------------------------------------------|
| LD_ERR_NUMNODES           | 101           | Number of nodes was outside of range allowed.         |
| LD_ERR_INTERNAL           | 102           | Internal error.                                       |
| PCT_LD_ERR_CONTROL_PORTAL | 103           | Error on control portal.                              |
| LD_ERR_TARGET_RANK        | 105           | Rank of requesting node is out of expected range.     |
| LD_ERR_TARGET_PORTAL      | 106           | Target portal number is out of expected range.        |
| LD_ERR_PULL               | 108           | Error while pulling data from parent in fan-out tree. |
| LD_ERR_VERSION            | 110           | Version mismatch.                                     |
| LD_ERR_NODE_TIMEOUT       | 111           | Time-out while communicating with node.               |
| LD_ERR_PORTALS_UID        | 112           | Portals UID mismatch.                                 |
| LD_ERR_PROTOCOL_ERROR     | 113           | General load-protocol error.                          |
| LD_ERR_BAD_PCT_MSG_TYPE   | 114           | Unexpected message type.                              |
| LD_ERR_EXEC_LOAD          | 115           | Error loading executable file.                        |
| LD_ERR_WRONG_NID          | 116           | Received response from wrong node ID.                 |
| LD_ERR_WRONG_RECV_LENGTH  | 117           | Received load with wrong length.                      |
| LD_ERR_PCT_EXIT           | 118           | PCT exited during load.                               |
| LD_ERR_NIDPID             | 119           | Node ID map was built or distributed incorrectly.     |
| ERROR_PCT_FAULT           | 120           | PCT fault.                                            |
| ERROR_SET_CACHE           | 121           | PCT failed to initialize processor.                   |
| ERROR_INIT_REGION         | 122           | PCT failed to initialize memory region.               |
| ERROR_APP_TIMER           | 123           | Application Timer Error.                              |
| ERROR_NO_MEM              | 124           | Out of memory on node.                                |
| ERROR_NO_MEM_FOR_BSS      | 125           | Text size is too big.                                 |
| ERROR_NO_MEM_FOR_HEAP     | 126           | Not enough memory for heap on node.                   |
| ERROR_NO_MEM_FOR_PROCESS  | 127           | Not enough memory for process.                        |
| ERROR_HEAP_SIZE_TOO_SMALL | 128           | Heap size is too small on node.                       |
| ERROR_NO_SMP              | 129           | Catamount virtual node mode is unavailable.           |
| ERROR_VA_OVERLAP          | 130           | Virtual addresses overlap kernel/PCT addresses.       |
| ERROR_PRIORITY            | 131           | PCT could not set processor priority.                 |

---

| <b>Error</b>                 | <b>Number</b> | <b>Description</b>                        |
|------------------------------|---------------|-------------------------------------------|
| ERROR_PORTALS                | 132           | Portals Error.                            |
| ERROR_BAD_ELF_FILE           | 133           | Bad ELF file.                             |
| ERROR_ELF_DYNAMIC_LOAD       | 134           | No dynamic load support for ELF files.    |
| ERROR_ELF_GENERIC            | 135           | ELF file error.                           |
| ERROR_INVALID_TARGET         | 136           | Invalid target.                           |
| ERROR_MSG_RCV_CACHE_OVERFLOW | 137           | Overflow in message received cache.       |
| ERROR_TOO_MANY_PARAMS        | 138           | Too many parameters passed to application |
| ERROR_TOO_MANY_PORTALS       | 139           | Too many portals were allocated.          |
| ERROR_TOO_MANY_PROCS         | 140           | Too many processes.                       |

---



# PETSc Makefiles [G]

---

The PETSc example (Section 14.6, page 123) includes `makefile.F` for program `ex2f.F`. This appendix lists the comparable `makefile.c` and `makefile_conventional.c` makefiles for program `ex2.c` and the `makefile_conventional.F` makefile for program `ex2f.F`.

For the source code of `ex2.c`, see <http://www-unix.mcs.anl.gov/petsc/petsc-as/>.

## **makefile.c**

```
.SUFFIXES: .c .o

Compilers, linkers and flags.
CC = cc
LINKER = ftn
CCFLAGS =
LINKFLAGS = -Mnomain (only for pgi)

Fortran optimization options.
COPTFLAGS = -O3

.o.c:
$(CC) -c ${COPTFLAGS} ${CCFLAGS} $*.c

all : ex2
ex2 : ex2.o
$(LINKER) $(LINKFLAGS) -o $@ ex2.o
```

## **makefile\_conventional.c**

```
include ${PETSC_DIR}/bmake/common/base

ex2: ex2.o
 -${CLINKER} -o ex2 ex2.o ${PETSC_KSP_LIB}
 ${RM} ex2.o
```

**makefile\_conventional.F**

```
include ${PETSC_DIR}/bmake/common/base
```

```
ex2f: ex2f.o
```

```
 -${FLINKER} -o ex2f ex2f.o ${PETSC_KSP_LIB}
```

```
 ${RM} ex2f.o
```

# Glossary

---

## **blade**

1) A field-replaceable physical entity. A Cray XT service blade consists of AMD Opteron sockets, memory, Cray SeaStar chips, PCI-X or PCIe cards, and a blade control processor. A Cray XT compute blade consists of AMD Opteron sockets, memory, Cray SeaStar chips, and a blade control processor. A Cray X2 compute blade consists of eight Cray X2 chips (CPU and network access links), two voltage regulator modules (VRM) per CPU, 32 memory daughter cards, a blade controller for supervision, and a back panel connector. 2) From a system management perspective, a logical grouping of nodes and blade control processor that monitors the nodes on that blade.

## **Catamount**

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

## **Catamount Virtual Node (CVN)**

The Catamount kernel enhanced to run on dual-core Cray XT compute nodes.

## **class**

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

## **CNL**

CNL is a Cray XT and Cray X2 compute node operating system. CNL provides a set of supported system calls. CNL provides many of the operating system functions available through the service nodes, although some functionality has been removed to improve performance and reduce memory usage by the system.

## **compute node**

A node that runs application programs. A compute node performs only computation; system services cannot run on compute nodes. Compute nodes run a specified kernel to support either scalar or vector applications. See also *node*; *service node*.

**compute processor allocator (CPA)**

A program that coordinates with yod to allocate processing elements.

**Cray Linux Environment (CLE)**

The operating system for Cray XT systems.

**CrayDoc**

Cray's documentation system for accessing and searching Cray books, man pages, and glossary terms from a web browser.

**deferred implementation**

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

**dual-core processor**

A processor that combines two independent execution engines ("cores"), each with its own cache and cache controller, on a single chip.

**GNU Compiler Collection (GCC)**

From The Free Software Foundation, a compiler that supports C, C++, Objective-C, Fortran, and Java code (see <http://www.x.org/gcc/>).

**login node**

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

**module**

See *blade*.

**node**

For Cray Linux Environment (CLE) 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.

**NUMA node**

A multicore processor and its local memory. Multisocket compute nodes have two or more NUMA nodes.

**processing element**

The smallest physical compute group. There are 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 or PCIe links.

**quad-core processor**

A processor that combines four independent execution engines ("cores"), each with its own cache and cache controller, on a single chip.

**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.

**specialization**

The process of setting files on the shared-root file system so that unique files can exist for a node or for a class of nodes.

**system interconnection network**

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

**TLB**

A table (Translation Lookaside Buffer) in the processor that contains cross-references between the virtual and real addresses of recently referenced pages of memory.



64-bit library  
  PathScale, 29  
  PGI, 27

## A

Accounts, 83  
ACML, 2, 19  
  required PGI linking option, 51  
ALPS, 61  
AMD Core Math Library, 19  
APIs, 13  
apkill command, 33  
Application  
  launching, 61  
Applications  
  launching, 61, 77  
  running in parallel, 115, 157  
  running on service nodes, 85  
aprun  
  CPU affinity options, 111  
  I/O handling, 74  
  launching applications, 61  
  memory affinity options, 110  
aprun command, 3, 61  
Authentication, 7

## B

Barcelona, 33  
Base pages, 35  
Batch job  
  submitting through PBS Professional, 87  
Batch processing, 4  
BLACS, 2, 13–14  
BLAS, 2, 13, 19  
Buffering  
  Fortran I/O, 42

## C

C compiler, 2  
C++ compiler, 2  
C++ I/O  
  changing default buffer size, 43  
  specifying a buffer, 43  
Catamount  
  C run time functions in, 193  
  C++ I/O, 43  
  glibc functions supported, 40, 193  
  I/O, 42  
  I/O handling, 82  
  programming considerations, 40  
  signal handling, 82  
  stderr, 42  
  stdin, 42  
  stdout, 42  
Catamount nodes  
  report showing status, 57  
Catamount Virtual Node (CVN), 78  
CLE  
  CNL, 1  
CNL, 1, 61  
  C run time functions in, 187  
  glibc functions supported, 187  
  I/O, 31  
  I/O handling, 74  
  programming considerations, 27, 30  
  stderr, 31  
  stdin, 31  
  stdout, 31  
CNL applications  
  requesting resources, 61  
CNL jobs, 61  
CNL nodes  
  report showing status, 57  
cselect command, 3  
Compiler

- C, 2
- C++, 2
- Fortran, 2
- Compiler commands, 49
- Compute node kernel
  - report showing status, 57
- Compute node operating system
  - Catamount, 1
  - CNL, 1
- Compute nodes
  - managing from an MPI program, 74, 82, 89
  - selecting, 3
- Compute Processor Allocator (CPA), 77
- Core file, 47
  - truncated, 47
- CPU affinity, 111
  - aprun options, 111
- CPU binding, 111
- cpusets, 110–111
- CRAFFT library, 16
- Cray Adaptive FFT (CRAFFT) library, 16
- Cray Apprentice2, 4, 108
- Cray Linux Environment (CLE), 1
  - Catamount, 1
- Cray MPICH2, 2, 21
  - limitations, 21
- Cray SHMEM, 22
  - atomic memory operations, 22
- Cray XT-LibSci, 2, 13
- Cray XT5 compute nodes, 34
  - memory affinity, 110
- CrayPat, 4, 104

## D

- Debugging, 93
  - GNU debugger, 99
  - lgdb, 93
  - using TotalView, 93
  - xtgdb, 93
- Documentation, 4
- Dual-core processor, 78
- Dynamic linking, 30

## E

### ELF

*See Executable and Linking Format*

### Endian

*See Little endian*

### Event set

how to create in PAPI, 104

### Example

basics (Catamount), 157

basics (CNL), 115

buffer for I/O (Catamount), 166

I/O-to-file streams (Catamount), 167

multiple parallel apps (Catamount), 174

multiple parallel apps (CNL), 142

multiple sequential apps (Catamount), 172

multiple sequential apps (CNL), 140

stdout (Catamount), 169

using Cray Apprentice2 (CNL), 156

using CrayPat (Catamount), 179

using CrayPat (CNL), 150

using `dclock()` (Catamount), 165

using MPI (Catamount), 158

using MPI (CNL), 116

using OpenMP (CNL), 135

using PAPI high-level interface  
(Catamount), 176

using PAPI high-level interface (CNL), 148

using PAPI low-level interface (Catamount), 178

using PAPI low-level interface (CNL), 149

using PBS interactive mode (CNL), 138

using PBS script (Catamount), 171

using PBS script (CNL), 139

using PETSc (CNL), 123

using quad-core processors (CNL), 135

using `shmem_get` (Catamount), 162

using `shmem_get` (CNL), 120

using `shmem_put` (Catamount), 160

using `shmem_put` (CNL), 118

using UPC (Catamount), 164

using UPC (CNL), 122

using `xtgdb` (Catamount), 175

### Examples

- 
- Catamount, 157
  - CNL, 115
    - running on service nodes, 85
  - Executable
    - launching, 61
  - Executable and Linking Format (ELF), 61
  - F**
  - Fast\_mv library, 20
  - FFT, 2, 19
    - CRAFFT, 16
  - FFTW, 3, 19
  - File system
    - Lustre, 3, 11
  - Fortran compiler, 2
  - Fortran STOP message, 28
  - G**
  - GCC compilers, 2, 49, 52
    - running on service nodes, 85
    - using OpenMP, 24
  - gdb debugger
    - See GNU debugger
  - getpagesize()
    - Catamount implementation of, 40
  - glibc, 3, 13
    - Catamount, 40
    - run time functions implemented in
      - Catamount, 193
      - run time functions implemented in CNL, 187
    - support in Catamount, 40
    - support in CNL, 30
  - GNU C library, 3, 13
  - GNU compilers, 49, 52
    - running on service nodes, 85
  - GNU debugger
    - Catamount applications, 99
    - CNL applications, 99
  - GNU Fortran libraries, 2
  - H**
  - Hardware counter presets
    - PAPI, 199
  - Hardware performance counters, 104
  - Huge pages, 35
  - I**
  - I/O
    - stdio performance, 43
    - stride functions, 44
  - I/O buffering
    - IOBUF library, 43
  - I/O performance
    - Fortran buffer size, 42
  - I/O support in Catamount, 42
  - I/O support in CNL, 31
  - Instrumenting a program, 104
  - IRT
    - See Iterative Refinement Toolkit
  - Iterative Refinement Toolkit, 2, 13, 15
  - J**
  - Job accounting, 83
  - Job launch
    - MPMD application, 73
  - Job scripts, 87
  - Job status, 90
  - Jobs
    - running on Catamount, 77
    - running on CNL, 61
  - K**
  - kill command (CNL), 33
  - kill() system call (Catamount), 47
  - kill() system call (CNL), 33
  - L**
  - LAPACK, 2, 13, 19
  - Launching Catamount applications, 77
  - Launching CNL applications, 61
  - Launching jobs
    - using aprun, 3
    - using yod, 3
  - LD\_PRELOAD environment variable, 30

lgdb debugger  
  See GNU debugger

Libraries, 13

Library

  ACML, 2, 19

  BLACS, 2, 13–14

  BLAS, 2, 13, 19

  CRAFFT, 16

  Cray MPICH2, 21

  Cray XT-LibSci, 13

  Fast\_mv, 20

  FFT, 2, 19

  FFTW, 3

  glibc, 13

  GNU C, 3

  Iterative Refinement Toolkit, 2, 15

  LAPACK, 2, 13, 19

  LibSci, 2

  PETSc, 2, 17

  ScaLAPACK, 2, 13–14

  SuperLU, 2, 13, 16

LibSci

  See Cray XT-LibSci

Little endian, 30

Loadfile

  launching MPMD applications with, 80

Lustre, 3

  programming considerations, 11

Lustre library, 11

## M

malloc(), 41

  Catamount implementation of, 40

Man pages, 4

Manuals, 4

Math transcendental library routines, 2, 19

Memory affinity, 110

  aprun options, 110

Memory allocation

  improving, 37

Message passing, 21

Message Passing Interface, 2

MMAP (memory-mapped regions)

  alternatives to, 37

  module command, 10

Modules, 9

MPI, 2, 21

  64-bit library, 27, 29

  managing compute nodes from, 74, 82, 89

  running program interactively, 115

MPICH2

  limitations, 21

MPMD applications

  using aprun, 73

  using yod, 80

## N

Node

  availability, 57

NUMA nodes, 34

## O

OpenMP, 2, 24

Optimization, 109

  CPU affinity, 111

  memory affinity, 110

PAPI, 103

  counter presets for constructing an event

  set, 199

P high-level interface, 103

Page size, 85, 87

Parallel programming models

  MPI, 39

  MPICH2, 2

  OpenMP, 2, 39

  SHMEM, 39

  supported on Catamount, 48

  UPC, 25, 39

passwordless logins, 7

passwordless ssh, 7

passwords, 7

- 
- PATH variable
    - how to modify, 10
  - PathScale compilers, 2, 53
    - running on service nodes, 85
    - using OpenMP, 24
  - PBS Professional, 4, 87
  - Performance analysis
    - Cray Apprentice2, 108
    - CrayPat, 104
    - PAPI, 103
  - Performance API (PAPI), 3
  - PETSc, 2, 17
  - PETSc makefiles, 215
  - PGI compilers, 2, 49–50
    - limitations, 27
    - running on service nodes, 85
    - unsupported options, 48
    - using OpenMP, 24
  - Portals interface, 21
  - Process Control Thread (PCT), 77
  - Process migration, 111
  - Programming considerations
    - Catamount, 27
    - CNL, 27
    - general, 27
  - Programming Environment, 2
  - Project accounting, 83
- Q**
- qdel command, 91
  - qstat command, 90
  - qsub command, 88
  - Quad-core processors, 33
    - memory affinity, 110
    - using cnselect, 71
- R**
- Random number generators, 2, 19
  - RSA authentication, 7
    - with passphrase, 7
    - without passphrase, 8
- Running applications
    - using aprun, 3
    - using yod, 3
  - Running Catamount applications, 77
  - Running CNL applications, 61
- S**
- ScaLAPACK, 2, 13–14
  - Scientific libraries, 13
  - Scripts
    - PBS Professional, 87
  - Secure shell, 7
  - Service nodes
    - running user programs, 85
  - Shared libraries, 30
  - SHMEM, 2
    - 64-bit library, 27, 29
  - Signal handling, 46, 82
  - Single-core processor, 77
  - Sockets
    - Cray XT5 compute nodes, 34
  - ssh, 7
  - stderr, 31, 42
  - stdin, 31, 42
  - stdio
    - performance, 43
  - stdout, 31, 42
  - STOP message, 28
  - SuperLU, 2, 13, 16
- T**
- Timers
    - Catamount support for, 40
  - Timing measurements, 45
  - TotalView, 93
    - Cray specific functions, 99
- U**
- Unified Parallel C (UPC), 25
  - User environment
    - setting up, 7

## X

xtgdb debugger

*See* GNU debugger

xtkill command, 33, 47

xtnodestat command, 3, 57

xtproadmin command, 57

## Y

yod, 77

    I/O handling, 82

yod command, 3