



Sun HPC ClusterTools™ 5 Software User's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 817-0084-10
February 2003, Revision A

Send comments about this document to: docfeedback@sun.com

Copyright 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, AnswerBook2, docs.sun.com, Solaris, Sun HPC ClusterTools, Prism, Sun ONE Studio, Sun Performance Library, Sun Fire, Sun Cluster, RSM, and UltraSPARC are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and in other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the Sun Microsystems, Inc. license agreements and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct. 1998), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, AnswerBook2, docs.sun.com, Solaris, Sun HPC ClusterTools, Prism, Sun ONE Studio, Sun Performance Library, RSM, Sun Fire, Sun Cluster, et UltraSPARC sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

Preface xi

- 1. Introduction to Sun HPC ClusterTools Software** 1
 - Supported Configurations 1
 - Sun HPC ClusterTools Runtime Environment (CRE) 2
 - Executing Programs With `mprun` 2
 - Killing Programs 2
 - Displaying Job Information 2
 - Displaying Node Information 3
 - Integration With Distributed Resource Management Systems 3
 - Sun MPI and MPI I/O 3
 - Prism 4
 - Support for TotalView 4
 - Sun S3L 5
 - MPPProf 6
- 2. Fundamental Concepts** 7
 - Clusters and Nodes 7
 - Partitions 8
 - How Partitions Are Enabled and Selected 8

Load Balancing	10
Processes	10
Jobs	10
How the CRE Environment Is Integrated With Distributed Resource Management Systems	11
How Programs Are Launched	12
How Distributed Resource Managers Work	13
3. Before You Begin	15
Prerequisites	15
Command and Man Page Paths	15
Authentication Methods	16
Core Files	16
4. Running Programs With <code>mprun</code>	17
Syntax	17
Controlling Where the Program Runs	20
Precedence for Program Execution	20
▼ How to Run a Program With Default Settings	21
▼ How to Run on a Different Cluster (<code>-c</code>)	21
▼ How to Run on a Different Partition (<code>-p</code>)	21
▼ How to Run as Multiple Processes (<code>-np</code>)	22
▼ How to Share Nodes (<code>-j</code>)	23
▼ How to Enable Process Spawning (<code>-ys</code>)	23
▼ How to Disable Process Spawning (<code>-Ns</code>)	23
▼ How to Wrap Multiple Processes (<code>-w</code>)	24
▼ How to Settle for Available Processes (<code>-s</code>)	24
▼ How to Include Independent Nodes (<code>-u</code>)	25
▼ How to Combine Process Placement Options	26
Mapping MPI Processes to Nodes	27

- ▼ How to Distribute Processes Among Nodes (-l) 27
- ▼ How to Distribute Processes by Block
(-z and -zt) 29
- ▼ How to Distribute Processes by Rank Map (-m) 30
- ▼ How to Reserve Resources For Spawning or Multithreading (-nr) 31
- ▼ How to Select Nodes by Resource Requirement (-R) 32

Controlling Input/Output 38

- ▼ How to Redirect Output to mprun (-D) 39
- ▼ How to Redirect Output to Individual Files (-B) 39
- ▼ How to Shut Off All Standard I/O (-N) 40
- ▼ How to Redirect With an Argument Vector (-A) 40
- ▼ How to Read Standard Input From /dev/null (-n) 41
- ▼ How to Redirect With a Custom Configuration (-I) 41

Controlling Other Job Attributes 48

- ▼ How to Include Shell-Specific Actions 48
- ▼ How to Move a Process to the Background 49
- ▼ How to Change the Working Directory (-C) 49
- ▼ How to Use a Different User Name (-U) 49
- ▼ How to Use a Different Group Name (-G) 50
- ▼ How to Run a Job on a Different Project (-P) 50
- ▼ How to Specify Verbose Output (-v) 50
- ▼ How to Display Command Help (-h) 51
- ▼ How to Display the Command's Version (-V) 52
- ▼ How to Display Job Status Information (-J) 52
- ▼ How to Store Job Name in a File (-d) 52
- ▼ How to Tag Output With Its Rank Number (-o) 52

Command Reference (mprun) 53

5. Running Programs With mprun in Distributed Resource Management Systems 55

	mpirun Options for DRM Integration	55
	Improper Flag Combinations for Batch Jobs	57
	Running Parallel Jobs in the PBS Environment	57
	▼ How to Run an Interactive Job in PBS	58
	▼ How to Run a Script Job in PBS	59
	Running Parallel Jobs in the LSF Environment	60
	▼ How to Run an Interactive Job in LSF	60
	▼ How to Run a Script Job in LSF	62
	▼ How to Run an LSF Job in Compatibility Mode	62
	Running Parallel Jobs in the SGE Environment	63
	▼ How to Run an Interactive Job in SGE	63
	▼ How to Run a Script Job in SGE	64
6.	Killing or Sending Signals to Programs With <code>mpkill</code>	67
	What You Can Do	67
	Return Values	67
	▼ How to Kill a Running Program	68
	▼ How to Remove All Traces of a Job	68
	▼ How to Display a List of Supported Signals (-l -d)	68
	▼ How to Send a Signal to a Job	69
7.	Displaying Program Information With <code>mpps</code>	71
	What You Can Do	71
	▼ How to Display Job Status	72
	▼ How to Display Information About Individual Jobs (-J)	73
	▼ How to Display Job Name, PID, and Host of Current Job (-b)	74
	▼ How to Display Information About All Jobs (-e)	74
	▼ How to Display a Job's Start Time (-f)	74

- ▼ How to Display Job Information by Partition
(-A -a) 74
- ▼ How to Display Job Information by Process
(-p -P) 75

Command Reference (mpps) 76

8. Profiling Programs With MPPROF 77

Enabling MPI Profiling 77

Controlling Data Collection 78

MPI_PROFDATADIR 78

MPI_PROFINDEXFDIR 78

MPI_PROFINTERVAL 79

MPI_PROFMAXFILESIZE 79

Using mpprof to Generate Reports 80

mpprof Command Syntax 80

Generating a Message Passing Report 82

Reporting on Specific Processes 82

Reporting Processes That Occur After a Specified Time Interval 82

To Save Report Output for Later Use 83

A Sample Report 83

Using mpdump to Convert Intermediate Binary Files to ASCII Files 88

The mpdump Command Syntax 88

A Sample mpdump File 89

9. Displaying Information With mpinfo 93

What You Can Do 93

- ▼ How to Display Information About Published Names (-T) 94

- ▼ How to Display Information About Any Cluster (-c) 94

- ▼ How to Display Information About the Current Cluster (-C) 95

- ▼ How to Display Information About Individual Partitions (-p) 96
- ▼ How to Display Information About All Partitions (-P) 96
- ▼ How to Display Information About Individual Nodes (-n) 97
- ▼ How to Display Information About All Nodes (-N) 98
- ▼ How to Display an Online List of Valid Attributes (-lc, -lp, -ln) 98
- ▼ How to Restrict Output to Individual Attributes (-A) 99
- ▼ How to Display Information in Verbose Mode (-v) 102

Command Reference (mpinfo) 104

Index 113

Tables

TABLE 3-1	User Commands Required by Authentication Methods	16
TABLE 4-1	Combining <code>mprun</code> Process Placement Options	26
TABLE 4-2	Predefined Resources	33
TABLE 4-3	RRS Operators	34
TABLE 4-4	<code>mprun</code> I/O Shortcut Summary	47
TABLE 4-5	Options for <code>mprun</code>	53
TABLE 6-1	Options for <code>mpkill</code>	69
TABLE 7-1	Job Status Displayed by <code>mpps</code>	72
TABLE 7-2	Job attributes for <code>-J</code> option to <code>mpps</code>	73
TABLE 7-3	Process attributes for <code>-P</code> option to <code>mpps</code>	75
TABLE 7-4	Options for <code>mpps</code>	76
TABLE 8-1	<code>mpprof</code> Command Options	80
TABLE 8-2	Options to the <code>mpdump</code> Command	89
TABLE 9-1	Attributes Displayed by <code>-A</code> option to <code>mpinfo</code>	100
TABLE 9-2	Options for <code>mpinfo</code>	104
TABLE A-1	Sun MPI Standard Error Classes	107
TABLE A-2	Sun MPI I/O Error Classes	109

Preface

The manual explains the use of select Distributed Resource Management packages for effective resource management and utilization accounting in conjunction with the Sun MPI parallel applications:

- Sun Grid Engine SGE Version 5.3 and Sun Grid Engine Enterprise Edition SGEEE Version 5.3
- Load Sharing Facility Version 4.x of Platform Computing
- Portable Batch System (PBS) PBS Pro 5.x.x of Veridian

Before You Read This Book

Release notes for the other components in the suite are included in *Sun HPC ClusterTools™ 5 Software Release Notes*. For information about writing MPI programs, refer to the *Sun MPI 6.0 Software Programming and Reference Guide*. For information about a specific distributed resource management package, refer to the documentation supplied with that package.

Using UNIX Commands

This document might not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- AnswerBook2™ online documentation for the Solaris™ software environment

- Other software documentation that you received with your system

Typographic Conventions

Typeface*	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
AaBbCc123	What you type, when contrasted with on-screen computer output	% su Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this. To delete a file, type <code>rm filename</code> .

* The settings on your browser might differ from these settings.

Table with descriptions and examples of the typographic conventions that are used in this book.

Shell Prompts

Shell	Prompt
C shell	<i>machine-name</i> %
C shell superuser	<i>machine-name</i> #
Bourne shell and Korn shell	\$
Bourne shell and Korn shell superuser	#

Table with examples of the types of shell prompts that are used in this book.

Related Documentation

This book focuses on Sun MPI and assumes familiarity with the *MPI Standard*. The following materials provide useful background about using Sun MPI and about the *MPI Standard*.

Application	Title	Part Number
Sun HPC ClusterTools Documentation	<i>Read Me First: Guide to Sun HPC ClusterTools Software Documentation</i>	817-0096-10
Sun HPC ClusterTools Software	<i>Sun HPC ClusterTools 5 Software Release Notes</i>	817-0081-10
	<i>Sun HPC ClusterTools 5 Software Installation Guide</i>	817-0082-10
	<i>Sun HPC ClusterTools 5 Software Performance Guide</i>	817-0090-10
	<i>Sun HPC ClusterTools 5 Software Administrator's Guide</i>	817-0083-10
Sun MPI Programming	<i>Sun MPI 6.0 Software Programming and Reference Guide</i>	817-0085-10
Sun S3L	<i>Sun S3L 4.0 Software Programming Guide</i>	817-0086-10
	<i>Sun S3L 4.0 Software Reference Manual</i>	817-0087-10
Prism™ graphical programming environment	<i>Prism 7.0 Software User's Guide</i>	817-0088-10
	<i>Prism 7.0 Software Reference Manual</i>	817-0089-10

Table listing other documents that are related to this book or product.

In addition, if you are using Platform Computing's Load Sharing Facility (LSF) Suite, consult the documentation available from their website:

<http://www.platform.com>

Accessing Sun Documentation

You can view, print, or purchase a broad selection of Sun documentation, including localized versions, at:

<http://www.sun.com/documentation>

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can email your comments to Sun at:

`docfeedback@sun.com`

Please include the part number (817-0084-10) of your document in the subject line of your email.

Introduction to Sun HPC ClusterTools Software

Sun HPC ClusterTools 5 software is a set of parallel development tools that extend Sun's network computing solutions to high-end distributed-memory applications. This chapter summarizes its required configuration and principal components. It has the following sections:

- “Supported Configurations” on page 1
- “Sun HPC ClusterTools Runtime Environment (CRE)” on page 2
- “Integration With Distributed Resource Management Systems” on page 3
- “Sun MPI and MPI I/O” on page 3
- “Prism” on page 4
- “Support for TotalView” on page 4
- “Sun S3L” on page 5

Supported Configurations

Sun HPC ClusterTools 5 software requires the Solaris 8 (32-bit or 64-bit) and Solaris 9 operating environments. All programs that execute under the Solaris 8 and Solaris 9 operating environments will execute in the Sun HPC ClusterTools environment.

Sun HPC ClusterTools 5 software supports Forte Developer 6, update 2 and Sun One Studio 7 Compiler Collection for C, C++, and Fortran compilers.

Sun HPC ClusterTools 5 software can run MPI jobs of up to 2048 processes on as many as 256 nodes. It also provides load-balancing and support for spawning MPI processes.

For high-performance clusters, the preferred interconnect technology will be the Sun Fire™ Link interconnect. The Sun HPC ClusterTools software also runs on clusters connected via any TCP/IP-capable interconnect, such as Ethernet, high-speed Ethernet, Gigabit Ethernet, ATM OC-3, ATM OC-12, FDDI, and HIPPI.

Sun HPC ClusterTools Runtime Environment (CRE)

Sun HPC ClusterTools 5 software provides a command line interface (also called CRE for *ClusterTools Runtime Environment*) that starts jobs and provides status information. It performs four primary operations:

- Executes programs
- Kills programs
- Displays job information
- Displays node information

Each of these operations is summarized below. Instructions appear in subsequent chapters.

Executing Programs With `mprun`

Sun HPC ClusterTools 5 software can start both serial and parallel jobs. It is particularly useful for balancing computing load in serial jobs executed across shared partitions, where multiple processes can be competing for the same node resources. The syntax and use of `mprun` are described in Chapter 4.

Killing Programs

The runtime environment uses the `mpkill` command to kill jobs in progress and send them signals. Its syntax and use are described in Chapter 6.

Displaying Job Information

The runtime environment uses the `mpps` command to display information about jobs and their processes. Its syntax and use are described in Chapter 7.

Displaying Node Information

The runtime environment uses the `mpinfo` command to display information about nodes and their partitions. Its syntax and use are described in Chapter 9.

Integration With Distributed Resource Management Systems

Sun HPC ClusterTools 5 software provides new integration facilities with three select Distributed Resource Management systems for proper resource allocation, parallel job control and monitoring, as well as proper job accounting. These are:

- SGE: Codine 5.2
- LSF: LSF 4.x
- PBS: PBS 2.3.15 and PBS Pro 5.0

The support of other available DRM systems than stated above are possible through the use of open APIs. Please contact Sun representative for further information.

You can launch parallel jobs directly from these distributed resource management systems. The DRM interacts closely with Sun CRE for proper resource description and subsequent of the multiple processes comprising the requested parallel job.

For a description of the scalable and open architecture of the DRM integration facilities, see “How the CRE Environment Is Integrated With Distributed Resource Management Systems” on page 11. For instructions, see Chapter 5.

Sun MPI and MPI I/O

Sun MPI is a highly optimized version of the Message Passing Interface (MPI) communications library. It implements all of the *MPI 1.2 Standard* and the *MPI 2.0 Standard*. Its highlights are:

- Integration with the Sun HPC ClusterTools Runtime Environment (CRE)
- Support for multithreaded programming
- Seamless use of different network protocols; for example, code compiled on a Sun HPC cluster that has fast Ethernet network can be run without change on a cluster that has an ATM network

- Multiprotocol support so that MPI picks the fastest available medium for each type of connection (such as shared memory, fast Ethernet, or ATM)
 - Communication via shared memory for fast performance on clusters of SMPs
 - Optimized collectives for symmetric multiprocessors (SMPs) and clusters of SMPs
 - Full F77, C, and C++ support, and basic F90 support
-

Prism

Prism is a graphical programming environment to develop, execute, debug, and visualize data in multithreaded or nonthreaded message-passing programs. It enables you to:

- Control various aspects of program execution, such as starting and stopping, breakpoints and traces, displaying values of variables, expressions, and the call stack
- Visualize data in various formats
- Aggregate processes and threads across multiprocess parallel jobs into meaningful groups, called process sets or *psets*

You can use Prism with applications written in F77, F90, C, and C++.

Support for TotalView

TotalView from Etnus is a third-party multiprocess debugger that runs on many platforms. Support for using the TotalView debugger on Sun MPI applications includes:

- Making Sun HPC ClusterTools software compatible with the TotalView debugger
- Allowing Sun MPI jobs to be debugged by TotalView using SGE, PBS, and LSF
- Displaying Sun MPI message queues
- Enabling a single cluster to support multiple TotalView instances debugging different Sun MPI jobs simultaneously

Sun S3L

The Sun Scalable Scientific Subroutine Library (Sun S3L) provides a set of parallel and scalable functions and tools widely used in scientific and engineering computing. It is built on top of Sun MPI and provides the following functionality for MPI programmers:

- Vector and dense matrix operations (level 1, 2, 3 Parallel BLAS, version 1.0)
- Iterative solvers for sparse systems
- Matrix-vector multiply for sparse systems
- FFT
- LU factor and solve
- Finite-difference stock option pricing
- Autocorrelation
- Convolution/deconvolution
- Tridiagonal solvers
- Banded solvers
- Eigensolvers
- Singular value decomposition
- Least squares
- One-dimensional sort
- Multidimensional sort
- Selected ScaLAPACK (version 1.6) and BLACS application program interfaces
- Conversion between ScaLAPACK and S3L
- Matrix transpose
- Random number generators (linear congruential and lagged Fibonacci)
- Random number generator and I/O for sparse systems
- Matrix inverse
- Array copy
- Safety mechanism
- An array syntax interface callable from message-passing programs
- Toolkit functions for operations on distributed data
- Support for the multiple instance paradigm (allowing an operation to be applied concurrently to multiple, disjoint data sets in a single call)
- Thread safety
- Detailed programming examples and support documentation online

Sun S3L routines can be called from applications written in F77, F90, C, and C++.

MPPProf

MPPProf is a message-passing profiler intended for use with Sun MPI programs. It extracts information about calls to Sun MPI routines, storing the data in a set of intermediate files, one file per process. It then uses the intermediate data to generate a report profiling the program's message-passing activity.

MPPProf's data gathering operations are enabled by setting an environment variable before running the user program. If this environment variable is not set, program execution proceeds without generating profiling data. The MPPProf report generator is invoked with the command-line utility, `mppprof`. The report is an ASCII text file that provides the following types of information:

- The percentage of total execution time spent in MPI calls across all processes
- The percentage of time each process spent in MPI calls
- The number of calls, time spent, and bytes sent or received per MPI routine, averaged over all processes, with percent variation among processes
- Connectivity statistics (message count and volume) between processor pairs
- The settings of MPI environment variables that have performance implications

MPPProf also includes a data conversion utility, `mpdump`, which converts the intermediate data to user-readable ASCII files with the data in a raw (unanalyzed) state. You can then use the `mpdump` output files as input to a report generator, which you would supply in place of `mppprof`.

The MPPProf tool is best suited for code analysis situations where message-passing behavior is of primary interest and where simplicity and ease-of-use are also important. For a comprehensive analysis of a complex MPI program, you would need to use MPPProf in combination with other profiling tools. For example,

- Use a trace-history viewer, such as Vampir, to collect information about sequential relationships in message-passing events. This would complement MPPProf's focus on aggregate statistics.
- Use the Forte Developer Performance Analyzer for analyzing parts of the code other than the MPI routines.

Fundamental Concepts

This chapter summarizes a few basic concepts that you should understand to get the most out of Sun's HPC ClusterTools software. It contains the following sections:

- "Clusters and Nodes" on page 7
 - "Partitions" on page 8
 - "Load Balancing" on page 10
 - "Processes" on page 10
 - "Jobs" on page 10
 - "How the CRE Environment Is Integrated With Distributed Resource Management Systems" on page 11
 - "How Distributed Resource Managers Work" on page 13
-

Clusters and Nodes

High performance computing clusters¹ are groups of Sun symmetric multiprocessor (SMP) servers interconnected by any Sun-supported, TCP/IP-capable interconnect or by the Sun Fire™ Link high-speed interconnect. Each server in a cluster is called a *node*.

Note – A *cluster* can consist of a single Sun SMP server. However, to execute MPI jobs on even a single-node cluster, CRE must be running on that cluster.

When using CRE, you can select the cluster and nodes on which your MPI programs will run, and how your processes will be distributed among them. For instructions, see Chapter 4, "Running Programs With `mprun`."

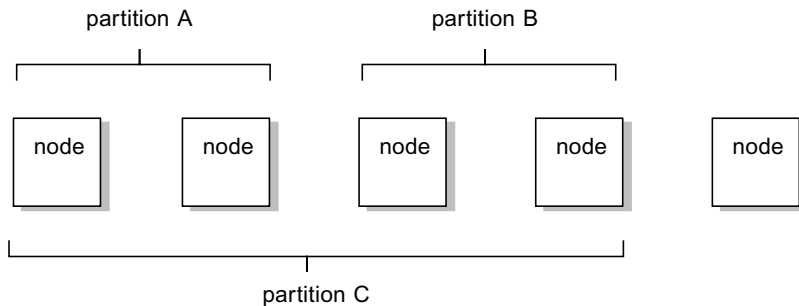
1. SunCluster™ is a completely different technology used for high availability (HA) applications.

Partitions

You can group a cluster's nodes into *partitions*. Partitions let you run different jobs simultaneously on different subsets of the cluster. You can also use partitions to create groups nodes of nodes with similar characteristics such as memory size, CPU count, or I/O support, so you can target jobs that benefit from those characteristics.

Note – The CPUs in the Sun Fire line of servers can be configured into “logical nodes,” called *domains*. You can also group these domains into CRE partitions.

You can define multiple partitions within a cluster.



Partitions do not have to include every node in the cluster. Nodes that are not included in any partition are called *independent* or *free-floating* nodes.

A single node can be included in more than one partition. However, two partitions with overlapping nodes cannot run jobs simultaneously. And only one of them can be enabled at a time. In the example above, partitions A and B can run jobs simultaneously with each other, but not with partition C.

How Partitions Are Enabled and Selected

A job can run only on a partition that has been *enabled*. Normally, the system administrator who manages the cluster enables and *disables* partitions (for more information, see the *Sun HPC ClusterTools Software Administrator's Guide*).

To find out which partitions are currently enabled, use the `-P` option to the `mpinfo` command, as described in “How to Display Information About All Partitions (`-P`)” on page 96.

If only one partition is enabled, all jobs must run on that partition. If multiple partitions are enabled, where your particular job runs depends upon which environment variables the cluster administrator set and which options to the `mpirun` command you entered. To determine the partition, CRE steps through the criteria shown in FIGURE 2-1, in order.

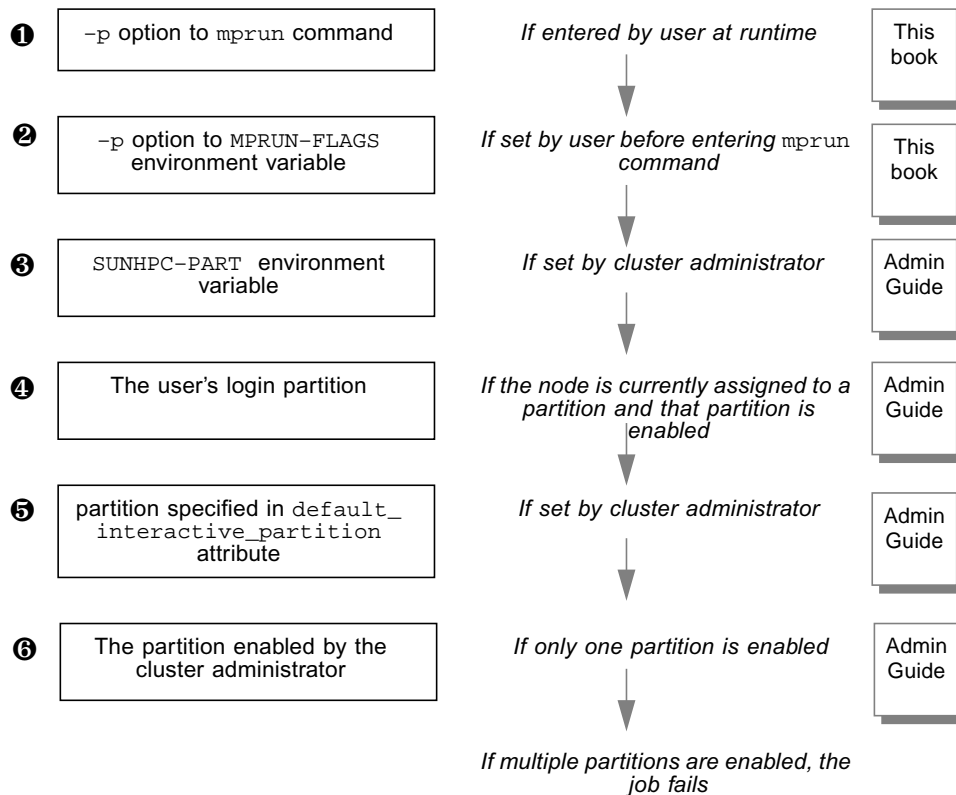


FIGURE 2-1 CRE's Partition Selection Criteria

Load Balancing

CRE load-balances programs when more CPUs are available than are required for a job. When you issue the `mprun` command to start a job, CRE first determines what criteria (if any) you have specified for the node or nodes on which the program is to run. It then determines which nodes within the partition meet these criteria. If more nodes meet the criteria than are required to run your program, CRE starts the program on the node or nodes that are least loaded. It examines the one-minute load averages of the nodes and ranks them accordingly.

This load-balancing mechanism ensures that your program's execution will not be unnecessarily delayed because it happened to be placed on a heavily loaded node. It also ensures that some nodes do not sit idle while other nodes are heavily loaded, thereby keeping overall throughput of the partition as high as possible.

Processes

When a serial program executes on a Sun HPC cluster, it becomes a Solaris process with a Solaris *process ID*, or *pid*. When CRE executes a distributed message-passing program it spawns multiple Solaris processes, each with its own *pid*.

CRE allows you to control several aspects of jobs and process execution, such as:

- Number of processes per job
- Process spawning
- Mapping processes to nodes

For tasks and instructions, see Chapter 4.

Jobs

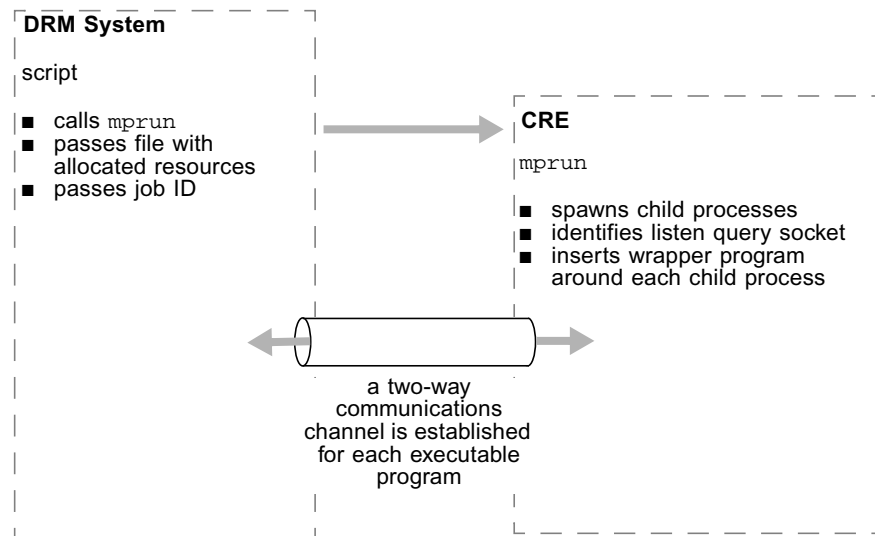
CRE assigns a *job ID*, or *jid*, to a program. In an MPI job, the *jid* applies to the overall job. Many CRE commands take *jids* as arguments. CRE provides a variety of information about jobs. To find out how to obtain that information, see Chapter 7.

How the CRE Environment Is Integrated With Distributed Resource Management Systems

As described in Chapter 1, the ClusterTools 5 environment provides close integration between CRE and three different DRM systems:

- SGE
- LSF
- PBS

The integration process is similar for all three, with some individual differences. The DRM system, whether SGE, LSF, or PBS, launches the job through a script. The script calls `mprun`, and passes it a host file of the resources that have been allocated for the job, plus the job ID assigned by the DRM system.



The CRE environment continues to perform most of its normal parallel-processing actions, but its child processes do not fork any executable programs. Instead, each child process identifies a communications channel (specifically, a listen query socket) through which it can be monitored by the CRE environment while running in the DRM system.

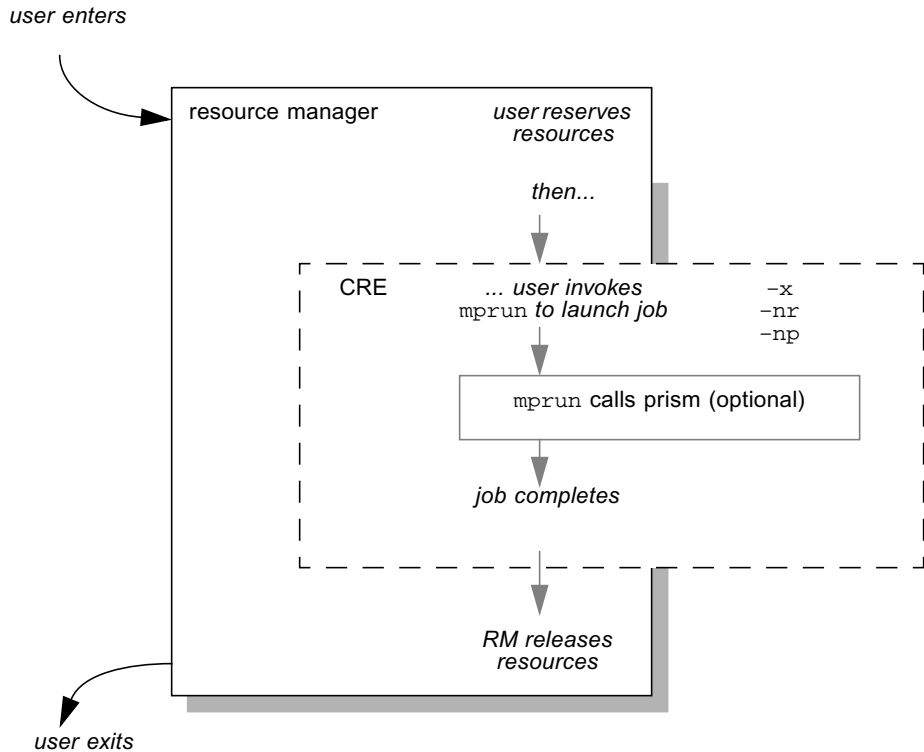
You can also invoke a similar process interactively, without a script. Instructions for script-based and interactive job launching are provided in Chapter 5.

How Programs Are Launched

The exact instructions vary from one resource manager to another, and are affected by CRE's configuration, but they all follow these general guidelines:

1. You can launch the job either interactively or through a script. Instructions for both are provided in Chapter 5 and the following man pages:
 - `lsf_cre(1)`
 - `pbs_cre(1)`
 - `sge_cre(1)`
2. Enter the DRM processing environment before launching jobs with `mprun`.
3. Reserve resources for the parallel job and set other job control parameters from within their resource manager.
4. Invoke the `mprun` command with the applicable resource manager flags. Those flags are described in Chapter 5 and the `mprun(1)` manpage.

Here is a diagram that summarizes the user interaction:



How Distributed Resource Managers Work

If you are using a Distributed Resource Manager (DRM) such as Sun Grid Engine, PBS, or LSF for resource management, all Sun HPC ClusterTools jobs are handled by the DRM's Batch system. Consequently, Sun HPC ClusterTools job submission involves the following:

- When a Sun HPC ClusterTools job is submitted, it is placed in a job queue, running interactively.
- These queues are created by the system administrator. Each queue is defined by a set of job-start criteria, called *job-scheduling policies*. These policies can be specified by the administrator, or default queue policies can be used.

- If a job has particular resource requirements and if a particular queue's job-scheduling policies meet those requirements, you can specify that the job be placed on that queue. If a job does not require special execution conditions, you can leave the choice of queue to the DRM's Batch system.
- The job waits in its queue until it reaches the head of the queue *and* the cluster is able to satisfy the job scheduling policies of that queue. At that point the job is started.

For further information about using DRMs with CRE, see the man pages `sge_cre.1`, `pbs_cre.1`, and `lsf_cre.1`.

Before You Begin

This chapter provides miscellaneous information about the runtime environment that you should know before you begin to use it. It contains the following sections:

- “Prerequisites” on page 15
- “Command and Man Page Paths” on page 15
- “Authentication Methods” on page 16
- “Core Files” on page 16

Prerequisites

If your program uses Sun HPC ClusterTools components, compile and link it on a cluster that contains the Sun HPC ClusterTools software. If you plan to use the Prism environment to debug your program, include the `-g` option when you compile your program.

See the *Sun S3L Software Programming Guide*, the *Sun MPI Software Programming and Reference Guide*, and the *Sun HPC ClusterTools Software Performance Guide* for information on linking in the Sun S3L and the Sun MPI libraries.

Command and Man Page Paths

CRE commands typically reside in the directory `/opt/SUNWhpc/bin`. If you are unable to execute them, you may need to add that directory to your path; check with your system administrator.

The man pages for Sun HPC ClusterTools commands reside in the `/opt/SUNWhpc/man` directory.

Authentication Methods

Sun HPC ClusterTools software supports two optional forms of user authentication that require the execution of user-level commands. The two methods are Kerberos Version 5 and DES. If one of these authentication methods is enforced on your Sun HPC cluster, use the commands listed in the following table.

TABLE 3-1 User Commands Required by Authentication Methods

Authentication Method	Required Command
Kerberos 5	While Kerberos Version 5 authentication is in use, you must issue a <code>kinit</code> command before running any CRE command.
DES	While DES authentication is in use, you must issue the <code>keylogin</code> command before issuing any CRE command.

See your system administrator for guidance.

Core Files

Core files are produced as they normally are in the Solaris environment. However, if more than one process dumps core in a multiprocess program, the resulting core file may be overwritten in the same directory. Use `coreadmin(1M)` to control the naming and placement of core files.

Running Programs With `mprun`

The `mprun` command controls several aspects of program execution. This chapter describes what you can do with the command. It contains the following sections:

- “Syntax” on page 17
- “Controlling Where the Program Runs” on page 20
- “Mapping MPI Processes to Nodes” on page 27
- “Controlling Input/Output” on page 38
- “Controlling Other Job Attributes” on page 48
- “Command Reference (`mprun`)” on page 53

Syntax

```
% mprun [ options ] [ - ] program-name [ program-arguments ]
```

Options

The *options* control the behavior of the command. The tasks they perform are summarized in the diagram on the previous page. TABLE 4-5 lists the options in alphabetical order, with a brief description.

The runtime environment applies the options to the `mprun` command according to useful program logic rather than sequential order. Some options override conflicting options that appear earlier in the command line or in the `MPRUN_FLAGS` environment variable. In some cases, the presence of one option causes other options in the command line to be ignored, even if they appear later in the command line. As a result, option precedence varies by task. A table at the beginning of each group of tasks lists precedence order for the options used in those tasks.

Program-Name

If *program-name* conflicts with the name of an `mprun` *option*, use the - (dash) symbol to separate the program name from the option list. Be sure to add a space between the - symbol and the dash in the program name. For example:

```
% mprun -np 4 - myprogram
```

Program-Arguments

Enter any required *program-arguments* after the *program-name*.

Pre-Entering Command Options with MPRUN-FLAGS

You can pre-enter options to the `mprun` command by setting the `MPRUN-FLAGS` environment variable. Since the `MPRUN-FLAGS` variable only affects default behavior, you can override those options by entering different ones when you enter the `mprun` command itself.

The `MPRUN-FLAGS` environment variable uses the same *options* as the `mprun` command. (For a complete list, see TABLE 4-5.) If you use more than one word, enclose the list in quotation marks.

For example, to make `part2` the default partition, enter:

C shell:

```
% setenv MPRUN_FLAGS "-p part2"
```

Bourne shell:

```
# MPRUN_FLAGS = "-p part2"; export MPRUN_FLAGS
```

You can check the current setting of `MPRUN_FLAGS` by issuing the command `printenv`.

```
% printenv MPRUN_FLAGS
```


Environment Variables Available for Scripts

Three environment variables related to `mprun` are available for your scripts:

<code>MP_RANK</code>	The rank of a process in the job: 0 – 2047
<code>MP_NPROCS</code>	The number of processes in a job: 1 – 2048
<code>MP_JOBID</code>	The <i>jid</i> of the job

Each variable is automatically set by the `mprun` command at execution time. For example, this instance of `mprun...`

```
% mprun -np 6 a.out
```

... would set the value of the variables to:

<code>MP_RANK</code>	0 through 5
<code>MP_NPROCS</code>	6
<code>MP_JOBID</code>	The same <i>jid</i> that can be displayed by <code>mpps</code> (see Chapter 7)

Controlling Where the Program Runs

To Perform This Task	Use This Option
How to run with default settings	
How to run on a different cluster	-c
How to run on a different partition	-p
How to run as multiple processes	-np
How to share nodes	-j
How to enable process spawning	-Ys
How to disable process spawning	-Ns
How to wrap multiple processes	-W
How to settle for available processes	-S
How to include independent nodes	-u
How to Combine Process Placement Options	

Precedence for Program Execution

This Option...	Nullifies the Previous Instance of This Option ...
-np	-np
-Ys	-Ns
-Ns	-Ys
-W	-S
-S	-W
-u	-u
-G	-G
-A	-A
-C	-C

This Option...	Nullifies the Previous Instance of This Option ...
-c	-c
-p	-p
-r	-r

▼ How to Run a Program With Default Settings

To run the program with default settings, enter the command and program name, followed by any required arguments to the program:

```
% mprun program-name
```

▼ How to Run on a Different Cluster (-c)

By default, a program runs on your login cluster. To run a program on a different cluster, use the -c option:

```
% mprun -c cluster-name program-name
```

To find the name of a cluster, use the `mpinfo` command with the -C option, as described in “How to Display Information About the Current Cluster (-C)” on page 95. Note case sensitivity.

▼ How to Run on a Different Partition (-p)

To run the program on a partition other than your login partition, use the -p option:

```
% mprun -p partition-name program-name
```

The partition must be enabled. If it is not enabled, the job fails. (As described in “Partitions” on page 8, if a node is included in multiple partitions, only one partition can be enabled at a time.)

▼ How to Run as Multiple Processes (`-np`)

By default, an MPI program started with `mprun` runs as one process. To run the program as multiple processes, use the `-np` option:

```
% mprun -np process-count program-name
```

When you request multiple processes, CRE attempts to start one process per CPU. If you request more processes than the number of available CPUs, you must use either the `-w` (Page 24) or `-s` (Page 24) options to prevent `mprun` from failing.

If you enter 0 as the number of processes, the runtime environment starts one process per available CPU. For example:

```
% mprun -np 4 a.out
% mprun -p partition2 -np 0 a.out
```

The first example runs four copies of the program `a.out` on the login partition. The second example runs the job on `partition2`, which has six CPUs. Because the second command specifies “0” processes, the runtime environment runs six copies of `a.out`, one for each available CPU.

```
% mprun -np process-count x threads program-name
```

When launching a multi-threaded program, use the `x threads` syntax to specify the number of threads per process. Although the job requires a number of resources equal to `process-count` multiplied by `threads`, only `process-count` processes are started. The ranks are numbered from 0 (zero) to `process-count` minus 1. The processes are allocated across nodes so that each node provides a number of CPUs equal to or greater than `threads`. If threading requirements cannot be met, the job fails and provides diagnostic messages. As with a processor value of 0, a thread value of 0 requests all available resources on the node. In this way it is equivalent to the `-Ns` option.

The syntax `-np process-count` is equivalent to the syntax `-np process-countx1`. The default is `-np 1x1`.

Note – If a batch job calls `MPI_Comm_spawn(3SunMPI)` or `MPI_Comm_spawn_multiple(3SunMPI)`, be sure to use the `-nr` option to reserve the additional resources.

▼ How to Share Nodes (-j)

To run a program on the same node(s) as another program, use the `-j` option:

```
% mprun -j jid [ mprun-options ] program-name
```

The `jid` argument is the program's *job ID* (described in “Jobs” on page 10).

Place additional *mprun-options*, if any, after the `-j` option. Here are two examples.

```
% mprun -j cre.85 a.out
% mprun -j cre.85 -Ns a.out
```

Both of the examples above run the program `a.out` on the same node as the program identified by the `jid` of `85`. The second example includes the `-Ns` option to disable process spawning (Page 23).

▼ How to Enable Process Spawning (-Ys)

To enable a program that runs on a node with multiple CPUs to spawn processes, use the `-Ys` option:

```
% mprun -Ys program-name
```

▼ How to Disable Process Spawning (-Ns)

To limit the number of processes a program uses to one per node, use the `-Ns` option:

```
% mprun -Ns program-name
```

The `-Ns` option prevents nodes that have multiple CPUs from spawning additional processes.

▼ How to Wrap Multiple Processes (-W)

Note – This option is incompatible with the `-z` option.

When you have more processes than available CPUs, use the `-w` option to *wrap* the processes:

```
% mprun -np process-count -w program-name
```

Without the `-w` option, excess processes would make the job fail. The `-w` option assigns as many processes as required to each CPU, and executes the processes one at a time. (To include independent nodes in the wrap, use the `-u` option, described on page 25.)

For example:

```
% mprun -p part2 -np 10 -w a.out
```

If the partition `part2` had six available CPUs and you specified 10 wrapped processes, CRE would distribute the processes among the CPUs according to load-balancing rules.

(The `-s` option, described below, provides a different solution to the same problem.)

▼ How to Settle for Available Processes (-S)

Note – This option is incompatible with the `-z` option.

When you have more processes than available CPUs, use the `-s` option to *settle* for the number of available CPUs.

```
% mprun -np process-count -s program-name
```

Without the `-s` option, excess processes would make the job fail. The `-s` option assigns one process to each CPU, and when it runs out of CPUs, it ignores the remaining processes. (To assign the remaining processes to independent nodes, use the `-u` option, described below.)

For example:

```
% mprun -p part2 -np 10 -s a.out
```

If the partition `part2` had six available CPUs and you specified 10 processes with the `-s` option, CRE would assign one process to each of the six CPUs, and discard the remaining four processes.

(The `-w` option, described on page 24, provides a different solution to the same problem.)

▼ How to Include Independent Nodes (`-u`)

When a partition does not have enough CPUs to handle all the processes of a job, and you select either the `-s` option or the `-w` option, you can use the `-u` option to assign the extra processes to independent nodes outside the partition:

```
% mprun -np process-count -w -u program-name
```

```
% mprun -np process-count -s -u program-name
```

To be eligible, an independent node must satisfy three requirements:

1. It must be enabled.
2. It cannot belong to another partition that is currently enabled.
3. It must be running the same version of the Solaris operating environment as the nodes in the partition. For the current release of Sun HPC ClusterTools software, this OS must be Solaris 8.

For example, assume `partition2` had six available CPUs and the node had two independent nodes. If you specified 10 wrapped processes and added the `-u` option...

```
% mprun -p part2 -np 10 -w -u a.out
```

... CRE would distribute the ten processes among the 8 CPUs, and use load-balancing rules to assign the remaining two processes.

If you specified 10 processes with the `-S` option and added the `-u` option...

```
% mprun -p part2 -np 10 -S -u a.out
```

... CRE would assign one process to each of the six CPUs, one to each independent node, and discard the remaining two processes.

▼ How to Combine Process Placement Options

As described in “How to Run as Multiple Processes (`-np`)” on page 22, you can request x processes, if as many as x processors are available, using the `-np` option. For example,

```
% mprun -np x a.out
```

If you specify 0 as the number of processes, the runtime environment starts one process per available CPU.

However, if you combine the `-np` option with the `-Ns` option (assign one process per node) or the `-W` option (assign processes to the available nodes until the `-np` argument is satisfied),

TABLE 4-1 Combining `mprun` Process Placement Options

Option Combination	Interpretation
<code>mprun -np 0 -Ns a.out</code>	Request a process on each node.
<code>mprun -np x -W a.out</code>	Request x processes, without regard to distribution on the nodes.
<code>mprun -np 0 -Ns -W a.out</code>	Request one process per node, wrap until all processors in your cluster are used.
<code>mprun -np x -Ns -W a.out</code>	Request one process per node until your cluster has x of them.

Mapping MPI Processes to Nodes

To Perform This Task	Use This Option
How to distribute processes among nodes	-l
How to distribute processes by block	-Z or -Zf
How to distribute processes by rank map	-m
How to select nodes by resource requirement	-R

If you assign to a node a number of processes that is greater than the number of CPUs on that node, the runtime environment complies with your request unless the value of `total_max_procs` prevents it.

Precedence for Mapping

Four primary `mprun` options affect rank placement: `-l`, `-m`, `-Z`, and `-Zt`. Four ancillary options also influence rank placement: `-W`, `-S`, `-np`, and `-u`. The following table summarizes an interaction matrix for these options:

This Option	Nullifies Previous Instances of	And Ignores
-l	-l -m -Z -Zt -j -R	
-m	-l -m -Z -Zt -j -R	
-R	-l -m -Z -Zt -j -R	
-j	-l -m -Z -Zt -j -R	-u
-Z	-l -m -Z -Zt -j	-Ns -Ys
-Zt	-l -m -Z -Zt -j	-Ns -Ys

▼ How to Distribute Processes Among Nodes (-l)

To distribute processes among individual nodes, use the `-l` option following the `-np` option:

```
% mprun -np process-count -l rank-spec program-name
```

process-count

The `-np` option (described in “How to Run as Multiple Processes (`-np`)” on page 22) specifies the number of *processes* the program uses.

rank-spec

The *rank-specs* specify how many processes go to each node. Be sure to enclose the set of *rank-specs* with one set of quotation marks, and use commas to separate them from each other:

```
"rank-spec, rank-spec, rank-spec"
```

The number of *rank-specs* you use must be a factor of the number of *processes* you specify with the `-np` option. For example:

```
% mprun -np 1 -l "node0" a.out
% mprun -np 2 -l "node0, node1" a.out
% mprun -np 4 -l "node0, node1, node2, node3" a.out
```

The examples above use one *rank-spec* for one process, two *rank-specs* for two processes, and three *rank-specs* for three processes. You cannot use three *rank-specs* with four processes, for instance, because four processes cannot be evenly distributed across three nodes.

Each *rank-spec* identifies one node and the number of processes that run on it:

```
rank-spec --> node-name [ process-count ]
```

The *node-name* can be a name or an IP address. The *process-count* argument is optional. If you omit it, as in the examples above, one process is assigned to each node. If you have more processes than nodes, you must include the *process-count* argument to indicate how many processes are assigned to each node. For example:

```
% mprun -np 2 -l "node0 2" a.out
```

In the example above, the program runs with two processes on one node, `node0`, so you must indicate that both processes are assigned to `node0`.

In the following example, the program runs with four processes on two nodes, so you must indicate how those processes are assigned to the nodes. Three combinations are possible:

```
% mprun -np 4 -l "node0 2, node1 2" a.out
% mprun -np 4 -l "node0 1, node1 3" a.out
% mprun -np 4 -l "node0 3, node1 1" a.out
```

▼ How to Distribute Processes by Block (-Z and -Zt)

Note – -Z is incompatible with -S or -W.

You can arrange a job's processes into blocks. The blocks of processes are then distributed among the nodes. The -Z option distributes the blocks among the available nodes using load balancing. In other words, two blocks may be assigned to the same node if that is the most efficient way to execute the job. To force each block to be assigned to a separate node instead, use the -Zt option. Use the -Z or -Zt option ahead of the -np option:

```
% mprun -Z block-count -np process-count program-name
% mprun -Zt block-count -np process-count program-name
```

Here are some examples:

```
% mprun -Z 2 -np 4 a.out
% mprun -Zt 2 -np 4 a.out
```

In the example above, the -Z option specifies two blocks. Because the total number of processes is four (-np 4), each block has two processes. They are distributed among available nodes as efficiently as possible. The -Zt option also creates two blocks, each with two processes, but they are distributed to two separate nodes.

Here are more examples:

```
% mprun -Z 3 -np 8 a.out
% mprun -Zt 3 -np 8 a.out
```

Both examples above create three blocks, two with three processes each, and one with two processes.

▼ How to Distribute Processes by Rank Map (-m)

To distribute processes among nodes with a rank map file, use the `-m` option:

```
% mprun -np process-count -m rankmap-file program-name
```

Use the `-m rankmap-file` option to assign processes to nodes as specified in the file *rankmap-file*. The rankmap in the file is specified as one or more nodenames, each followed optionally by the number of processes to assign to that node (in rank order); the default is one. The rankmap file can also accept IP addresses instead of nodenames.

Multiple nodenames (or IP addresses) may be separated by newlines; if multiple nodenames appear on the same line, they are separated by commas.

You can obtain the names and IP addresses of nodes using the `-Nv` option to the `mpinfo` command.

```
% mpinfo -Nv
```

Restrictions

The rank map specified with the `-m` option will be rejected if any of the following conditions are true:

- One or more of the requested nodes is not enabled or is otherwise invalid
- The `max_total_procs` value set via the `mpadmin` command defeats the requested number of ranks for a node
- The requested nodes span multiple enabled partitions
- The requested nodes are running different versions of the operating system
- One or more of the following options is listed either in the command line or in the `MPRUN_FLAGS` environment variable: `-j`, `-Ns`, `-R`, `-Ys`, or `-Z`

process-count

If the *process-count* used with the `-np` option is greater than the number of ranks specified in the rank map, you must use either `-S` (to settle for the available number of ranks in the rank map) or `-W` (to wrap the requested processes on the specified nodes). Otherwise your job will fail.

If the value specified in the `-np` option is less than the number of ranks specified in the rank map, the rank assignment will be limited to the value of `-np`.

If you use `-np 0`, the number of processes will be derived from the number or ranks described in the rank map.

rankmap-file

A rank map file has this syntax:

```
rank-map file-->    node-name [ , ]
node-name [ , ]
node-name [ , ] ...
```

A *node-name* can be a name or an IP address. Since commas can be used to separate node names in a file, you could simply place the contents of an inline rank map in a file. However, new-line characters (`\n`) are also recognized as separators in rank map files, so you will probably find it easier to list each node on its own line. For example:

```
mars 2
venus 2
jupiter 2
```

▼ How to Reserve Resources For Spawning or Multithreading (`-nr`)

This syntax reserves a number of resources equal to *numprocs* x *threads*. These resources are held in reserve over and above the number of resources specified by the `-np` option. Use this option when the batch job contains calls to `MPI_Comm_spawn(3SunMPI)` or `MPI_Comm_spawn_multiple(3SunMPI)`. Specify a number of resources equal to or greater than the total number of processes that will be spawned. For example,

```
% mprun -nr numprocs [ x threads ]...
```

In a multithreaded environment, use the *xthreads* syntax to specify the number of threads per process. The syntax `-nr numprocs` is equivalent to the syntax `-nr numprocsx1`. The default is `-nr 0x1`.

A *threads* setting of 0 allocates the processes among all available reserved resources. It is equivalent to the `-Ns` option.

▼ How to Select Nodes by Resource Requirement (-R)

To distribute processes among nodes by resource requirement, use the `-R` option:

```
% mprun -np process-count -R resource-requirement-spec program-name
```

process-count

The processes are distributed among the nodes that satisfy the criteria in the *resource requirement spec* (RRS).

resource-requirement-spec

The RRS accommodates computing requirements that are more complex than those accepted by rank maps. It has this syntax:

```
RRS --> "resource-requirement [& | | resource-requirement ]..."
```

The `&` symbol is a logical AND operation. In other words, a node must satisfy all the criteria in the spec. The `|` symbol is a logical OR operation. A node must satisfy either of the criteria in the spec. Use them alone or in combination:

```
resource-requirement & resource-requirement  
resource-requirement | resource-requirement
```

Each individual *resource-requirement* has this syntax:

```
resource-requirement --> resource [ operator value ]
```

The *resource* argument identifies the resource whose requirement is specified. For a list of resources, see TABLE 4-2.

The *operator* argument is an arithmetic or logical symbol such as `=` or `>` that indicates the relationship between the *resource* and its *value*. For example:

```
"name=node0"
```

In the example above, the processes are distributed to a node whose name `resource` is equal to `node0`. For a list of *operators*, see TABLE 4-3.

The *value* argument is simply the value of the *resource* that must be met. Although the *operator* and *value* are optional, they are used in the great majority of cases.

The runtime environment parses the attribute settings in the order in which they are listed in the RRS, along with other options you specify. It then merges these results with the results of an internally specified RRS that controls load-balancing.

The result is an ordered list of CPUs that meet your requirements. If a job uses only one process, the process is sent to the first CPU on the list. If a job uses *n* processes, they are distributed among the first *n* CPUs, wrapping if necessary.

Note – Unless `-Ns` is specified, the RRS specifies node resources but generates a list of CPUs. If `-Ns` is specified, the list refers only to nodes.

TABLE 4-2 lists the predefined *resources* you can use. Your system administrator may have defined additional resources for your particular cluster. To display them, use the `mpinfo` command described in Chapter 9.

TABLE 4-2 Predefined Resources

Resource	Description
<code>cpu_idle</code>	Percent of time that the CPU is idle.
<code>cpu_iowait</code>	Percent of time that the CPU spends waiting for I/O.
<code>cpu_kernel</code>	Percent of time that the CPU spends in the kernel.
<code>cpu_type</code>	CPU architecture.
<code>cpu_user</code>	Percent of time that the CPU spends running user's program.
<code>load1</code>	Node's load average for the past minute.
<code>load5</code>	Node's load average for the past 5 minutes.
<code>load15</code>	Node's load average for the past 15 minutes.
<code>manufacturer</code>	Hardware manufacturer.
<code>mem_free</code>	Nodes's available memory, in Mbytes.
<code>mem_total</code>	Node's total physical memory, in Mbytes.
<code>name</code>	Node's host name.
<code>os_max_proc</code>	Maximum number of processes allowed on the node, including cluster daemons.
<code>os_arch_kernel</code>	Node's kernel architecture.
<code>os_name</code>	Operating system's name.

TABLE 4-2 Predefined Resources *(Continued)*

Resource	Description
os_release	Operating system's release number.
os_release_maj	The major number of the operating system's release number.
os_release_min	The minor number of the operating system's release number.
os_version	Operating system's version.
serial_number	Node's serial number.
swap_free	Node's available swap space, in Mbytes.
swap_total	Node's total swap space, in Mbytes.

TABLE 4-3 RRS Operators

Operator	Meaning
<	Select all nodes where the value of the specified attribute is less than the specified value.
<=	Select all nodes where the value of the specified attribute is less than or equal to the specified value.
=	Select all nodes where the value of the specified attribute is equal to the specified value.
>=	Select all nodes where the value of the specified attribute is greater than or equal to the specified value.
>	Select all nodes where the value of the specified attribute is greater than the specified value.
!=	Attribute must not be equal to the specified value. (Precede with a backslash in the C shell.)
!	Boolean FALSE.
<<	Select the node(s) that have the lowest value for this attribute.
>>	Select the node(s) that have the highest value for this attribute.

The operators have the following precedence, from strongest to weakest:

```
unary -
*, /
+, binary -
=, !=, >=, <=, >, <, <<, >>
!
&, |
?
```

Examples

Here are some examples of resource requirement specifiers in use.

```
% mprun -R "name = hpc-demo" a.out
% mpinfo -N -R "partition.name=part1"
% mprun -R "load5 < 4" a.out
```

The last example specifies that you only want nodes whose individual load averages over the previous five minutes were less than four.

When the value of an attribute contains a floating point number or a string decimal number, you must enclose the number in single quotes. For example:

```
% mpinfo -R "os_release='5.8'"
```

Attributes that use either << or >> take no value. For example:

```
% mprun -R "mem_total>>" a.out
```

The example above specifies that you prefer nodes with the largest physical memory available.

If you use the << or >> operator, CRE does not provide load-balancing. In the previous example, CRE would choose the node with the most free swap space, regardless of its load. If you use << or >> more than once, only the last use has any effect — it overrides the previous uses. For example:

```
% mprun -R "mem_free>> swap_free>>" a.out
```

The example above initially selects the nodes that have the most free memory, but then selects nodes that have the largest amount of available swap space. The second selection may yield a different set of nodes than were selected initially.

You can also use arithmetic expressions for numeric attributes anywhere. For example:

```
% mprun -R "load1 / load5 < 2" a.out
```

specifies that the ratio between the one-minute load average and the five-minute load average must be less than two. In other words, the load average on the node must not be growing too fast.

You can use standard arithmetic operators as well as the C conditional operator.

Note – Because some shell programs interpret characters used in RRS arguments, you may need to protect your RRS entries from undesired interpretation by your shell program. For example, if you use `csh`, write `-R \!private` instead of `-R !private`.

Boolean attributes are either true or false. If you want the attribute to be true, simply list the attribute in the RRS. For example, if your system administrator has defined an attribute called `ionode`, you can request a node with that attribute:

```
% mprun -R "ionode" a.out
```

If you want the attribute to be false (that is, you do not want a resource with that attribute), precede the attribute's name with `!`. (Precede this with a backslash in the C shell; the backslash is an escape character to prevent the shell from interpreting the exclamation point as a "history" escape.) For example:

```
% mprun -R "\!ionode" a.out
```

For example:

```
% mprun -R "mem_free > 256" a.out
```

The example above specifies that the node must have over 256 Mbytes of available RAM.

```
% mprun -R "swap_free >>" a.out
```

The example above specifies that the node picked must have the highest available swap space.

The following example specifies that the program must run on a node in the partition with 512 Mbytes of memory:

```
% mprun -p part2 -R "mem_total=512" a.out
```

The following example specifies that you want to run on any of the three nodes listed:

```
% mprun -R "name=node1 | name=node2 | name=node3" a.out
```

The following example chooses nodes with over 300 Mbytes of free swap space. Of these nodes, it then chooses the one with the most total physical memory:

```
% mprun -R "swap_free > 300 & mem_total>>" a.out
```

The following example assumes that your system administrator has defined an attribute called `framebuffer`, which is set (TRUE) on any node that has a frame buffer attached to it. You could then request such a node via this command:

```
% mprun -R "framebuffer" a.out
```

Controlling Input/Output

To Perform This Task	Use This Option
How to redirect output to <code>mprun</code>	<code>-D</code>
How to redirect output to individual files	<code>-B</code>
How shut off all standard I/O	<code>-N</code>
How to redirect with an argument vector	<code>-A</code>
How read standard input from <code>/dev/null</code>	<code>-n</code>
How to redirect with a custom configuration	<code>-I</code>

By default, `mprun` handles standard output and standard error the way `rsh` does: the output and error streams are merged and are displayed on your terminal screen. Note that this behavior is slightly different from the standard Solaris behavior when you are not executing remotely; in that case, the `stdout` and `stderr` streams are separate. You can obtain this behavior with `mprun` via the `-D` option.

Likewise, the `mprun` standard input (`stdin`) is sent to the standard input of all the processes.

You can redirect the `mprun` standard input, output, and error using the standard shell syntax. For example,

```
% mprun -np 4 echo hello > hellos
```

You also can change what happens to the standard input, output, and error of each process in the job. For example,

```
% mprun echo hello > message
```

The example above sends `hello` across the network from the `echo` process to the `mprun` process, which writes it to a file called `message`.

Precedence for Input/Output

Option	Nullifies Previous
-B	-D -N -B -I
-D	-D -N -B -I
-I	-D -N -B -I
-N	-D -N -B -I

The set of `mprun` options that control `stdio` handling cannot be combined. These options override one another. If more than one is given on a command line, the last one overrides all of the rest. The relevant options are: `-D`, `-N`, `-B`, `-n`, `-i`, `-o`, and `-I`.

▼ How to Redirect Output to `mprun` (`-D`)

To redirect a job's `stdout` and `stderr` to those of the `mprun` command, use the `-D` option:

```
% mprun -D program-name
```

▼ How to Redirect Output to Individual Files (`-B`)

You can merge the standard output and standard error streams from each process and direct them to individual files by using the `-B` option.

```
% mprun -B program-name
```

The `-B` option writes one file for each process. The filename has this nomenclature:
`out.jid.rank`

The `jid` is the program's job ID. The `rank` is the rank of the process. The files are stored in the job's working directory.

▼ How to Shut Off All Standard I/O (-N)

To shut off all standard I/O to all processes, use the `-N` option:

```
% mprun -N program-name
```

This option closes all `stdin`, `stdout`, and `stderr` connections for the job. For instance, you can reduce the overhead incurred by establishing standard I/O connections for each remote process and then closing those connections as each process ends.

▼ How to Redirect With an Argument Vector (-A)

By default, `mprun` passes the vector of a program's command-line arguments to the program in the standard way. In cluster-level programming, it is sometimes useful to specify a first argument that is not the name of the program. You can use the `-A` option to do this.

```
% mprun -A program-name argument...
```

The argument to `-A` is the name of the program to be executed. After the program name you can add the argument of your choice. For example, if you issue the command:

```
% mprun a.out arg1 arg2
```

`mprun` passes an array in which the name of the program, `a.out`, is the first element and `arg1` and `arg2` are the second and third elements. Or, to pass `newarg` as the first argument to the program `a.out`, along with `arg1` and `arg2`, you could issue the command:

```
% mprun -A a.out newarg arg1 arg2
```

▼ How to Read Standard Input From `/dev/null` (`-n`)

To read `stdin` from `/dev/null`, use the `-n` option:

```
% mprun -n program-name
```

Reading input from `/dev/null` can be useful when running `mprun` in the background, either directly or through a script. Without `-n`, `mprun` would block in this situation, even if no reads were posted by the remote job. With `-n`, the user process encounters an EOF if it attempts to read from `stdin`. This behavior is similar to the behavior of the `-n` option to `rsh`.

▼ How to Redirect With a Custom Configuration (`-I`)

To redirect output with a custom configuration, use the `-I` option:

```
% mprun -I custom-configuration program-name
```

custom-configuration

A custom configuration tells the runtime environment how to handle each job's I/O streams (standard input, output, and error). It has this syntax:

```
custom-configuration --> file-descriptor [ , file-descriptor ] . . .
```

file-descriptor

Each *file-descriptor* provides handling instructions for one process. It has this syntax:

```
file-descriptor --> stream-number attribute
```

Quotation marks are optional. You can place the *file-descriptors* in any order. A custom configuration can include a *file-descriptor* for each stream associated with a job; if any *file-descriptor* is omitted, its stream is not connected to any device.

If you include strings to redirect both standard output and standard error, you must also redirect standard input. If the job has no standard input, you can redirect file descriptor 0 to `/dev/null`.

stream-number

The *stream* identifies the input, output, or error stream. The standard I/O streams are assigned these numbers:

Stream	Stream Number
standard input (<code>stdin</code>)	0
standard output (<code>stdout</code>)	1
standard error (<code>stderr</code>)	2

attribute

The handling instructions for each *stream* are specified by the *attribute*.

Attribute	Description	Dependencies
<code>r</code>	Read from the stream	
<code>w</code>	Write to the stream	
<code>p</code>	Attach the stream to a pseudo-terminal (<code>pty</code>)	
<code>b</code>	Input only goes to the first process	Must use with <code>r</code>
<code>i</code>	Input only goes to rank 0, not to any other ranks	
<code>l</code>	Make the output line-buffered	Must use with <code>w</code>
<code>t</code>	Tag the line-buffered output with rank number	Must use with <code>w</code>
<code>a</code>	Append the stream to a file	Must use with <code>w</code>
<code>m</code>	Echo keystrokes multiple times for multiple processes	Must use with <code>rp</code>

You must specify either `r` or `w` for each file descriptor—that is, whether the file descriptor is to be written to or read from. Thus, the string

```
5w
```

means that the stream associated with file descriptor 5 is to be written. And

```
0rp
```

means that the standard input is to be read from the pseudo-terminal.

If you use the `p` (`pty`) attribute, you must have one `rp` and one `wp` in the complete series of file descriptor strings. In other words, you must specify both reading from and writing to the `pty`. No other attributes can be associated with `rp` and `wp`.

Note – NFS does not support append operations.

For example, you can make each process send its standard output or standard error to a file on its own node. In the following example, each node will write `hello` to a local file called `message`:

```
% mprun -I "lw=message" echo hello
```

Use the `l` attribute in combination with the `w` attribute to line-buffer the output of multiple processes. This takes care of the situation in which output from one process arrives in the middle of output from another process. For example:

```
% mprun -np 2 echo "Hello"
HelHello
lo
```

With the `l` attribute, you ensure that processes do not intrude on each other's output. The following example shows how using the `l` attribute could prevent the problem illustrated in the previous example:

```
% mprun -np 2 -I "0r, lwl" echo "Hello"
[Return]
Hello
Hello
```

Be sure to press the Return or Enter key to begin the output.

Use the `t` attribute in place of `l` to force line-buffering and, additionally, to prefix each line with the rank of the process producing the output. For example:

```
% mprun -np 2 -I "0r, lwt" echo "Hello"
[Return]
r0:Hello
r1:Hello
```

As with the `-l` option, be sure to press the Return or Enter key to begin the output.

The `b` attribute is input-related and thus can be used only in combination with `r`. In multiprocess jobs, the `b` attribute specifies that input is to go only to the first process, rather than to all processes, which is the default behavior.

The `m` attribute pertains to reading from a pseudo-terminal and thus can be used only with `rp`. The `m` attribute in combination with `rp` causes keystrokes to be echoed multiple times when multiple processes are running. The default is to display multiple keystrokes only once.

Redirecting Output to Other File Descriptors

You can direct one file descriptor's output to the same location as that specified by another file descriptor by using the syntax:

```
fdattr=@other_fd
```

For example, `2w=@1` means that the standard error is to be sent wherever the standard output is going. You cannot do this for a file descriptor string that uses the `p` attribute.

If the behavior of the second file descriptor in this syntax is changed later in the `-I` argument list, the change does not affect the earlier reference to the file descriptor. That is, the `-I` argument list is parsed from left to right.

Redirecting File Descriptor Output to a File

You can tie a file descriptor's output to a file by using the syntax

```
fdattr=filename
```

For example, `10w=output` means that the stream associated with file descriptor 10 is to be written to the file `output`. Once again, however, you cannot use this feature for a file descriptor defined with the `p` attribute.

In the following example, the standard input is read from the `pty`, the standard output is written to the `pty`, and the standard error is sent to the file named `errors`:

```
% mprun -I "0rp,1wp,2w=errors" a.out
```

If you use the `w` attribute without specifying a file, the file descriptor's output is written to the corresponding output stream of the parent process; the parent process is typically a shell, so the output is typically written to the user's terminal.

For multiprocess jobs, each process creates its own file; the file is opened on the node on which the process runs.

Note – If output is redirected such that multiple processes open the same file over NFS, the processes will overwrite each other's output.

In specifying the individual file names for processes, you can use the following symbols:

- `&J` – The job ID of the job
- `&R` – The rank of the process within the job

The symbols will be replaced by the actual values. For example, assuming the job ID is 15, this file descriptor string

```
1w=myfile.&J.&R
```

redirects stdout output from a multiprocess job to a series of files named `myfile.15.0`, `myfile.15.1`, `myfile.15.2`, and so on, one file for each rank of the job.

In the following example, there is no standard input (it comes from `/dev/null`), and the standard output and standard error are written to the files `out.job.rank`:

```
% mprun -I "0r=/dev/null,1w=out.&J.&R,2w=@1" a.out
```

This is the behavior of the `-B` option. Note the inclusion in this example of a file descriptor string for standard input even though the job has none. This is required because both standard output and standard error are redirected.

Maximum Number of File Descriptors

By default, the maximum number of file descriptors that a process can have open is 1024. This is because CRE enforces only the hard limit for file descriptors and ignores any file descriptor soft limit that may be set.

Note – CRE enforces soft limits for all other kernel parameters.

The default, per-process limit of 1024 file descriptors is likely to be more than enough for all but the most extreme MPI job execution requirements. You can, however, easily accommodate exceptional file descriptor demands by taking the following steps:

- Compiling and linking the MPI application to 64-bit libraries
- Running the job in a 64-bit Solaris 8 operating environment
- Increasing the open file descriptor limit to a value that will satisfy expected demands

For example, to increase the file descriptor hard limit to 2048, add the following line to the `/etc/system` file on each node in the cluster:

```
set rlim_fd_max=2048
```

You can also increase the file descriptor hard limit in a 32-bit Solaris 8 environment. However, this approach is not recommended because the 32-bit environment has a kernel-level limit of 1024. Consequently, you would also have to define the C pre-processor symbol `FD_SETSIZE` in your application to be at least as large as the new `rlim_fd_max` value, and then recompile/relink the application.

Using `mprun` Options Instead of Shell Syntax

The default I/O behavior of `mprun` (merged standard error and standard output) is equivalent to:

```
% mprun -I "0rp,1wp,2w=@1" a.out
```

The `-D` option provides separate standard output and standard error streams; it is equivalent to:

```
% mprun -I "0rp,1wp,2w" a.out
```

You can use the `-o` option to force each line of output to be prepended with the rank of the process writing it. This is equivalent to:

```
% mprun -I "0rp,1wt,2w=@1" a.out
```

If you redirect output to a shared file, you must use standard shell redirection rather than the equivalent `-I` formulation (`-I "lwt=outfile"`). The same restriction also applies to the linebuffer formulation (`-I "lwt=outfile"`).

For example, the following command line concatenates the outputs of the individual processes of a job and writes them to `outfile.dat`:

```
% mprun -np 4 myprogram > outfile.dat
```

The following command line concatenates the outputs of the individual processes and appends them to the previous content of the output file:

```
% mprun -np 4 myprogram >> outfile.dat
```

The following table describes three `mprun` command-line options that provide the same control over standard I/O as some `-I` constructs, but are much simpler to express. Their `-I` equivalents are also shown.

TABLE 4-4 `mprun` I/O Shortcut Summary

Command	Description
<code>mprun -i</code>	Standard input to <code>mprun</code> is sent only to rank 0, and not to all other ranks. Equivalent to <code>mprun -I "0rpb,1wp,2w=@1" a.out</code>
<code>mprun -B</code>	Standard output and standard error are written to the file <code>out.job.rank</code> . Equivalent to <code>mprun -I "0r=/dev/null,1w=out.&J.&R,2w=@1" a.out</code>
<code>mprun -o</code>	Use line buffering on standard output, prefixing each line with the rank of the process that wrote it. Equivalent to <code>mprun -I "0rp,1wt,2w=@1" a.out</code>

Note – Specifying `-o` (forcing processes to prepend rank on output lines), or the equivalent `-I` syntax (such as `-I1wt`) will not work if redirection is also specified with `-I` (such as with `-I1w=outfile`). Use the standard shell redirection operator instead.

Use the `-i` option to `mprun` with caution, since the `-i` option provides only one `stdin` connection (to rank 0). If that connection is closed, keyboard signals are no longer forwarded to those remote processes. To signal the job, you must go to another window and issue the `mpkill` command. For example, if you issue the command `mprun -np 2 -i cat` and then type the `Ctrl-d` character (which causes `cat` to close its `stdin` and exit), rank 0 will exit. However, rank 1 is still running, and can no longer be signaled from the keyboard.

These shortcuts are not exact substitutions. CRE uses `ptys` correctly, whether the `-I` option is present or absent. Also, CRE merges standard error with standard output when it is appropriate. If either `stderr` or `stdout` is redirected (but not both), `ptys` are not used and `stderr` and `stdout` are separated. If both `stderr` and `stdout` are redirected, `ptys` are still not used, but `stderr` and `stdout` are combined.

Controlling Other Job Attributes

To Perform This Task	Use This Option
How to include shell-specific actions	
How to move a process to the background	
How to change the working directory	-C
How to use a different user name	-U
How to use a different group name	-A
How to run a job on a different project	-P
How to display command help	-h
How to display the command's version number	-V
How to display job status information	-J
How to store the job name in a file	-d
How to tag output with its rank number	-o

▼ How to Include Shell-Specific Actions

To perform actions that are shell specific, such as executing compound commands, invoke the appropriate shell as part of the `mprun` command:

```
% mprun shell-command shell-options
```

Here are two examples:

```
% mprun csh -c 'echo $USER'  
% mprun csh -c 'cd /foo ; bar'
```

▼ How to Move a Process to the Background

To move either a process started with `mprun` or a script that issues `mprun` commands to the background, redirect `stdin` to a file, like this:

```
% mprun < /dev/null
```

You can also use the `-n` option to `mprun` so that standard input is read from `/dev/null`. See “How to Read Standard Input From `/dev/null` (`-n`)” on page 41.

```
% mprun -n
```

When `mprun` stops, whether via Control-Z or in terminal output, the job under control of `mprun` is stopped.

▼ How to Change the Working Directory (`-C`)

Use the `-C` option to specify the path of an alternative working directory to be used by the processes spawned when you run your program:

```
% mprun -C working-directory program-name
```

Setting a path with `-C` does not affect where the runtime environment looks for executables. If you do not specify `-C`, the default is the current working directory. For example:

```
% mprun -C /home/collins/bin a.out
```

The syntax above changes the working directory for `a.out` to `/home/collins/bin`.

▼ How to Use a Different User Name (`-U`)

To start a program with a different user name or ID, use the `-U` option:

```
% mprun -U username program-name  
% mprun -U userid program-name
```

If you are not the user identified by *username*, you must have superuser privileges.

▼ How to Use a Different Group Name (-G)

To start a program with a different group name or ID, use the -G option:

```
% mprun -G group-name program-name  
% mprun -G groupid program-name
```

You must belong to the group you use, or be the superuser.

▼ How to Run a Job on a Different Project (-P)

For accounting purposes, any job you run is part of your current project. You can set a default project by changing the value of the variable `SUNHPC_PROJECT`. That value overrides your current project. However, you can override both values by adding the -P option to the `mprun` command:

```
% mprun -P project-name
```

▼ How to Specify Verbose Output (-v)

Use this syntax to specify verbose output. For example,

```
% mprun -v
```


▼ How to Display Command Help (-h)

To display a list of `mprun` options, use the `-h` option (alone):

```
% mprun -h
where {options} may include:
-h           Displays this help/usage text
-V           Displays tool version information
-c <cluster> Specifies the cluster to use
-p <partition> Specifies the partition to use
-A <aout>     Specify the argv [0] explicitly
-U <uid>     Specify uid to execute as
-G <gid>     Specify gid to execute as
-I <iofds>   Specify the I/O fd set to multiplex
-Is         Specify CRE I/O (use with -x)
-C <path>   Specify an alternate working directory
-P <project> Specify a project name
-r <path>   Chroot to working dir before execution
-J          Show job id after exec
-np <PxT>   Specify the number of processes/threads in job
-nr <PxT>   Specify the number of processes/threads to reserve
-R <rrs>    Specify Resource Requirement String
-W          Allow wrapping of hosts
-S          Settle for available hosts
-j <job name> Run this job on same resources as <job name>
-i          Only rank 0 gets stdin
-o          Rank-tag stdout
-D          Separate stdout/stderr streams
-N          No stdio connections
-B          Batch stream handling
-n          No stdin connection
-Ns        No spawning on SMP's
-Ys        Enable spawning on SMP's
-Z <n>     Group procs <n> to an SMP
-Zt <n>    Group/tile procs <n> to an SMP
-l "<host> [<procs>][,...]" Specify rankmap string
-m <file>  Specify rankmap file
-u          Use any partition independent nodes
-t <n>     Multiply daemon and mprun timeouts by factor n; n > 1
-d <filename> Dump JID to a file
-v          Verbose. Gives extra information during job startup.
-x <RM>    Run processes under control of resource manager RM
```

▼ How to Display the Command's Version (-V)

To display the command's version number, use the -V (upper case) option (alone):

```
% mprun -V
```

▼ How to Display Job Status Information (-J)

To display information about the job after it finishes executing, add the -J option to the command:

```
% mprun options -J program-name
```

In this example, the job ID (*jid*), cluster name, and number of processes are displayed after the job finishes executing:

```
% mprun -np 4 -J a.out
```

▼ How to Store Job Name in a File (-d)

To store the job name in a user-specified file for later access, use the -d option:

```
% mprun options -d output-file hostname
```

▼ How to Tag Output With Its Rank Number (-o)

To precede each output line with the number of the rank that wrote it, use the -o option:

```
% mprun options -o program-name
```

Command Reference (mprun)

TABLE 4-5 Options for `mprun`

Option	Description
-A	Redirect output with an argument vector
-B	Redirect <code>stderr</code> and <code>stdout</code> output streams to individual files
-C	Run the program using a different working directory
-c	Run the job on a different cluster
-d	Store the job name in a file
-D	Redirect output to <code>mprun</code>
-G	Start the program with a different group name
-h	List the options of the <code>mprun</code> command
-I	Redirect output with a custom configuration
-i	Standard input is sent only to rank 0
-J	Display a program's <i>jid</i> and number of processes after it finishes executing
-j	Run a program on the same nodes as another program
-l	Distribute processes among nodes
-m	Distribute processes among nodes as specified in a rank map file
-n	Read standard input from <code>/dev/null</code>
-N	How to shut off all I/O connections
-np	Run a program on multiple processes
-Ns	Run a program with process spawning disabled
-o	Tag each output line with the rank of the process that wrote it.
-P	Run the program on a different partition
-P	Run the job as part of a different project
-R	Distribute nodes among processes using a resource requirement spec
-S	Settle for the available number of processes
-U	Start the program with a different user name

TABLE 4-5 Options for `mpirun` (Continued)

Option	Description
-u	Include independent nodes when you distribute processes among the nodes of a partition
-V	Display the command's version information
-W	Wrap multiple processes around available nodes
-Ys	Execute the program with process spawning enabled
-Z	Distribute processes among nodes by block
-Zt	Distribute processes among nodes by block, but force each block to use a different node

Running Programs With `mprun` in Distributed Resource Management Systems

This chapter describes the options to the `mprun` command that are used for distributed resource management, and provides instructions for each resource manager. It has four sections:

- “`mprun` Options for DRM Integration” on page 55
- “Running Parallel Jobs in the PBS Environment” on page 57
- “Running Parallel Jobs in the LSF Environment” on page 60
- “Running Parallel Jobs in the SGE Environment” on page 63

`mprun` Options for DRM Integration

Call `mprun` from within the resource manager, as explained in “Integration With Distributed Resource Management Systems” on page 3. Use the `-x` flag to specify the resource manager, and the `-np` and `-nr` flags to specify the resources you need. In addition, the `-Is` flag selects the default CRE I/O environment, the `-v` flag produces verbose output, and the `-J` flag displays a fuller identification of each process.

Some `mprun` flags do not make sense for a batch job, and will cause the `mprun` request to be rejected if used with the `-x` flag. See “Improper Flag Combinations for Batch Jobs” on page 57.

These are the DRM integration options for `mprun`:

-Is

When launching `mprun` from a resource manager, the `-Is` option selects CRE's default I/O behavior, overriding the I/O behavior of the resource manager.

You do not need this option when using any of these `mprun` flags: `-I`, `-D`, `-N`, `-B`, `-n`, `-i`, or `-o`.

Each of those flags already invokes CRE's default I/O behavior. You also do not need this option if you prefer to keep the resource manager's default I/O behavior.

-np *numprocs* [*x threads*]

Request the specified number of processes in a job. The default is 1. Use the argument 0 to specify that you want to start one process for each available CPU, based on your resource requirements.

When launching a multithreaded program, use the *x threads* syntax to specify the number of threads per process. Although the job requires a number of resources equal to *numprocs* multiplied by *threads*, only *numprocs* processes are started. The ranks are numbered from 0 to *numprocs* minus 1. The processes are allocated across nodes so that each node provides a number of CPUs equal to or greater than *threads*. If threading requirements cannot be met, the job fails and provides diagnostic messages.

A *threads* setting of 0 allocates the processes among all available resources. It is equivalent to the `-Ns` option.

The syntax `-np numprocs` is equivalent to the syntax `-np numprocsx1`. The default is `-np 1x1`.

If your batch job calls `MPI_Comm_spawn(3SunMPI)` or `MPI_Comm_spawn_multiple(3SunMPI)`, be sure to use the `-nr` option to reserve the additional resources.

-x *resource_manager*

The `-x` option specifies the *resource_manager*. Supported *resource_managers* are:

- `pbs`
- `lsf`
- `sgc`

If you set a default resource manager in the `hpc.conf(4)` file, `mprun` is automatically launched with that resource manager and you don't have to use the

`-x` option. To override the default, use the `-x resource-manager` flag.

-v

Verbose output. Each interaction of the CRE environment with the resource manager is displayed in the output.

Improper Flag Combinations for Batch Jobs

Do not use the following flags with the `-x resource_manager` flag; if you do, the `mprun` request will be rejected:

- -C
- -G
- -j
- -l
- -m
- -S
- -U
- -W
- -Z

Running Parallel Jobs in the PBS Environment

First reserve the number of resources by invoking the `qsub` command with the `-l` option. The `-l` option specifies the number of nodes and the number of processes per node. For example, this command sequence reserves four nodes with four processes per node for the job `myjob.sh`:

```
% qsub -l nodes=4:ppn=4 myjob.sh
```

Once you enter the PBS environment, you can launch an individual job or a series of jobs with `mprun`. Use the `-x pbs` option to the `mprun` command. The `mprun` command launches the job using the rankmap file produced by PBS and stored in the environment variable `PBS_NODEFILE`. The job ranks are children of PBS, not CRE.

You can run a CRE job within the PBS environment in two different ways:

How to Run an Interactive Job in PBS

How to Run a Script Job in PBS

▼ How to Run an Interactive Job in PBS

1. **Enter the PBS environment interactively with the `-I` option to `qsub`, and use the `-l` option to reserve resources for the job.**

Here is an example.

```
hpc-u2-6% qsub -l nodes=1:ppn=2 -I
```

The command sequence shown above enters the PBS environment and reserves one node with two processes for the job. Here is the output:

```
qsub: waiting for job 20.hpc-u2-6 to start
qsub: job 20.hpc-u2-6 ready
Sun Microsystems Inc. SunOS 5.8 Generic February 2000
pbs%
```

2. **Launch the `mprun` command with the `-x pbs` option.**

Here is an example that launches the `hostname` command with a verbose output:

```
pbs% mprun -x pbs -v hostname
```


The `hostname` program uses the rankmap specified by the PBS environment variable `PBS_NODEFILE`. The output shows the `hostname` program being run on ranks `r0` and `r1`:

```
[mprun:/opt/SUNWhpc/lib/pbsrun -v --  
/opt/SUNWhpc/lib/mpexec -x pbs -v --/usr/bin/hostname]  
  
[pbsrun:r0-r1:/opt/SUNWhpc/lib/mpexec -x pbs -v  
-- /usr/bin/hostname]  
  
[mpexec:r0:/usr/bin/hostname]  
  
[mpexec:r1:/usr/bin/hostname]
```

▼ How to Run a Script Job in PBS

1. Write a script that calls `mpexec` with the `-x pbs` option.

As described on page 56, the `-x` flag identifies the resource manager that will be used for the job launched by `mpexec`. In the following examples, the script is called `myjob.csh`. Here is an example.

```
mpexec -x pbs -v hostname
```

The line above launches the `hostname` program in verbose mode, using PBS as the resource manager.

2. Enter the PBS environment and use the `-l` option to `qsub` to reserve resources for the job.

Here is an example.

```
hpc-u2% qsub -l nodes=1:ppn=2 myjob.csh
```

The command sequence shown above enters the PBS environment and reserves one node with two processes for the job that will be launched by the script named `myjob.csh`.

3. Invoke the script.

Here is the output to the script `myjob.csh`.

```
[mprun:/opt/SUNWhpc/lib/pbsrun -v
--/opt/SUNWhpc/lib/mpexec-x pbs -v --/usr/bin/hostname]

[pbsrun:r0-r1:/opt/SUNWhpc/lib/mpexec -x pbs -v
-- /usr/bin/hostname]

[mpexec:r0:/usr/bin/hostname]

[mpexec:r1:/usr/bin/hostname]
```

As you can see, because the `mprun` command was invoked with the `-x pbs` option, it calls the `pbsrun` command, which calls `mpexec`, which forks into two calls of the `hostname` program, one for each node.

Running Parallel Jobs in the LSF Environment

Because of CRE's integration with LSF in previous versions of ClusterTools, you can launch MPI programs from within LSF in three different ways:

How to Run an Interactive Job in LSF
How to Run a Script Job in LSF
How to Run an LSF Job in Compatibility Mode

Note – When you use the `bkill` command to kill a parent job and its spawned jobs in the LSF environment, the command kills only the spawned job. It does not kill the parent job. The corresponding command in the PBS and SGE environments kills both the parent and spawned jobs. For more information, see the LSF documentation.

▼ How to Run an Interactive Job in LSF

1. Enter the LSF environment with the `bsub` command, and ...

- a. Use the `-Is` option to select interactive mode.
- b. Use the `-n` option to reserve resources for the job.
- c. Use the `-q` option to select the queue

Here is an example.

```
hpc-u2-6% bsub -n 4 -q short -Is csh
```

The command sequence shown above enters the LSF environment in interactive mode, reserves 4 nodes, and selects the `short` queue. Here is the output:

```
Job <24559> is submitted to queue <short>
<<Waiting for dispatch...>>
<<Starting on hpc-u2-6>>
hpc-u2-6
```

2. Enter the `mprun` command with the `-x lsf` option.

Here is an example.

```
hpc-u2-6% mprun -x lsf -v hostname
```

The output shows the `hostname` program being run on ranks `r0` through `r3`:

```
[mpexec:r0:/usr/bin/hostname hpc-u2-6]
[mpexec:r2:/usr/bin/hostname hpc-u2-7]
[mpexec:r1:/usr/bin/hostname hpc-u2-6]
[mpexec:r3:/usr/bin/hostname hpc-u2-7]
```

▼ How to Run a Script Job in LSF

1. Write a script that calls `mprun` with the `-x lsf` option.

As described on page 56, the `-x` flag identifies the resource manager that will be used for the job launched by `mprun`. Here is an example.

```
mprun -x lsf -v hostname
```

The line above launches the `hostname` program in verbose mode, using LSF as the resource manager.

2. Enter the LSF environment with the `bsub` command, and ...

a. Use the `-n` option to reserve resources for the job.

b. Use the `-q` option to select the queue.

c. Invoke the script.

Here is an example.

```
hpc-u2-6% bsub -n 4 -q short myjob.csh
```

The command sequence shown above enters the LSF environment, reserves 4 nodes, selects the `short` queue, and invokes the script `myjob.csh`, which calls `mprun`.

▼ How to Run an LSF Job in Compatibility Mode

The previous release of ClusterTools provided special options for launching jobs in close integration with LSF. They were specified as flags of the `-sunhpc` argument to LSF's `bsub` command. Although they have been deprecated, you can continue to use all the options to the `-sunhpc` argument, except for the `-j` and `-J` flags. Those two flags are no longer valid.

The default queue is usually set to `hpc` (see the *Sun HPC ClusterTools Software Administrator's Guide*), but if it is not, you can use the `-q` flag to select either the `hpc` or `hpc-batch` queue. For example:

```
% bsub -n 16 -q hpc -sunhpc flags a.out arguments
```

The example shown above launches a job with the `bsub` command, uses the `-n` flag to reserve the resources, the `-q` flag to select the `hpc` queue, and the `-sunhpc` flag to specify the job particulars normally specified by the `mprun` command.

Running Parallel Jobs in the SGE Environment

You can launch MPI programs from within SGE in two different ways:

How to Run an Interactive Job in SGE

How to Run a Script Job in SGE

▼ How to Run an Interactive Job in SGE

1. Enter the SGE environment with the `qsh` command, and ...

a. Use the `-pe` option to reserve resources for the job.

b. Use the `-cre` option to specify CRE as the parallel processing environment.

Here is an example.

```
hpc-u2-6% qsh -pe cre 2
```

The command sequence shown above enters the SGE environment in interactive mode, reserves 2 nodes, and specifies CRE as the parallel processing environment. Here is the output:

```
waiting for interactive job to be scheduled ...  
  
Your interactive job 24 has been successfully scheduled.
```

2. Enter the `mprun` command with the `-x sge` option.

Here is an example.

```
hpc-u2-6% mprun -x sge -v hostname
```

The output shows the `hostname` program being run on ranks `r0` and `r1`:

```
[r0: aout: qrsh, args:
  qrsh -inherit -V hpc-u2-7 /opt/SUNWhpc/lib/mpexec
  -x sge -- hostname]

[r1: aout: qrsh, args:
  qrsh -inherit -V hpc-u2-6 /opt/SUNWhpc/lib/mpexec
  -x sge -- hostname]
```

▼ How to Run a Script Job in SGE

1. Write a script that calls `mprun` with the `-x sge` option.

As described on page 56, the `-x` flag identifies the resource manager that will be used for the job launched by `mprun`. Here is an example.

```
set echo
mprun -x sge -v hostname
```

The line above launches the `hostname` program in verbose mode, using SGE as the resource manager.

2. Enter the SGE environment with the `qsub` command, and ...

- a. Use the `-pe` option to reserve resources for the job.**
- b. Use the `-cre` option to specify CRE as the parallel processing environment.**

c. In the `qsub` syntax, use the script name instead of the program name.

Here is an example.

```
hpc-u2-6% qsub -pe cre 2 myjob.csh
```

The command sequence shown above enters the CRE environment, reserves 2 nodes, and invokes the script `myjob.csh`, which calls `mpirun`. Here is the output:

```
your job 33 ("myjob.csh") has been submitted
```

That's all you have to do to run the job.

3. To display the output from the job, find the output file and display its contents.

a. First use the `ls` command to list the files into which the script has loaded the output.

This example uses the job number to identify the output files:

```
hpc-u2-6% ls *33
myjob.csh.e33 myjob.csh.o33 myjob.csh.pe33 myjob.csh.po33
```

The file that contains the job's errors is named `myjob.csh.e33`. The file that contains the job's output has the name `myjob.csh.o33`.

b. To view the job's output, display the contents of the job's output file.

Continuing with the example above:

```
hpc-u2-6% cat myjob.csh.o33

[r0: aout: qysh, args:
qysh -inherit -V hpc-u2-6
/opt/SUNWhpc/lib/mpexec -x sge --hostname]

[r1: aout: qysh, args:
qysh -inherit -V hpc-u2-7
/opt/SUNWhpc/lib/mpexec -x sge --hostname]
```


Killing or Sending Signals to Programs With `mpkill`

What You Can Do

To Perform This Task	Use This Option
How to kill a running program	<code>mpkill</code>
How to remove all traces of a job	<code>-C</code>
How to display a list of signals	<code>-l -d</code>
How to send a signal to a job	<code>-signal</code>

Return Values

The `mpkill` command returns these values:

- 0 - The command executed successfully.
- 1 - An error occurred during execution. For example, the job was not known.
- 2 - The command was partially successful. This typically occurs when you send a signal to a job in which one or more of the processes has already exited and therefore could not receive the signal. Note that this is usually not an error, since the reason you are using `mpkill` is most likely to eliminate a job that has hung in this intermediate state.

▼ How to Kill a Running Program

To kill a running program, use the `mpkill` command and the program's job ID:

```
% mpkill jid
```

The `mpkill` command stops all the processes associated with the Job ID.

The job ID now begins with the name of the resource manager (`cre`, `lsf`, `pbs`, or `sge`). For example: `lsf.1289`. To obtain a program's job ID, use the `mpps` command, described in “How to Display Information About Individual Jobs (-J)” on page 73.

▼ How to Remove All Traces of a Job

If you have killed a job but it continues to appear in the output of the `mpps` command (described in Chapter 7), log in as root to the master node and invoke the `mpkill` command with the `-C` option and the *jid*.

```
% mpkill -C jid
```

The `-C` option purges the job from the CRE database, including unpublishing names associated with the job.

Note – Processes spawned in the ClusterTools Runtime Environment are not killed by the `mpkill` or `kill` commands so long as they have (spawned) child processes running. To remove the parent process, you must first remove all of its child processes.

▼ How to Display a List of Supported Signals (-l -d)

To simply list the supported signals, use the `-l` option.

```
% mpkill -l
```

To display a list with brief descriptions, use the `-d` option.

```
% mpkill -d
```

▼ How to Send a Signal to a Job

To send a signal to a job, use this syntax:

```
% mpkill -signal jid
```

For example:

```
% mpkill -CONT sge.59
```

The example above sends a `SIGCONT` signal to the processes of the program whose job ID is `sge.59`.

Issuing `mpkill` without specifying a signal sends a `SIGTERM` to the job. Command Reference (`mpkill`)

TABLE 6-1 Options for `mpkill`

Command	Description
<i>none</i>	Stop all processes associated with a particular job
<code>-C</code>	Remove all traces of a job, including unpublished names, from the CRE database
<code>-l</code>	Display a list of supported signals
<code>-d</code>	Display a descriptive list of supported signals
<code>-signal</code>	Send a signal to a job

Displaying Program Information With `mpps`

What You Can Do

To Perform This Task	Use this Option
How to display job status	<i>none</i>
How to display information about individual jobs	-J
How to display name, PID, and host of current job	-b
How to display information about all jobs	-e
How to display a job's start time	-f
How to display job information by partition	-A -a
How to display job information by process	-P -p

▼ How to Display Job Status

To display status information about your jobs running in the default partition, enter the `mpps` command without options:

```
% mpps
```

For example:

```
% mpps
JOBNAME  NPROC  UID   STATE  AOUT
cre.41   3      slu  RUN    AAA
cre.46   4      slu  EXNG   tmp
cre.49   1      slu  EXIT   tmp
cre.99   9      slu  EXNG   uname
cre.100  9      slu  EXNG   uname
```

The status fields are described in TABLE 7-1.

TABLE 7-1 Job Status Displayed by `mpps`

<code>mpps</code> Output	Description
CORE	The job or process exited due to a signal and core was dumped.
CORING	The job is exiting due to a signal. The first process to die dumped core.
EXIT	The job or process exited normally.
EXITING	The job is exiting. At least one process exited normally.
FAIL	The job or process failed while starting, or was aborted.
FAILING	Initialization of the job failed, or a job abort has been signaled.
RUN	The job or process is running.
SIGNALING	The job or process exited due to a signal.
SIGNALED	The job is exiting due to a signal. The first process to die was killed by a signal. At least one of its processes is still in the RUN state.
SPAWN	The job or process is being spawned.
STOPPED	The job or process is stopped.

▼ How to Display Information About Individual Jobs (-J)

To display information about a job, use the `-J` option and a *job-attribute*.

```
% mpps -J job-attribute[ ,job-attribute...]
```

Separate multiple *job-attributes* either with a comma or a space, but not both.

TABLE 7-2 Job attributes for `-J` option to `mpps`

Attribute	Description
<code>part</code>	The name of the partition running the job
<code>jobname</code>	The job's unique ID, expressed as <code><resource-manager>.jobname</code>
<code>mprun_pid</code>	The process identifier (PID) of the current <code>mprun</code> job
<code>mprun_host</code>	The host of the current <code>mprun</code> job
<code>nproc</code>	The number of processes requested (the actual number of processes started may differ if the <code>-W</code> ("How to Wrap Multiple Processes (-W)" on page 24) or <code>-S</code> ("How to Settle for Available Processes (-S)" on page 24) flags were used with <code>mprun</code>)
<code>uid</code>	The user on whose behalf the job was run (normally the user who submitted the job)
<code>gid</code>	The group on whose behalf the job was run (normally the group of the user who submitted the job)
<code>state</code>	<code>BUILD</code> - The job is being submitted <code>WAIT</code> - The job is waiting to run <code>SPAWN</code> - The job is preparing to run <code>RUN</code> - The job is running <code>RSTRT</code> - The job has been killed because one of the nodes on which it was running went down; the job will be restarted
<code>running</code>	The number of processes actually running in this job. Not always equal to the number of processes started for the job because processes that have exited are not counted
<code>wkdir</code>	The directory in which the job's processes start
<code>aout</code>	The name of the program
<code>paout</code>	The full path of the program
<code>ctime</code>	The time when <code>mprun</code> was invoked

TABLE 7-2 Job attributes for `-J` option to `mpps`

Attribute	Description
<code>args</code>	The command-line arguments of the program
<code>stime</code>	The time the job was started
<code>prio</code>	The job priority (higher numbers run first)

▼ How to Display Job Name, PID, and Host of Current Job (`-b`)

Use the `-b` option to display job name, process identifier, and host of a current MPI job.

```
% mpps -b
```

▼ How to Display Information About All Jobs (`-e`)

Use the `-e` option to display information about all jobs.

```
% mpps -e
```

▼ How to Display a Job's Start Time (`-f`)

Use the `-f` option to display the start time for each job.

```
% mpps -f
```

▼ How to Display Job Information by Partition (`-A -a`)

To display information about jobs running in all partitions, use the `-A` option.

```
% mpps -A
```


To display information about jobs running in a specific partition, use the `-a` option, followed by the name of the partition.

```
% mpps -a partition-name
```

▼ How to Display Job Information by Process (`-p` `-P`)

Use the `-p` option to include information about the processes that make up the jobs:

```
% mpps -p
```

For example:

```
% mpps -p
JID      NPROC  UID    STATE  AOUT   RANK   PID    STATE  NODE
lsf.2320 4      shaw   RUN    sleep  0      10190  RUN    node6
                                     1      4744  RUN    node7
                                     2      16564 RUN    node4
                                     3      9412  RUN    node5
```

The output fields are described in TABLE 7-3, below.

To display information about a particular process attribute, use the `-P` option:

```
% mpps -P process-attribute[ , process-attribute...]
```

Separate multiple *process-attributes* either with a comma or a space, but not both. Use the attributes described in TABLE 7-3, below.

TABLE 7-3 Process attributes for `-P` option to `mpps`

Attribute	Description
rank	The rank of the process within the job
pid	The process ID
state	The current execution state of the process

TABLE 7-3 Process attributes for `-P` option to `mpps`

Attribute	Description
<code>ioid</code>	The process ID of the I/O daemon for this process
<code>load</code>	The load on the node executing the process
<code>node</code>	The name of the node executing the process

Command Reference (`mpps`)

TABLE 7-4 Options for `mpps`

Option	Description
<code>none</code>	Display status information about your jobs running in the default partition
<code>-J</code>	Display information about a particular job
<code>-b</code>	Display name, process identifier, and host of current job
<code>-e</code>	Display information about all jobs
<code>-f</code>	Display the time a job started
<code>-A</code>	Display information about jobs running in all partitions
<code>-a</code>	Display information about jobs running in a particular partition
<code>-P</code>	Display process information about a job
<code>-P</code>	Display information about a particular process attribute

Profiling Programs With MPPROF

This chapter explains how to use the MPPProf utilities and environment variables to extract profiling information from executing MPI programs and generate formatted reports from that data. It also explains how to convert intermediate binary files to unanalyzed ASCII files. It has four sections:

- “Enabling MPI Profiling” on page 77
- “Controlling Data Collection” on page 78
- “Using mpprof to Generate Reports” on page 80
- “Using `mpdump` to Convert Intermediate Binary Files to ASCII Files” on page 88

Enabling MPI Profiling

To enable MPI profiling with MPPProf, set the `MPI_PROFILE` environment variable to 1 before starting the MPI program.

```
% setenv MPI_PROFILE 1
```

When MPPProf is enabled, it will extract information about calls to Sun MPI routines and store the information in a set of intermediate files. A separate data file is created for each MPI process rank in which MPI calls were made.

MPPProf also creates an index file describing the locations of the intermediate files. The index file name has the form:

```
mpprof.index.rm.jid
```

where *rm* is the name of the resource manager and *jid* is the job ID. The report generator, `mpprof`, uses this index file to gather the profiling data and associate it with particular MPI processes.

Note – If the MPI program uses an instrumented loadable protocol module (PM), MPPProf passes PM-related profiling data back to the PM. This allows the reporting of PM profile data independent of `mpprof`.

Controlling Data Collection

MPPProf provides the following environment variables to control various aspects of MPPProf behavior:

- `MPI_PROFDATADIR` – Specify a location where the intermediate files will be stored.
- `MPI_PROFINDEXDIR` – Specify a location where the index file will be stored.
- `MPI_PROFINTERVAL` – Specify an interval between intermediate file updates.
- `MPI_PROFMAXFILESIZE` – Specify the maximum size, in kilobytes, that can be written to any intermediate file.

These environment variables are explained in the following sections.

`MPI_PROFDATADIR`

The environment variable `MPI_PROFDATADIR` can be used to specify a nondefault location where the intermediate files will be created for each process. This directory must exist.

If profiling is enabled and `MPI_PROFDATADIR` specifies a directory that does not exist, MPPProf will output an error message and abort the program. If `MPI_PROFDATADIR` is not set and profiling is enabled, intermediate files will be created in `/usr/tmp`.

`MPI_PROFINDEXDIR`

The environment variable `MPI_PROFINDEXDIR` can be used to specify a nondefault location for storing the index file. This directory must exist.

If profiling is enabled and `MPI_PROFDATADIR` specifies a directory that does not exist, MPPProf will output an error message and abort the program. If profiling is enabled and `MPI_PROFINDEXDIR` is not set, the index file will be stored locally in the current directory.

MPI_PROFINTERVAL

The environment variable `MPI_PROFINTERVAL` can be used to specify a data sampling period. When this value is a number greater than 0, the intermediate files will be updated at the prescribed intervals. The data recorded with each update represent the MPI activity that occurred since the previous update.

Setting `MPI_PROFINTERVAL` to 0 forces updates to be made for every MPI call. Setting `MPI_PROFINTERVAL` to `Inf` (meaning infinity) causes the intermediate files to be updated only once, at `MPI_Finalize` time. If `MPI_PROFINTERVAL` is unset or is set without a value, the default value of 60 seconds will be used.

The following example sets an interval of 5 seconds:

```
% setenv MPI_PROFINTERVAL 5
```

In this case, the first update of intermediate files would occur approximately 5 seconds after the `MPI_Init` call, with additional updates appended to the file at 5-second intervals.

If time intervals are used and an MPI program terminates before the `MPI_Finalize` call, any updates that were made can be used by `mpprof` to generate a profile of program operations up to the point of termination.

MPI_PROFMAXFILESIZE

The environment variable `MPI_PROFMAXFILESIZE` can be used to specify, in kilobytes, the maximum amount of data a process can record in its intermediate file. The default value is 51,200 kilobytes (approximately 50 megabytes). Setting `MPI_PROFMAXFILESIZE` to unlimited removes any limits on the size to which the intermediate files can grow. Setting `MPI_PROFMAXFILESIZE` to 0 is an error and will cause MPPProf to abort the program.

If a write to a given intermediate file exceeds the `MPI_PROFMAXFILESIZE` limit, the write operation will complete, but that process will be unable to record additional profiling data. Profiling can continue for other processes that have not reached the file size limit.

Using mpprof to Generate Reports

This section shows how to use the `mpprof` command, which is used to generate reports from profiling information that is collected by the MPI library. To enable MPI profiling, you set the `MPI_PROFILE` environment variable to 1 before executing the `mprun` command.

When you use the `mpprof` command, you always specify an index file that points to files that contain profiling data. The naming convention for the index file is `mpprof.index.rm.jid`, where *rm* is the resource manager used and *jid* is the job ID assigned to the MPI program. When profiling is enabled, the MPI library creates the index file in the current directory by default.

Note – The `mpdump` command, discussed in the next section, is used to translate binary profiling data into ASCII text.

mpprof Command Syntax

The `mpprof` command syntax is

```
mpprof [-h] [-V] [-r] [-S] [-g output-directory] [-p process-set]
[-c columns] -s start-time [-e end-time] [-o output-file]
index-file
```

where the following command options are available:

TABLE 8-1 mpprof Command Options

Option	Description
-h	Lists the optional arguments and exits.
-V	Displays the program version during startup.
-r	Removes the intermediate files associated with the specified <i>index-file</i> . By default, the remove command will use <code>rsh</code> for each node in order to unlink the files. If the <code>-r</code> option is used with the <code>-S</code> option, the secure shell (<code>ssh</code>) will be used instead of <code>rsh</code> . If the <code>-r</code> option is used with the <code>-p</code> option, only a subset of the files will be removed.
-S	Use the secure <code>ssh</code> and <code>scp</code> commands instead of <code>rsh</code> and <code>rcp</code> .

TABLE 8-1 mpprof Command Options

Option	Description
-g	<p>Gathers (copies) the intermediate files associated with the specified index file to the specified <i>output-directory</i>. The file names will be preserved and will overwrite any existing files of the same name. A new index file will be created in the specified <i>output-directory</i> to be used with the files. If the -p option is used with the -g option, a subset of the files will be copied.</p> <p>.The -g option can be used with the -r option as a way to archive the intermediate data files. If the -g <i>output-directory</i> contains previously stored intermediate files, a new index file will be created. No data will be lost.</p>
-p	<p>Specifies a subset of the processes to be analyzed. The -p <i>process-set</i> option is entered as a comma-separated list of tuples.</p> <p>If the -p option is used with -s and/or -g, the operation will be performed only on the specified subset. If a process set encompasses the complete set of ranks, the -p option will have no effect.</p> <p>If the -p option is given a stride, the upper bound of the stride is omitted. For example, if a stride is specified as 1:4:2, the upper bound (4) is not included in the set. In this example, the resulting set is {1,3}.</p> <p>See the section “Reporting on Specific Processes” below for additional information.</p>
-c	<p>Specifies the output width of the generated report. The smallest number of <i>columns</i> supported is 35. When mpprof prints to a terminal window, the display width will be adjusted automatically unless this option is specified. Wrapping will occur if the terminal window has fewer than 35 columns.</p>
-s	<p>Specifies that any profiling data generated before the specified <i>start-time</i> should be ignored. This time reference is measured from the point at which the program calls <code>BMPI_Init</code>. The -s option is useful only when a profiling interval has been specified for the intermediate files. If no <i>start-time</i> is specified on the mpprof command line, the earliest time specified in any intermediate file is used to begin data recording.</p>
-e	<p>Specifies that any profiling data occurring after the specified <i>end-time</i> should be ignored. This time is also measured in seconds, beginning from the time the program calls <code>MPI_Init</code>. This option can be used with, or in place of, the -s option. If no <i>end-time</i> is specified, the latest time in any intermediate file is used for reporting.</p>
-o	<p>Specifies that the report should be written to the specified <i>output-file</i>. If this option is not used or if the <i>output-file</i> is '-', the report will be written to stdout. If the <i>output-file</i> already exists, it will be overwritten.</p>

Generating a Message Passing Report

The `mpprof` command generates a report file based on processes listed in a specified index file. The following example generates a report file called `report.txt` that is based on process profile data stored in files that are specified in an index file called `mpprof.index.cre.14`:

```
% mpprof -o report.txt mpprof.index.cre.14
```

When you enter this command, a new file called `report.txt` is created in the current directory. If you did not use the `-o report.txt` command line option, the report would be sent to the standard output.

Reporting on Specific Processes

You can fine-tune profiling output by specifying a subset of processes of the job, using the `-p` option as shown in the following example:

```
% mpprof -p 0,3-5,9:12,18:27:3 -o report.txt mpprof.index.cre.14
```

The output report file is always specified before the index file. The preceding command causes these processes to be analyzed for the report: 0, 3, 4, 5, 9, 10, 11, 12, 18, 21, 24, and 27. To view the report, open the `report.txt` file in the current directory.

Reporting Processes That Occur After a Specified Time Interval

The following example runs the `mpprof` command to generate a report on all processes that occur starting 2 minutes (120 seconds) or later after a call to `MPI_Init`, which initializes the accumulation of data on processes:

```
% mpprof -s 120 -o report.txt mpprof.index.cre.14
```

The `-s` option is useful when you want to exclude startup routines from reporting because startup routines may not be critical to performance. To view the report, open the `report.txt` file in the current directory.

To Save Report Output for Later Use

If you want to save report output to a specific directory other than the current directory, use the `-g` option:

```
% mpprof -g report.txt mpprof.index.cre.14
```

You can combine the `-g reportname` option with the `-r` option, removing the intermediate data files created during the profiling process at the same time as you store the report output:

```
% mpprof -r -g report.txt mpprof.index.cre.14
```

A Sample Report

The sample report shown is the result of running the following command, which places the report in a file called `report.txt`:

```
% mpprof -p 0,3-5,9:12,18:27:3 -o report.txt mpprof.index.cre.39
```

Note – The file `mpprof.index.cre.39` was created when this MPI command was run:

```
% mprun -np 28 connectivity
```

The command generates 28 separate files.

The sample report is shown below:

```
OVERVIEW
```

```
=====
```

```
The program being reported on is "/home/mscfie/tests/first-  
string/connectivity," which ran as job name "cre.39" on Tue Oct 29 14:04:29 2002.
```

```
Profiled Time Range:
```

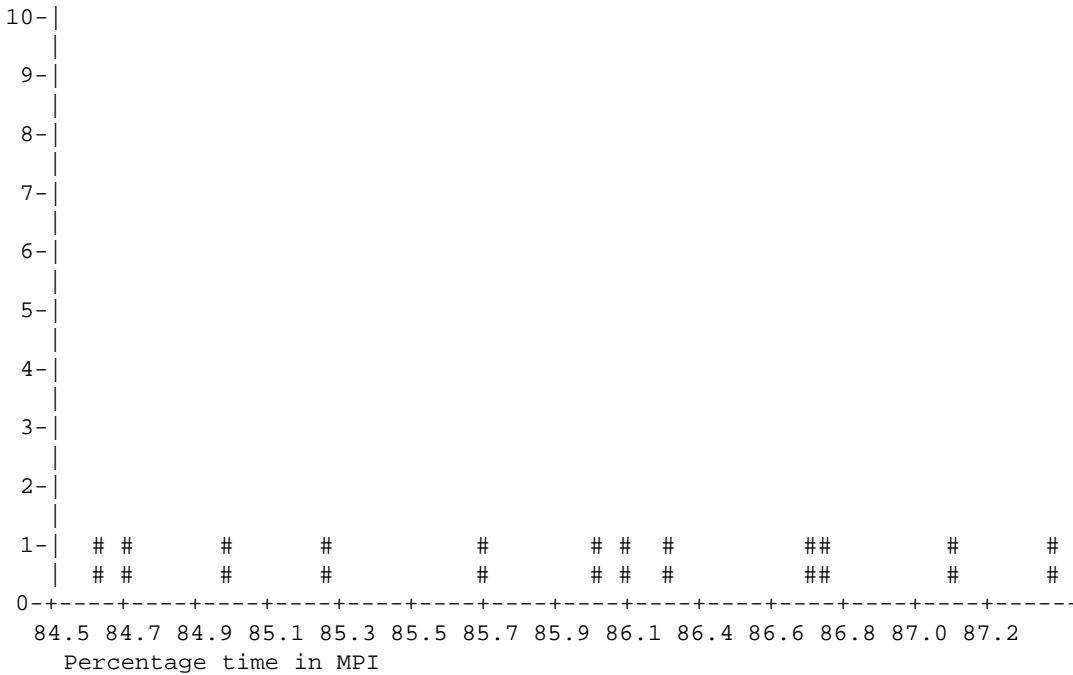
```
Start at elapsed time 0.000030 secs  
End at elapsed time 0.008181 secs  
Total duration is 0.008151 secs  
Fraction spent in MPI 85.9%
```

Elapsed time is measured from the end of MPI_Init. Data is being reported for 12 processes of a 28-process job.

LOAD BALANCE
 =====

Data is being reported on 12 MPI processes. The following histogram shows how these processes were distributed as a function of the fraction of the time the processes spent in MPI routines:

Number of MPI Processes



Rank	Hostname	MPI Time
12	hpc-smp6-0	84.51%
24	hpc-smp6-0	84.62%
27	hpc-smp6-0	84.91%
0	hpc-smp6-0	85.18%
5	hpc-smp6-0	85.62%
10	hpc-smp6-0	85.96%
4	hpc-smp6-0	86.03%
3	hpc-smp6-0	86.18%
21	hpc-smp6-0	86.59%
11	hpc-smp6-0	86.63%
18	hpc-smp6-0	87.00%
9	hpc-smp6-0	87.30%

Low MPI time for an MPI process may indicate the process has too much of a compute load. A high compute load forces the other processes to wait, increasing their MPI time.

To focus reporting on one or more particular processes you may use the `-p` option on the command line for `mpprof`. Type `"mpprof -h"` for more information.

MPI ENVIRONMENT VARIABLES =====

MPI_POLLALL:

You ran with full polling of connections. This means that Sun MPI monitored all connections for incoming messages, whether your program explicitly posted receive requests for those connections or not. Typically, this leads to a degradation in performance.

Suggestion: Set the environment variable `MPI_POLLALL` to `"0"`.

Warning: If your program relies on `MPI_Send` to provide substantial internal buffering of messages, this suggestion could result in deadlock. On the other hand, that would be an indication that the program is not MPI compliant. If such deadlock results, it may be resolved by disregarding this suggestion. Even better performance could result, however, by modifying the program to post the appropriate receives or, in some cases, by setting MPI environment variables to increase internal buffers.

SHM ===

The loadable protocol module `"shm"` has nothing to report.

SUGGESTION SUMMARY =====

Summary of environment variable suggestions:

Set: `MPI_POLLALL=0`

In the C shell, these environment variables may be set by the following commands:

```
setenv MPI_POLLALL 0
```

In the Bourne or Korn shell, these environment variables may be set by the following commands:

```
export MPI_POLLALL=0
```

BREAKDOWN BY MPI ROUTINE

=====

Here, averages over all MPI processes profiled are reported. The numbers in parentheses roughly indicate the variations there are among all of the MPI processes. These variations are computed as $(1-\min/\max)/2$ where "min" and "max" are the minimum and maximum values, respectively, for each statistic reported. A total of 5 different MPI APIs were called.

MPI Routine	Time	Calls Made	Sent	Received
MPI_Barrier	0.001625 (49.8%)	1 (0.0%)	0 (0.0%)	0 (0.0%)
MPI_Comm_rank	0.000003 (13.5%)	1 (0.0%)	0 (0.0%)	0 (0.0%)
MPI_Comm_size	0.000002 (10.1%)	1 (0.0%)	0 (0.0%)	0 (0.0%)
MPI_Recv	0.003758 (39.3%)	27 (0.0%)	0 (0.0%)	108 (0.0%)
MPI_Send	0.001612 (43.2%)	27 (0.0%)	108 (0.0%)	0 (0.0%)

Where "Time" is in seconds and "Sent" and "Received" are in bytes.

TIME DEPENDENCE

=====

Here is a rough depiction of time variations in MPI usage over the reported time range. The fraction of time spent in each of the top 4 MPI routines overall are shown for 60.0-second time periods.

Each time period is specified with an integer. This integer roughly corresponds to the number of time periods passed, with the first time period lasting from 0-60.0 seconds. There are 1 different time periods. Time periods may be missing if no MPI calls were made during the period. Times for MPI calls that persist over multiple reporting intervals will only be reported in a single interval; these reported times may be greater than 100%.

period	MPI_Recv	MPI_Barrier	MPI_Send	MPI_Comm_rank
1	0.0%	0.0%	0.0%	0.0%

CONNECTIONS

=====

A connection is a sender/receiver pair. For 28 processes, there are $28 \times 28 = 784$ connections, including send-to-self connections.

You asked to see data on a 12-member subset of the MPI processes. Only point-to-point messages that were sent from this subset are reported. The following statistics represent $12 \times 12 = 144$ connections.

Here are statistics on the messages sent for each connection, reported on a scale of 0-99 with 99 corresponding to 1 messages:

sender	0	3	4	5	9	10	11	12	18	21	24	27
--------	---	---	---	---	---	----	----	----	----	----	----	----

```
receiver
```

```
0 _ 99 99 99 99 99 99 99 99 99 99 99
1 99 99 99 99 99 99 99 99 99 99 99 99
2 99 99 99 99 99 99 99 99 99 99 99 99
3 99 _ 99 99 99 99 99 99 99 99 99 99
4 99 99 _ 99 99 99 99 99 99 99 99 99
5 99 99 99 _ 99 99 99 99 99 99 99 99
6 99 99 99 99 99 99 99 99 99 99 99 99
7 99 99 99 99 99 99 99 99 99 99 99 99
8 99 99 99 99 99 99 99 99 99 99 99 99
9 99 99 99 99 _ 99 99 99 99 99 99 99
10 99 99 99 99 99 _ 99 99 99 99 99 99
11 99 99 99 99 99 99 _ 99 99 99 99 99
12 99 99 99 99 99 99 99 _ 99 99 99 99
13 99 99 99 99 99 99 99 99 99 99 99 99
14 99 99 99 99 99 99 99 99 99 99 99 99
15 99 99 99 99 99 99 99 99 99 99 99 99
16 99 99 99 99 99 99 99 99 99 99 99 99
17 99 99 99 99 99 99 99 99 99 99 99 99
18 99 99 99 99 99 99 99 99 _ 99 99 99
19 99 99 99 99 99 99 99 99 99 99 99 99
20 99 99 99 99 99 99 99 99 99 99 99 99
21 99 99 99 99 99 99 99 99 99 _ 99 99
22 99 99 99 99 99 99 99 99 99 99 99 99
23 99 99 99 99 99 99 99 99 99 99 99 99
24 99 99 99 99 99 99 99 99 99 99 _ 99
25 99 99 99 99 99 99 99 99 99 99 99 99
26 99 99 99 99 99 99 99 99 99 99 99 99
27 99 99 99 99 99 99 99 99 99 99 99 _
```

Here are statistics on the bytes sent for each connection, reported on a scale of 0-99 with 99 corresponding to 4 bytes:

```
sender
```

```
0 3 4 5 9 10 11 12 18 21 24 27
```

```
receiver
```

```
0 _ 99 99 99 99 99 99 99 99 99 99 99
1 99 99 99 99 99 99 99 99 99 99 99 99
2 99 99 99 99 99 99 99 99 99 99 99 99
3 99 _ 99 99 99 99 99 99 99 99 99 99
4 99 99 _ 99 99 99 99 99 99 99 99 99
5 99 99 99 _ 99 99 99 99 99 99 99 99
6 99 99 99 99 99 99 99 99 99 99 99 99
7 99 99 99 99 99 99 99 99 99 99 99 99
8 99 99 99 99 99 99 99 99 99 99 99 99
9 99 99 99 99 _ 99 99 99 99 99 99 99
10 99 99 99 99 99 _ 99 99 99 99 99 99
11 99 99 99 99 99 99 _ 99 99 99 99 99
12 99 99 99 99 99 99 99 _ 99 99 99 99
```

```
13 99 99 99 99 99 99 99 99 99 99 99 99
14 99 99 99 99 99 99 99 99 99 99 99 99
15 99 99 99 99 99 99 99 99 99 99 99 99
16 99 99 99 99 99 99 99 99 99 99 99 99
17 99 99 99 99 99 99 99 99 99 99 99 99
18 99 99 99 99 99 99 99 99 _ 99 99 99
19 99 99 99 99 99 99 99 99 99 99 99 99
20 99 99 99 99 99 99 99 99 99 99 99 99
21 99 99 99 99 99 99 99 99 99 _ 99 99
22 99 99 99 99 99 99 99 99 99 99 99 99
23 99 99 99 99 99 99 99 99 99 99 99 99
24 99 99 99 99 99 99 99 99 99 99 _ 99
25 99 99 99 99 99 99 99 99 99 99 99 99
26 99 99 99 99 99 99 99 99 99 99 99 99
27 99 99 99 99 99 99 99 99 99 99 99 _
```

The average length of point-to-point messages was 4 bytes per message.

Using `mpdump` to Convert Intermediate Binary Files to ASCII Files

The `mpdump` command converts each raw (unanalyzed) intermediate file that is generated by the MPI library into a readable ASCII file. The `mpdump` command produces files that have the `.txt` extension. For example, the following command creates a series of ASCII files in the current directory (the default) based on all the processes that are listed in the index file `mpprof.index.cre.14`:

```
% mpdump mpprof.index.cre.14
```

The `mpdump` Command Syntax

The syntax of the `mpdump` command is

```
mpdump [-h] [-V] [-S] [-p process-set] [-o output-directory]  
index-file
```

where the following command options are available:

TABLE 8-2 Options to the `mpdump` Command

Option	Description
-h	Lists the optional arguments and exits.
-V	Displays the program version during start up.
-S	Uses the secure <code>ssh</code> and <code>scp</code> commands instead of <code>rsh</code> and <code>rcp</code> .
-p	Specifies a subset of the processes to be included in the <code>mpdump</code> output. The <code>-p process-set</code> option is entered as a comma-separated list of tuples. If the <code>-p</code> option is used with <code>-S</code> and/or <code>-g</code> , the operation will be performed only on the specified subset. If a process set encompasses the complete set of ranks, the <code>-p</code> option will have no effect. If the <code>-p</code> option is given a stride, the upper bound of the stride is omitted. For example, if a stride is specified as <code>1:4:2</code> , the upper bound (4) is not included in the set. In this example, the resulting set is <code>{1,3}</code> .
-o	Specifies that the <code>mpdump</code> output files are to be written to the named directory. The named directory must already exist. If the specified <code>output-directory</code> does not exist, an error message will be issued. If <code>-o</code> is not specified, the output files will be written to the current directory.

A Sample `mpdump` File

The following command creates 28 ASCII text files that are based on the 28 files that were created in the previous section using the `mprun` command.

```
% mpdump mpprof.index.cre.39
```

Each of the 28 files contains profiling data about a process in the job. The contents of *one* of the files is shown below:

```
H: version="MPPProf 1.0" np=28 rank=0 timeperiod=60.000000;  
H: jobname="cre.39" pid=8399 date="Tue Oct 29 14:04:29 2002"  
H: maxfilesize=52428800 hostname="hpc-smp6-0"  
H: arg0="/home/mshonle/tests/first-string/connectivity"  
V: variable=MPI_POLLALL actual=1 user=<unset>;  
V: variable=MPI_PROCBIND actual=1 user=L;  
V: variable=MPI_SPIN actual=1 user=1;  
V: variable=MPI_CANONREDUCE actual=0 user=<unset>;  
V: variable=MPI_OPTCOLL actual=1 user=<unset>;  
V: variable=MPI_EAGERONLY actual=1 user=<unset>;
```

```

V: variable=MPI_COSCHED actual=2 user=<unset>;
V: variable=MPI_FLOWCONTROL actual=0 user=<unset>;
V: variable=MPI_FULLCONNINIT actual=0 user=<unset>;
V: variable=MPI_WARMUP actual=0 user=<unset>;
x:shm.2: H: version="MPPProf 1.0" hostname="hpc-smp6-0";
x:shm.2: V: variable=MPI_SHM_CPOOLSIZE actual=24576 user=<unset>;
x:shm.2: V: variable=MPI_SHM_CYCLESIZE actual=8192 user=<unset>;
x:shm.2: V: variable=MPI_SHM_CYCLESTART actual=24576 user=<unset>;
x:shm.2: V: variable=MPI_SHM_NUMPOSTBOX actual=16 user=<unset>;
x:shm.2: V: variable=MPI_SHM_PIPESIZE actual=8192 user=<unset>;
x:shm.2: V: variable=MPI_SHM_PIPESTART actual=2048 user=<unset>;
x:shm.2: V: variable=MPI_SHM_SBPOOLSIZE actual=0 user=<unset>;
x:shm.2: V: variable=MPI_SHM_SHORTMSGSIZE actual=256 user=<unset>;
x:shm.2: V: variable=MPI_SHM_RENDVSIZE actual=24576 user=<unset>;
x:shm.2: V: variable=MPI_SHM_GBPOOLSIZE actual=20971520 user=<unset>;
h: start=0.000030 end=0.008181 snapshot=1;
c: routine=MPI_Barrier time=0.005588 bytesent=0 bytesrecv=0 calls=1;
c: routine=MPI_Comm_rank time=0.000003 bytesent=0 bytesrecv=0 calls=1;
c: routine=MPI_Comm_size time=0.000002 bytesent=0 bytesrecv=0 calls=1;
c: routine=MPI_Recv time=0.001103 bytesent=0 bytesrecv=108 calls=27;
c: routine=MPI_Send time=0.000248 bytesent=108 bytesrecv=0 calls=27;
p: destrank=1 bytesent=4 msgssent=1;
p: destrank=2 bytesent=4 msgssent=1;
p: destrank=3 bytesent=4 msgssent=1;
p: destrank=4 bytesent=4 msgssent=1;
p: destrank=5 bytesent=4 msgssent=1;
p: destrank=6 bytesent=4 msgssent=1;
p: destrank=7 bytesent=4 msgssent=1;
p: destrank=8 bytesent=4 msgssent=1;
p: destrank=9 bytesent=4 msgssent=1;
p: destrank=10 bytesent=4 msgssent=1;
p: destrank=11 bytesent=4 msgssent=1;
p: destrank=12 bytesent=4 msgssent=1;
p: destrank=13 bytesent=4 msgssent=1;
p: destrank=14 bytesent=4 msgssent=1;
p: destrank=15 bytesent=4 msgssent=1;
p: destrank=16 bytesent=4 msgssent=1;
p: destrank=17 bytesent=4 msgssent=1;
p: destrank=18 bytesent=4 msgssent=1;
p: destrank=19 bytesent=4 msgssent=1;
p: destrank=20 bytesent=4 msgssent=1;
p: destrank=21 bytesent=4 msgssent=1;
p: destrank=22 bytesent=4 msgssent=1;
p: destrank=23 bytesent=4 msgssent=1;
p: destrank=24 bytesent=4 msgssent=1;
p: destrank=25 bytesent=4 msgssent=1;
p: destrank=26 bytesent=4 msgssent=1;

```



```
p: destrank=27 bytessent=4 msgssent=1;  
x:shm.2: alloc_mem=MPI_Alloc_mem no_memory_allocated=0;  
e:
```


Displaying Information With `mpinfo`

What You Can Do

Task	Option
How to display information about published names	-T
How to display information about any cluster	-c
How to display information about the current cluster	-C
How to display information about individual partitions	-p
How to display information about all partitions	-P
How to display information about individual nodes	-n
How to display information about all nodes	-N
How to display an online list of valid attributes	-lc -lp -ln
How to restrict output to individual attributes	-A
How to display information in verbose mode	-v

▼ How to Display Information About Published Names (-T)

Use the -T option:

```
% mpinfo -T
```

The name, associated port value, and the *jid* of the publishing job are displayed. For example:

```
% mpinfo -T
JID  NAME          PORT
14   bachelor      xxx.xxx.xxx.xxx:48944
14   biker         xxx.xxx.xxx.xxx:48944
14   centralia    xxx.xxx.xxx.xxx:48944
14   rockstar     xxx.xxx.xxx.xxx:48944
18   freddie      xxx.xxx.xxx.xxx:501
18   lackcluster  xxx.xxx.xxx.xxx:503
18   bennie       xxx.xxx.xxx.xxx:505
18   stellar      xxx.xxx.xxx.xxx:507
```

The fields in the output are described in TABLE 9-1.

Include the -A option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (-A)” on page 99.)

▼ How to Display Information About Any Cluster (-c)

Use the -c option:

```
% mpinfo -c [cluster] -C | -P | -N
```

If you do not enter a *cluster*, the command uses the cluster named by the SUNHPC_CLUSTER environment variable. If that environment variable has not been set, be sure to manually enter a *cluster*, or the command fails.

Use one of the three options `-C`, `-P`, or `-N` to indicate the type of information you want to display (cluster-level, partition, or node). Use only one option at a time. For example:

```

% mpinfo -c hpc-cluster-0 -C
NAME                ADMINISTRATOR DEF_INTER_PART
hpc-cluster-0 -          all

% mpinfo -c hpc-cluster-0 -P
NAME NODES: Tot(cpu) Enb(cpu) Onl(cpu) ENA LOG MP
all          1(28)    1(28)    1(28)    yes yes yes

% mpinfo -c hpc-cluster-0 -N
NAME  UP  PARTITION OS      OSREL  NCPU  FMEM   FSWP  LOAD1  LOAD5  LOAD15
node0 y  all      SunOS  5.8    10    748.07 1459  10.54  10.62  10.66
node1 y  all      SunOS  5.8    10    811.63 1492  10.51  10.53  10.55
node2 y  all      SunOS  5.8    10    715.10 1432  10.87  10.88  10.91
node3 y  all      SunOS  5.8    10    837.91 1514  10.06  10.24  10.31

```

The fields in the output are described in TABLE 9-1.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (`-A`)” on page 99.)

▼ How to Display Information About the Current Cluster (`-C`)

Use the `-C` option (upper case):

```

% mpinfo -C

```

This option is a shortcut for a common use of the `-c` option:

```

% mpinfo -C
NAME                ADMINISTRATOR DEF_INTER_PART
hpc-cluster-0 -          all

% mpinfo -c hpc-cluster-0 -C
NAME                ADMINISTRATOR DEF_INTER_PART
hpc-cluster-0 -          all

```

The fields in the output are described in TABLE 9-1.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (`-A`)” on page 99.)

▼ How to Display Information About Individual Partitions (`-p`)

Use the `-p` option:

```
% mpinfo -p partition-name[ ,partition-name ... ]
```

Separate multiple partition names with a comma. You can also enclose the set of partition names in quotation marks.

For example:

```
% mpinfo -p part1,part2
NAME          NODES: Tot(cpu) Enb(cpu) Onl(cpu) ENA LOG MP
part1         1( 4)   1( 4)   1( 4)  no  yes yes
part2         1( 4)   1( 4)   1( 4)  yes yes yes
```

The fields of the output are described in TABLE 9-1.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (`-A`)” on page 99.)

▼ How to Display Information About All Partitions (`-P`)

Use the `-P` option (upper case):

```
% mpinfo -P
```

For example:

```
% mpinfo -P
NAME          NODES: Tot(cpu) Enb(cpu) Onl(cpu) ENA LOG MP
part10        1( 4)   1( 4)   1( 4) no  yes yes
part11        1( 4)   1( 4)   1( 4) yes yes yes
```

The fields of the output are described in TABLE 9-1.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (`-A`)” on page 99.)

▼ How to Display Information About Individual Nodes (`-n`)

Use the `-n` option:

```
% mpinfo -n node-name[ , node-name... ]
```

When listing multiple node names, separate the names with commas but no spaces. For example:

```
% mpinfo -n node0,node1
NAME  UP  PARTITION OS      OSREL  NCPU  FMEM   FSWP   LOAD1  LOAD5  L
OAD1
node0 y  p0      Solaris
8    1    0.89   158.34 0.09   0.11  0.13
node1 y  p0      Solaris
8    1    31.41  276.12 0.00   0.01  0.01
```

The fields in the output are described in TABLE 9-1.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (`-A`)” on page 99.)

▼ How to Display Information About All Nodes (-N)

Use the `-N` option (upper case).

```
% mpinfo -N
```

For example:

```
% mpinfo -N
NAME  UP  PARTITION OS      OSREL NCPU FMEM   FSWP   LOAD1 LOAD5 L
OAD1
node0 y  p0      Solaris
8    1    0.89   158.34 0.09  0.11  0.13
node1 y  p0      Solaris
8    1    31.41  276.12 0.00  0.01  0.01
node2 y  p1      Solaris
8    1    25.59  279.77 0.00  0.00  0.01
node3 y  p1      Solaris
8    1    25.40  279.88 0.00  0.00  0.01
```

The fields in the output are described in TABLE 9-1.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See “How to Restrict Output to Individual Attributes (-A)” on page 99.)

▼ How to Display an Online List of Valid Attributes (-lc, -lp, -ln)

Use the `-lc`, `-lp`, or `-ln` options for clusters, partitions, or nodes:

```
% mpinfo -lc
% mpinfo -lp
% mpinfo -ln
```


For example:

```
% mpinfo -lc
name          cluster name (NAME)
admin         cluster administrator (ADMINISTRATOR)
definter      default interactive partition (DEF_INTER_PART)

% mpinfo -lp
name          partition name (NAME)
enabled       partition state (ENA)
nodes         node count (NODES: Tot(cpu) Enb(cpu) Onl(cpu))
maxt          max total procs (MAXT)
login         logins allowed (LOG)
mp            mp jobs allowed (MP)

% mpinfo -ln
cpu_idle      idle          cpu idle time (%) (IDLE)
cpu_iowait    iowait        cpu iowait time (%) (IWAIT)
cpu_kernel    kernel        cpu kernel time (%) (KERNL)
cpu_type      cpu           cpu architecture (CPU)
...
```

▼ How to Restrict Output to Individual Attributes (-A)

Add the `-A` option to other `mpinfo` options to restrict the output to specific attributes of a node, partition, or cluster:

```
% mpinfo -p partition -A attribute[ ,attribute... ]
% mpinfo -P -A attribute[ ,attribute... ]
% mpinfo -n node -A attribute[ ,attribute... ]
% mpinfo -N -A attribute[ ,attribute... ]
% mpinfo -c cluster [-C | -P | -N] -A attribute[ ,attribute... ]
% mpinfo -C partition -A attribute[ ,attribute... ]
```

Separate multiple partition attributes with commas but no spaces. For a list of valid attributes, see TABLE 9-1.

This example begins by showing the full set of node attributes displayed when you identify the object (in this case with the `-N` option), but leave out the `-A` option. Then it shows how adding the `-A` option restricts the list to a subset of the information:

```

% mpinfo -N
NAME UP PARTITION OS OSREL NCPU FMEM FSWP LOAD1 LOAD5 LOAD15
node0 y all SunOS 5.8 10 750.19 1527 10.52 10.66 10.70
node1 y all SunOS 5.8 10 816.07 1576 10.55 10.59 10.61
node2 y all SunOS 5.8 10 721.84 1524 10.91 10.95 10.96
node3 y all SunOS 5.8 10 840.41 1596 10.42 10.42 10.40

% mpinfo -N -A name
NAME
node0
node1
node2
node3

```

TABLE 9-1 Attributes Displayed by `-A` option to `mpinfo`

Object	Attribute	Description
Partition	NAME	Name of the partition
	NODES	Information about the nodes in the partition: Tot - total number of nodes Enb - number that are enabled Onl - number currently online ENA - whether the partition is enabled (YES/NO) LOG - whether the node accepts logins (YES/NO) MP - whether the node accepts multinode jobs (YES/NO)
	MAXT	Maximum number of simultaneously running processes allowed on each node of the partition
Cluster	NAME	Name of the cluster (host name of the master node)
	ADMINISTRATOR	Name of the cluster's administrator
	DEF_INTER_PART	Default interactive partition
Node	cpu_idle	Percent of time CPU is idle (IDLE).
	cpu_iowait	Percent of time CPU spends waiting for I/O (IWAIT).
	cpu_kernel	Percent of time CPU spends in kernel (KERNL).
	cpu_type	CPU architecture (CPU).

TABLE 9-1 Attributes Displayed by `-A` option to `mpinfo` (Continued)

Object	Attribute	Description
	<code>cpu_user</code>	Percent of time CPU spends running user's program (USER).
	<code>domain</code>	DNS domain.
	<code>enabled</code>	If set, node is available for spawning jobs on it.
	<code>load1</code>	Load average for the past minute (LOAD1).
	<code>load5</code>	Load average for the past five minutes (LOAD5).
	<code>load15</code>	Load average for the past 15 minutes (LOAD15).
	<code>manufacturer</code>	Hardware manufacturer (MANUFACTURER).
	<code>mem_free</code>	Node's available RAM (in Mbytes) (FMEM).
	<code>mem_total</code>	Node's total physical memory (in Mbytes) (MEM).
	<code>name</code>	Name of the node (NAME).
	<code>ncpus</code>	Number of CPU modules in the node (NCPU).
	<code>os_arch_kernel</code>	Node's kernel architecture (MACH).
Node (continued)	<code>os_max_proc</code>	Maximum number of processes allowed on the node (note that this is <i>all</i> processes, including cluster daemons) (MPROC).
	<code>os_name</code>	Name of the operating system running on the node (OS).
	<code>os_release</code>	Operating system's release number (OSREL).
	<code>os_release_maj</code>	The major number of the operating system release number (MAJ).
	<code>os_release_min</code>	The minor number of the operating system release number (MIN).
	<code>os_version</code>	Operating system's version (OSVER).
	<code>partition</code>	The partition of which the node is a member (PARTITION).
	<code>serial_number</code>	Hardware serial number (SERIAL).
	<code>swap_free</code>	Node's available swap space (in Mbytes) (FSWP).
	<code>swap_total</code>	Node's total swap space (in Mbytes) (SWAP).

▼ How to Display Information in Verbose Mode (-v)

Add the `-v` option to any other option to display its information in verbose mode:

```
% mpinfo -p partition -v
% mpinfo -P -v
% mpinfo -n node -v
% mpinfo -N -v
% mpinfo -c cluster [-C | -P | -N] -v
% mpinfo -C partition -v
```

Verbose mode displays a little more information than standard mode, and makes it easier to read. This example shows how the information is displayed first without, and then with the verbose mode:

```

% mpinfo -N
NAME UP PARTITION OS      OSREL NCPU FMEM   FSWP LOAD1 LOAD5 LOAD15
node0 y  all          SunOS 5.8   10   839.30 1610 10.18 10.57 10.65
node1 y  all          SunOS 5.8   10   900.45 1646 10.12 10.47 10.54
node2 y  all          SunOS 5.8   10   802.66 1592 10.42 10.78 10.84
node3 y  all          SunOS 5.8   10   927.55 1676 10.11 10.48 10.48

% mpinfo -N -v

Node "node0":
  LPM Interfaces: shm,tcp
  State:  enabled & online
  partition: "all"

  os: SunOS 5.8 (Generic_108528-07)
  arch: sun4u,  cpu: sparc,  ncpus: 10
  manufacturer: Sun_Microsystems,  serial no: 809deb49
  memory: 1280.000M (775.727M free),
  swap: 1932.539M (1579.609M free)
  isalist: sparcv9+vis sparcv9 sparcv8plus+vis sparcv8plus
  sparcv8 sparcv8-fsmuld sparcv7 sparc

  load averages: 10.53, 10.57, 10.64
  cpu states:  0.00% idle, 46.31% user, 53.69% kernel, 0.00% iowait

  local attributes:

Node "node1":
  LPM Interfaces: shm,tcp
  State:  enabled & online
  partition: "all"
.
.
.

```

Command Reference (mpinfo)

TABLE 9-2 Options for mpinfo

Command	Description
-T	Display name publishing information
-c	Display cluster-level, partition, or node information about any cluster
-C	Display cluster-level information about the current cluster; equivalent to <code>mpinfo -c cluster-name -C</code>
-p	Display information about individual partitions
-P	Displays information about all partitions in the cluster
-n	Displays information about individual nodes
-N	Displays information about all nodes in the cluster
-lc	List the cluster attributes that can be displayed by -A
-lp	List the partition attributes that can be displayed by -A
-ln	List the node attributes that can be displayed by -A
-A	Restrict the display of cluster, partition, or node information to individual attributes—must combine with one of these options: -c -C -p -P -n -N
-v	Display information in verbose mode—must combine with one of these options: -c -C -p -P -n -N

Troubleshooting

This appendix describes some common problem situations, resulting error messages, and suggestions for fixing the problems. Sun MPI error reporting, including I/O, follows the *MPI-2 Standard*. By default, errors are reported in the form of standard error classes. These classes and their meanings are listed in TABLE A-1 (for non-I/O MPI) and TABLE A-2 (for MPI I/O), and are also available on the MPI man page.

Three predefined error handlers are available in Sun MPI:

- `MPI_ERRORS_RETURN` – The default, returns an error code if an error occurs.
- `MPI_ERRORS_ARE_FATAL` – I/O errors are fatal, and no error code is returned.
- `MPI_THROW_EXCEPTION` – A special error handler to be used only with C++.

MPI Messages

You can make changes to and get information about the error handler using any of the following routines:

- `MPI_Comm_create_errhandler`
- `MPI_Comm_get_errhandler`
- `MPI_Comm_set_errhandler`

Messages resulting from an MPI program fall into two categories:

- *Error messages* – Error messages stem from within MPI. Usually an error message explains why your program cannot complete, and the program aborts.
- *Warning messages* – Warnings stem from the environment in which you are running your MPI program and are usually sent by `MPI_Init()`. They are not associated with an aborted program, that is, programs continue to run despite warning messages.

Error Messages

Sun MPI error messages use a standard format:

`[xyz] Error in function_name: errclass_string:intern(a): description: unixerrstring`

where

- `[xyz]` is the *process communication identifier*, and:
 - `x` is the job ID (or jid).
 - `y` is the name of the communicator if a name exists; otherwise it is the address of the opaque object.
 - `z` is the rank of the process.

The process communication identifier is present in every error message.

- *function_name* is the name of the associated MPI function. It is present in every error message.
- *errclass_string* is the string associated with the MPI error class. It is present in every error message.
- *intern* is an internal function. It is optional.
- `a` is a system call, if one is the cause of the error. It is optional.
- *description* is a description of the error. It is optional.
- *unixerrstring* is the UNIX error string that describes system call `a`. It is optional.

Warning Messages

Sun MPI warning messages also use a standard format:

`[xyz] Warning message`

where *message* is a description of the error.

Standard Error Classes

Listed below are the error return classes you may encounter in your MPI programs. Error values may also be found in `mpi.h` (for C), `mpif.h` (for Fortran), and `mpi++.h` (for C++).

TABLE A-1 Sun MPI Standard Error Classes

Error Code	Value	Meaning
<code>MPI_SUCCESS</code>	0	Successful return code.
<code>MPI_ERR_BUFFER</code>	1	Invalid buffer pointer.
<code>MPI_ERR_COUNT</code>	2	Invalid count argument.
<code>MPI_ERR_TYPE</code>	3	Invalid datatype argument.
<code>MPI_ERR_TAG</code>	4	Invalid tag argument.
<code>MPI_ERR_COMM</code>	5	Invalid communicator.
<code>MPI_ERR_RANK</code>	6	Invalid rank.
<code>MPI_ERR_ROOT</code>	7	Invalid root.
<code>MPI_ERR_GROUP</code>	8	Null group passed to function.
<code>MPI_ERR_OP</code>	9	Invalid operation.
<code>MPI_ERR_TOPOLOGY</code>	10	Invalid topology.
<code>MPI_ERR_DIMS</code>	11	Illegal dimension argument.
<code>MPI_ERR_ARG</code>	12	Invalid argument.
<code>MPI_ERR_UNKNOWN</code>	13	Unknown error.
<code>MPI_ERR_TRUNCATE</code>	14	Message truncated on receive.
<code>MPI_ERR_OTHER</code>	15	Other error; use <code>Error_string</code> .
<code>MPI_ERR_INTERN</code>	16	Internal error code.
<code>MPI_ERR_IN_STATUS</code>	17	Look in status for error value.
<code>MPI_ERR_PENDING</code>	18	Pending request.
<code>MPI_ERR_REQUEST</code>	19	Illegal <code>MPI_Request()</code> handle.
<code>MPI_ERR_KEYVAL</code>	36	Illegal key value.
<code>MPI_ERR_INFO</code>	37	Invalid info object.
<code>MPI_ERR_INFO_KEY</code>	38	Illegal info key.

TABLE A-1 Sun MPI Standard Error Classes (Continued)

Error Code	Value	Meaning
MPI_ERR_INFO_NOKEY	39	No such key.
MPI_ERR_INFO_VALUE	40	Illegal info value.
MPI_ERR_TIMEOUT	41	Timed out.
MPI_ERR_RESOURCES	42	Out of resources.
MPI_ERR_TRANSPORT	43	Transport layer error.
MPI_ERR_HANDSHAKE	44	Error accepting/connecting.
MPI_ERR_SPAWN	45	Error spawning.
MPI_ERR_WIN	46	Invalid window.
MPI_ERR_BASE	47	Invalid base.
MPI_ERR_SIZE	48	Invalid size.
MPI_ERR_DISP	49	Invalid displacement.
MPI_ERR_LOCKTYPE	50	Invalid locktype.
MPI_ERR_ASSERT	51	Invalid assert.
MPI_ERR_RMA_CONFLICT	52	Conflicting accesses to window.
MPI_ERR_RMA_SYNC	53	Erroneous RMA synchronization.
MPI_ERR_NO_MEM	54	Memory exhausted.
MPI_ERR_LASTCODE	55	Last error code.

MPI I/O message are listed separately, in TABLE A-2.

MPI I/O Error Handling

Sun MPI I/O error reporting follows the *MPI-2 Standard*. By default, errors are reported in the form of standard error codes (found in `/opt/SUNWhpc/include/mpi.h`). Error classes and their meanings are listed in TABLE A-2. They can also be found in `mpif.h` (for Fortran) and `mpi++.h` (for C++).

You can change the default error handler by specifying `MPI_FILE_NULL` as the file handle with the routine `MPI_File_set_errhandler()`, even if no file is currently open. Or, you can use the same routine to change a specific file's error handler.

TABLE A-2 Sun MPI I/O Error Classes

Error Class	Value	Meaning
<code>MPI_ERR_FILE</code>	20	Bad file handle.
<code>MPI_ERR_NOT_SAME</code>	21	Collective argument not identical on all processes.
<code>MPI_ERR_AMODE</code>	22	Unsupported <code>amode</code> passed to open.
<code>MPI_ERR_UNSUPPORTED_DATAREP</code>	23	Unsupported <code>datarep</code> passed to <code>MPI_File_set_view()</code> .
<code>MPI_ERR_UNSUPPORTED_OPERATION</code>	24	Unsupported operation, such as seeking on a file that supports only sequential access.
<code>MPI_ERR_NO_SUCH_FILE</code>	25	File (or directory) does not exist.
<code>MPI_ERR_FILE_EXISTS</code>	26	File exists.
<code>MPI_ERR_BAD_FILE</code>	27	Invalid file name (for example, path name too long).
<code>MPI_ERR_ACCESS</code>	28	Permission denied.
<code>MPI_ERR_NO_SPACE</code>	29	Not enough space.
<code>MPI_ERR_QUOTA</code>	30	Quota exceeded.
<code>MPI_ERR_READ_ONLY</code>	31	Read-only file system.
<code>MPI_ERR_FILE_IN_USE</code>	32	File operation could not be completed, as the file is currently open by some process.
<code>MPI_ERR_DUP_DATAREP</code>	33	Conversion functions could not be registered because a data representation identifier that was already defined was passed to <code>MPI_REGISTER_DATAREP</code> .
<code>MPI_ERR_CONVERSION</code>	34	An error occurred in a user-supplied data-conversion function.
<code>MPI_ERR_IO</code>	35	I/O error.
<code>MPI_ERR_INFO</code>	37	Invalid info object.
<code>MPI_ERR_INFO_KEY</code>	38	Illegal info key.

TABLE A-2 Sun MPI I/O Error Classes (Continued)

Error Class	Value	Meaning
MPI_ERR_INFO_NOKEY	39	No such key.
MPI_ERR_INFO_VALUE	40	Illegal info value.
MPI_ERR_LASTCODE	55	Last error code.

Exceeding the File Descriptor Limit

If your application attempts to open a file descriptor when the maximum limit of open file descriptors has been reached, the job will fail and display the following message:

```
Too many open file descriptors
```

Should this occur, increase the value of the file descriptor hard limit before starting your job again.

If you are logged in to a C shell as superuser, you can determine the current hard limit value via the `limit` function, as follows:

```
# limit -h descriptors
```

If you are logged in to a Bourne shell as superuser, use the `ulimit` function.

```
# ulimit -Hn
```

Each function returns the file descriptor hard limit that was in effect. Once you know what the previous hard limit was, you can estimate what the new hard limit value should be and set it accordingly.

From a C shell, use the `limit` command to set the new value in the `.login` file.

```
# limit -h descriptors limit
```

From a Bourne shell, use the `ulimit` command to set the new value in the `.profile` file.

```
# ulimit -Hn limit
```

In each case, *limit* is the value of the new hard limit.

Alternatively, you can determine whether the file descriptor hard limit is anything other than the default by looking in the `/etc/system` file to see whether the `rlim_fd_max` parameter has been set to a nondefault value. If not, the file descriptor hard limit will be 1024. To change the hard limit in a 64-bit Solaris 8 environment, simply add the following line to the `/etc/system` file:

```
set rlim_fd_max=limit
```

Again, *limit* is the value of the new file descriptor hard limit.

You can also increase the file descriptor hard limit in a Solaris 8 32-bit environment. However, this approach is not recommended. See “Maximum Number of File Descriptors” on page 45 for information about defining the C pre-processor symbol `FD_SETSIZE` should you choose to make such a change.

Exceeding the TCP Port Limit

If you are running a large (highly parallel), communication-intensive MPI job on a Sun HPC cluster that includes both of the following conditions,

- TCP/IP as the only interconnect medium
- A node that has more than 32 CPUs

the number of TCP ports may be too limited. If the MPI job attempts to access a TCP port when no more are available, the job will fail and print the following message:

```
low level communications error: Cannot assign requested address
```

Most likely, this occurs only when the job is running on the configuration described above *and* one of the following conditions exists:

- `MPI_FULLCONNINIT` is set.
- `MPI_Alltoall` is used.
- The application includes its own all-to-all code.

Other activity on the cluster, such as file I/O or other MPI jobs, will increase the chance of this occurring.

You can avoid exceeding the TCP port limit by taking one or more of the following steps:

- Configure the node with more than 32 nodes into two or more domains. From the TCP perspective, each domain will be seen as a separate node with its own supply of TCP ports.
- Reconfigure the cluster to exclude the node with more than 32 CPUs.
- Avoid running multiple MPI jobs or other tasks that would compete for available TCP ports.
- If two large MPI jobs must run on the same cluster, wait a few minutes between the jobs to give the OS time to reclaim the ports created for the previous job.
- If the application does not include any all-to-all operations, use the default lazy connections mode instead of `MPI_FULLCONNINIT`.
- If the application contains any all-to-all operations, either `MPI_Alltoall` or custom code, use a non-TCP network technology.

Index

SYMBOLS

, 34
!, 34
!=, 34
/dev/null, how to read input from, 41
=, 34
>, 34
>=, 34
>>, 34

A

argument vector, how to redirect output with, 40
attribute
 how to display a list of valid cluster, node, or
 partition attributes, 98
 how to restrict output to individual
 attributes, 99
 list of, displayed by `mpinfo -A`, 100
attributes
 custom configuration attributes, 42
 job attributes displayed by `mpps`, 73

B

background, how to move a process to the, 49
block, distribute processes by, 29

C

cluster
 about, 7
 how to display a list of valid attributes, 98
 how to display information about any, 94
 how to display information about the current, 95
 how to run a program on a different cluster, 21
 partitions, 8
ClusterTools Runtime Environment, 2
command line interface (CRE), 2
configuration
 how to redirect output custom, 41
configurations, supported, 1
controlling input / output, 38
Controlling where a program runs, 20
`cpu_idle`, 33
`cpu_iowait`, 33
`cpu_kernel`, 33
`cpu_type`, 33
`cpu_user`, 33
CRE, 2

D

default settings
 how to run a program with, 21
`default_interactive_partition` attribute, 9
documentation
 LSF on web, xiii
 MPI Reference Manual, xi

product notes, xi
domains, 8

E

environment variable
 MP_JOBID, 19
 MP_NPROCS, 19
 MP_RANK, 19
 MPRUN-FLAGS, 18
 SUNHPC_PART, 9
error classes, standard, 107
error classes, Sun MPI I/O, 109
error handling, MPI I/O, 108
error messages
 about, 105
 format, 106
exceeding the file descriptor limit, 110
exceeding the TCP port limit, 111

F

File descriptor
 exceeding the limit, 110
file descriptor, 41
 maximum number, 45
 redirecting output to other, 44
 redirecting their output to a file, 44

G

group name, how to use a different, 50

H

help, how to display, 51
How to
 change the working directory, 49
 disable process spawning, 23
 display a job's start time, 74
 display an online list of valid attributes, 98
 display command help, 51
 display information about all jobs, 74
 display information about all nodes, 98

display information about all partitions, 96
display information about any cluster, 94
display information about individual jobs, 73
display information about individual nodes, 97
display information about individual
 partitions, 96
display information about the current cluster, 95
display information in verbose mode, 102
display job information by partition, 74
display job information by process, 75
display job status information, 52
display the command's version, 52
distribute processes among nodes, 27
distribute processes by block, 29
distribute processes by rankmap, 30
enable process spawning, 23
include independent nodes, 25
include shell-specific actions, 48
kill a running program, 68
move a process to the background, 49
read standard input from /dev/null, 41
redirect output to individual files, 39
redirect output to mprun, 39
redirect with a custom configuration, 41
redirect with an argument vector, 40
restrict output to individual attributes, 99
run a job on a different project, 50
run a program as multiple processes, 22
run a program on a different cluster, 21
run a program on a different partition, 21
run a program with default settings, 21
select nodes by resource requirement, 32
send a signal to a job, 69
settle for available processes, 24
share nodes, 23
shut off all standard I/O, 40
tag output with its rank number, 52
use a different group name, 50
use a different user name, 49
wrap multiple processes, 24
how to
 display a list of supported signals, 68

I

interconnect technology, preferred, 2

J

- job
 - how to display information about all jobs, 74
 - how to display information by partition, 74
 - how to display information by process, 75
 - how to display start time, 74
 - how to display status information, 52
- job status
 - how to display, 72

K

- killing a program, how to, 68
- killing programs with mpkill, 67

L

- limit -h, 110
- load balancing
 - about, 10
- load1, 33
- load15, 33
- load5, 33
- login partition, 9

M

- manufacturer, 33
- mapping MPI processes to nodes, 27
- max_total_procs, 30
- mem_free, 33
- mem_total, 33
- messages, MPI, 105
- MPI
 - Sun MPI, 3
 - Sun MPI I/O, 3
- MPI messages, 105
- MPI_ERRORS_ARE_FATAL, 105
- MPI_ERRORS_RETURN, 105
- MPI_THROW_EXCEPTION, 105
- mpinfo
 - A, 99
 - C, 95

- c, 94
- lc, 98
- ln, 98
- lp, 98
- N, 98
- n, 97
- P, 96
- p, 96
- V, 102
- what you can do, 93

mpkill

- l, 68
- return values, 67
- what you can do, 67

mpkill-d, 68

mpps

- A, 74
- a, 74
- e, 74
- f, 74
- J, 73
- P, 75
- p, 75
- what you can do, 71

mprun

- A, 40
- B, 39
- C, 49
- c, 21
- D, 39
- d, 52
- default settings, 21
- G, 50
- h, 51
- I, 41
- J, 52
- j, 23
- l, 27
- m, 30
- N, 40
- n, 41
- np, 22
- nr, 31
- Ns, 23
- o, 52
- P, 50
- p, 21
- R, 32

- rank-spec, 28
- S, 24
- syntax, 17
- U, 49
- u, 25
- V, 52
- v, 50
- W, 24
- x, 56
- Ys, 23
- Z, 29
- Zt, 29

N

- name (resource), 33

- node

- how to display a list of valid attributes, 98
 - how to display information about all nodes, 98
 - how to display information about individual nodes, 97
 - how to distribute processes among, 27
 - how to include independent, 25
 - how to select by resource requirement, 32
 - how to share, 23
 - mapping MPI processes to, 27

- nodes

- about, 7
 - independent, 8

O

- os arch kernel, 33

- os_max_proc, 33

- os_name, 33

- os_release, 34

- os_release_maj, 34

- os_release_min, 34

- os_version, 34

- output

- how to redirect to mprun, 39

P

- partition

- about, 8

- enabling and selecting, 8

- how to display a list of valid attributes, 98

- how to display information about all partitions, 96

- how to display information about individual partitions, 96

- how to display job information by, 74

- how to run a program on a different, 21

- login, 9

- selection criteria, 9

- precedence

- about, 17

- for input/output, 39

- for mapping processes to nodes, 27

- for program execution, 20

- Prism

- about, 4

- process

- how to display job information by, 75

- how to distribute among nodes, 27

- how to distribute by block, 29

- how to distribute by rankmap, 30

- how to move to the background, 49

- how to run a program as multiple, 22

- how to wrap, 24

- mapping to nodes, 27

- pid, 10

- settling for available, how to, 24

- spawning, how to disable, 23

- spawning, how to enable, 23

- processes

- about, 10

- program

- displaying program information with mpps, 71

R

- rank

- how to tag output with rank number, 52

- rankmap, 30

- how to distribute processes by, 30

- rankmap file, 31

- rankmap file, 31

- rank-spec, 28

- redirect output to individual files, how to, 39

- redirect output to mprun, how to, 39
- redirecting file descriptor output to a file, 44
- redirecting output to other file descriptors, 44
- resource requirement
 - examples of, 35
 - how to select nodes by, 32
 - operators, list of, 34
 - predefined resources, list of, 33
 - resource requirement spec, 32
- runtime environment, 2

S

- S3L
 - Sun S3L, about, 5
- scalability, 1
- serial_number, 34
- sharing, nodes, how to, 23
- shell, how to include shell-specific actions, 48
- signal
 - how to send to a job, 69
 - SIGTERM, 69
- signals, how to display a list of supported, 68
- spawning, process, how to disable, 23
- spawning, processes, how to enable, 23
- standard error, how mprun handles, 38
- standard output, how mprun handles, 38
- status
 - how to display job status, 72
- status, how to display job status information, 52
- stream-number, 42
- SUNHPC_PART environment variable, 9
- swap_free, 34
- swap_total, 34

T

- TCP port limit, exceeding, 111
- total_max_procs, 27
- troubleshooting, 105

U

- ulimit -Hn, 110
- user name, how to use a different, 49

V

- verbose, how to display information in verbose mode, 102
- version, how to display, 52

W

- warning messages
 - about, 105
 - format, 106
- working directory, how to change the, 49

