Sun™ HPC ClusterTools 4 User's Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 U.S.A.
650-960-1300

Part No 816-0650-10
August 2001, Revision A

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

Contents

Tables

Preface

This manual describes how to execute Sun™ MPI jobs on systems running Sun HPC ClusterTools™ software with the Sun HPC ClusterTools Runtime Environment (CRE). If you are using Platform Computing Corporation's Load Sharing Facility (LSF) Suite, see Appendix A.

Before You Read This Book

Product notes for the other components in the suite are included in *Sun HPC ClusterTools 4 Product Notes*. For information about writing MPI programs, refer to the *Sun MPI 5.0 Programming and Reference Guide.* If you are using LSF instead of CRE, see the documentation that came with the LSF Suite, especially the *LSF Batch User's Guide*.

Using UNIX Commands

This document does not describe how to use basic UNIX® commands. For that type of information, see:

- AnswerBook2™ online documentation for the Solaris™ software environment
- Other software documentation that you received with your system

Typographic Conventions

TABLE P-1 Typographic Conventions

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

Shell Prompts

TABLE P-2 Shell Prompts

Shell	Prompt
C shell	%
C shell superuser	#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

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.

TABLE P-3 Related Documentation

Application	Title	Part Number
Sun HPC	*Read Me First: Guide to Sun HPC ClusterTools Documentation*	816-0646-10
Sun HPC ClusterTools software	*Sun HPC ClusterTools 4 Product Notes*	816-0647-10
	Sun HPC ClusterTools 4 Installation Guide	816-0648-10
	Sun HPC ClusterTools 4 Performance Guide	816-0656-10
	Sun HPC ClusterTools 4 Administrator's Guide	816-0649-10
Sun MPI Programming	*Sun MPI 5.0 Programming and Reference Guide*	816-0651-10
Sun S3L	*Sun S3L Programming Guide*	816-0652-10
	Sun S3L Reference Manual	816-0653-10
Prism™ graphical programming environment	*Prism User's Guide*	816-0654-10
	Prism Reference Manual	816-0655-10

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 Online

A broad selection of Sun system documentation is located at:

`http://www.sun.com/products-n-solutions/hardware/docs`

A complete set of Solaris documentation and many other titles are located at:

`http://docs.sun.com`

Ordering Sun Documentation

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at:

`http://www.fatbrain.com/documentation/sun`

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 (806-0650-10) of your document in the subject line of your email.

CHAPTER 1

Introduction to Sun HPC ClusterTools

Sun HPC ClusterTools 4 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.

Supported Configurations

Sun HPC ClusterTools 4 software requires the Solaris 8 (32-bit or 64-bit) operating environment. All programs that execute under Solaris 8 will execute in the Sun HPC ClusterTools environment.

Sun HPC ClusterTools 4 software supports three versions of the Forte™ Developer compilers: Forte 6, Forte 6 update 1, and Forte 6 update 2.

Sun HPC ClusterTools 4 software can run MPI jobs of up to 2048 processes on as many as 64 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™ series high-performance cluster interconnect, when it becomes available. 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.

Any MPI application that uses a TCP/IP network for message passing will incur the normal latencies that are inherent in TCP operations. Also, under certain exceptional (and avoidable) circumstances, an MPI application that uses a TCP/IP network for high-volume message passing may experience a limitation in TCP port availability. If you expect to run jobs with hundreds of processes using a TCP/IP network for message passing, read "Exceeding the TCP Port Limit" on page 101 for additional information.

Sun HPC ClusterTools Runtime Environment (CRE)

Sun HPC ClusterTools 4 software provides a command line interface (also called CRE for *ClusterTools Runtime Environment*) that starts jobs and provides information. It uses four commands to perform four primary functions:

- Executes Programs With mprun
- Kills Programs With mpkill
- Displays Job Information With mpps
- Displays Node Information With mpinfo

Executes Programs With `mprun`

Sun HPC ClusterTools 4 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.

Kills Programs With `mpkill`

The runtime environment uses the `mpkill` command to kill jobs in progress and send them signals. Its syntax and use are described in Chapter 5.

Displays Job Information With `mpps`

The runtime environment uses the `mpps` command to display information about jobs and their processes. Its syntax and use are described in Chapter 6.

Displays Node Information With `mpinfo`

The runtime environment uses the `mpinfo` command to display information about nodes and their partitions. Its syntax and use are described in Chapter 7.

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 most of the MPI 2.0 standard. Its highlights are:

- Integration with the Sun HPC ClusterTools Runtime Environment (CRE) and Platform Computing's Load Sharing Facility (LSF) Suite
- 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

Sun Parallel File System

The Sun Parallel File System (PFS) provides high-performance file I/O for multiprocess applications running in a cluster-based, distributed-memory environment.

PFS file systems closely resemble UFS file systems, but provide significantly higher file I/O performance by striping files across multiple PFS I/O server nodes. This striping reduces the time required to read or write a PFS file by an amount roughly proportional to the number of file server nodes in the PFS file system.

PFS uses a conventional inverted-tree hierarchy, with a `root` directory at the top and subdirectories and files branching down from there. The fact that individual PFS files are distributed across multiple disks managed by multiple I/O servers is transparent to the programmer. The way that PFS files are actually mapped to the physical storage facilities is based on file system configuration entries in the CRE database.

PFS is optimized for the large files and complex data access patterns that are characteristic of parallel scientific applications.

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
- Analyze performance of MPI programs
- 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^{++}.

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)
- 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 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^{++}.

LSF Suite

The Sun HPC ClusterTools products can be teamed with the LSF Suite, Platform Computing's resource management software.

The LSF Suite is a collection of resource-management products that provide distributed batch scheduling, load-balancing, job execution, and job termination services across a network of computers. The LSF products required by Sun HPC ClusterTools 4 software are:

- *LSF Base* – Provides the fundamental services upon which LSF Batch and LSF Parallel depend. It supplies cluster configuration information as well as the up-to-date resource and load information needed for efficient job allocation. It also supports interactive job execution.
- *LSF Batch* – Performs batch job processing, load-balancing, and policy-based resource allocation.
- *LSF Parallel* – Extends the LSF Base and Batch services with support for parallel jobs.

The LSF Suite is described in Appendix A.

Note – Read the *LSF Batch User's Guide* for detailed information about the LSF Batch system's general features.

CHAPTER 2

Fundamental Concepts

This chapter summarizes a few basic concepts that you should understand to get the most out of Sun's HPC ClusterTools.

Clusters and Nodes

High performance computing clusters[1] are groups of Sun symmetric multiprocessor (SMP) servers interconnected by any Sun-supported, TCP/IP-capable interconnect or by the Sun Fire series high-speed interconnect, when it becomes available. 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 for your MPI programs, and distribute processes among them. For instructions, see Chapter 4, "Running Programs with mprun."

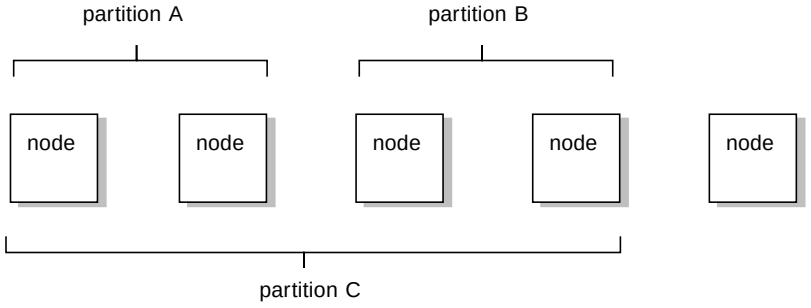
1. SunCluster™ is a completely different technology used for high availability (HA) applications.

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.

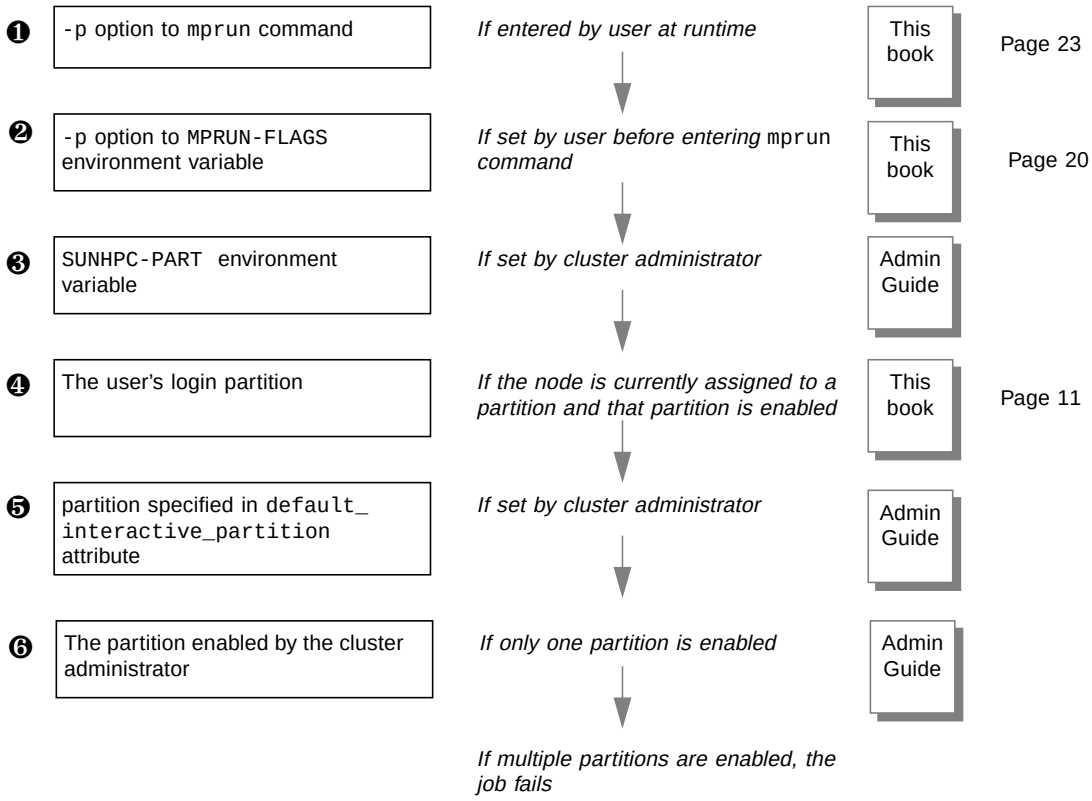
How Partitions Are Enabled and Selected

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 4 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 71.

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 `mprun` 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 6.

How LSF Works

If you are using LSF for resource management, all Sun HPC jobs are handled by the LSF Batch system. Consequently, Sun HPC job submission involves the following:

- When a Sun HPC job is submitted, it is placed in a job queue rather than being started immediately.
- 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 LSF 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.

The LSF Batch system offers an enhanced form of queue-based job scheduling, called *interactive batch*. This job submission mode provides all the job scheduling and resource management services of the batch environment, while keeping the terminal session from which the job was submitted attached to the job. This allows the user to interact with the job throughout its execution.

CHAPTER 3

Before You Begin

This chapter provides miscellaneous information about the runtime environment that you should know before you begin running programs with it.

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 Programming Guide*, the *Sun MPI Programming and Reference Guide*, and the *Sun HPC ClusterTools 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 this directory to your path; check with your system administrator.

The man pages for Sun HPC 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 `kinit` command before running any CRE command.
DES	While DES authentication is in use, you must issue the `keylogin` command before issuing any CRE command.

See your system administrator for details.

Core Files

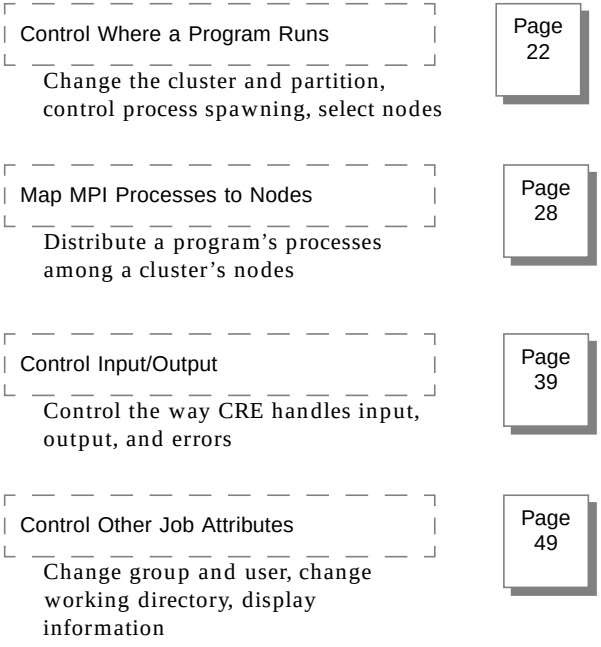
Core files are produced as they normally are in the Solaris environment. However, if more than one process dumps core in a multiprocess program, the resulting core file may be overwritten in the same directory. Use `coreadmin`(1M) to control the naming and placement of core files.

CHAPTER 4

Running Programs with `mprun`

What You Can Do

The `mprun` command controls several aspects of program execution. This diagram summarizes what you can do with the command, and where to find instructions.



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 4-4 on page 54 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.

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 4-4` on page 54.) 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:

MP_RANK	The rank of a process in the job: 0 - 2047
MP_NPROCS	The number of processes in a job: 1 - 2048
MP_JOBID	The *jid* 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:

MP_RANK	0 through 5
MP_NPROCS	6
MP_JOBID	The same *jid* that can be displayed by mpps (see Chapter 6)

Controlling Where the Program Runs

To Perform This Task	Use This Option	Described On
▼ How to run with default settings		Page 23
▼ How to run on a different cluster	`-c`	Page 23
▼ How to run on a different partition	`-p`	Page 23
▼ How to run as multiple processes	`-np`	Page 23
▼ How to share nodes	`-j`	Page 24
▼ How to enable process spawning	`-Ys`	Page 25
▼ How to disable process spawning	`-Ns`	Page 25
▼ How to wrap multiple processes	`-W`	Page 25
▼ How to settle for available processes	`-S`	Page 26
▼ How to include independent nodes	`-u`	Page 27

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`
`–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 69. 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 10, 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 25) or `-S` (Page 26) 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.

▼ 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* 13).

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

```
% mprun –j 85 a.out

% mprun -j 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 25).

▼ 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 27.)

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 25, 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, 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.

Mapping MPI Processes to Nodes

To Perform This Task	Use This Option	Described On
▼ How to distribute processes among nodes	`-l`	Page 29
▼ How to distribute processes by block	`-Z or -Zf`	Page 30
▼ How to distribute processes by rank map	`-Mf`	Page 31
▼ How to select nodes by resource requirement	`-R`	Page 33

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

Five primary `mprun` options affect rank placement: `–Mf`, `–Z`, `–Zt`, `–j`, and `–R`. 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
`–j`	`–Mf –Z –Zt –j –R`	`–u`
`–Mf`	`–Mf –Z –Zt –j –R`	`–Ns –Ys`
`–R`	`–Mf –Z –Zt –j –R`	
`–Z`	`–Mf –Z –Zt –j`	`–Ns –Ys`
`–Zt`	`–Mf –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 23) 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 *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 (`-Mf`)

Note – This option is being deprecated; functionality will be removed after this release. Use the `-l` option instead, as described in "How to Distribute Processes Among Nodes (`-l`)" on page 29.

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

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

Restrictions

The rank map specified with the `–Mf` 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.

map

The *map* argument can be:

map --> inline-rank-map | *rank-map-file*

inline-rank-map

An inline rank map has this syntax:

rank-map --> "**,***node-name* [*process-count*]**, ...**"

Each *node-name* is assigned the *process-count* you specify, in order. You can repeat the same *node-name* in the rank map. The *process-count* argument is optional. If you omit it, one process is assigned to the node. For example, here are two inline rank maps:

```
% mprun -np 3 -Mf ",mars, venus, jupiter" a.out
% mprun -np 2 -Mf ",mars, venus" a.out
```

The first rank map assigns one process to each of the three nodes, `mars`, `venus`, and `jupiter`. The second rank map assigns one process to `mars` and one to `venus`. Here are three more rank maps:

```
% mprun -np 6 -Mf ",mars 2, venus 2, jupiter 2" a.out
% mprun -np 5 -Mf ",mars 2, venus, jupiter 2" a.out
% mprun -np 5 -Mf ",mars 2, venus, mars 2" a.out
```

The first rank map assigns the first two processes to `mars`, the second two processes to `venus`, and the next two to `jupiter`. The second rank map assigns the first two processes to `mars`, the next one to `venus`, and the next two to `jupiter`. The third rank map assigns the first two processes to `mars`, the next one to `venus`, and the next two to `mars`.

rank-map-file

A rank map file has this syntax:

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

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 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 4-1 on page 34.

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 4-2 on page 35.

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 4-1 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 7.

TABLE 4-1 Predefined Resources

Resource	Description
`cpu_idle`	Percent of time that the CPU is idle.
`cpu_iowait`	Percent of time that the CPU spends waiting for I/O.
`cpu_kernel`	Percent of time that the CPU spends in the kernel.
`cpu_type`	CPU architecture.
`cpu_user`	Percent of time that the CPU spends running user's program.
`load1`	Node's load average for the past minute.
`load5`	Node's load average for the past 5 minutes.
`load15`	Node's load average for the past 15 minutes.
`manufacturer`	Hardware manufacturer.
`mem_free`	Nodes's available memory, in Mbytes.
`mem_total`	Node's total physical memory, in Mbytes.
`name`	Node's host name.

TABLE 4-1 Predefined Resources *(Continued)*

Resource	Description
os_max_proc	Maximum number of processes allowed on the node, including cluster daemons.
os_arch_kernel	Node's kernel architecture.
os_name	Operating system's name.
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 4-2 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	Described On
▼ How to redirect output to `mprun`	`-D`	Page 40
▼ How to redirect output to individual files	`-B`	Page 40
▼ How shut off all standard I/O	`-N`	Page 41
▼ How to redirect with an argument vector	`-A`	Page 41
▼ How read standard input from `/dev/null`	`-n`	Page 42
▼ How to redirect with a custom configuration	`-I`	Page 42

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 (`stdin`)	`0`
standard output (`stdout`)	`1`
standard error (`stderr`)	`2`

attribute

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

Attribute	Description	Dependencies
`r`	Read from the stream	
`w`	Write to the stream	
`p`	Attach the stream to a pseudo-terminal (`pty`)	
`b`	Input only goes to the first process	Must use with `r`
`l`	Make the output line-buffered	Must use with `w`
`t`	Tag the line-buffered output with rank number	Must use with `w`
`a`	Append the stream to a file	Must use with `w`
`m`	Echo keystrokes multiple times for multiple processes	Must use with `rp`

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 "1w=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, 1wl" 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, 1wt" 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 standout 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 4-3 `mprun` I/O Shortcut Summary

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

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	Described On
▼ How to include shell-specific actions		Page 49
▼ How to move a process to the background		Page 50
▼ How to change the working directory	`-C`	Page 50
▼ How to use a different user name	`-U`	Page 51
▼ How to use a different group name	`-A`	Page 51
▼ How to display command help	`-h`	Page 52
▼ How to display the command's version number	`-V`	Page 53
▼ How to display job status information	`-j`	Page 53
▼ How to tag output with its rank number	`-o`	Page 53

▼ 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 42.

```
% 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
```

▼ 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 Display Command Help (-h)

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

```
% mprun -h
USAGE:   mprun {options} [-] <exec> [<arg1> [<arg2> ...]]
  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
        -C <path>    Specify an alternate working directory
        -r <path>    Chroot to working dir before execution
        -J           Show job id after exec
        -np <procs>  Specify the number of processes in job
        -R <rrs>     Specify Resource Requirement String
        -W           Allow wrapping of hosts
        -S           Settle for available hosts
        -j <job id>  Run this job on same resources as <job id>
        -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 "<nname> [<procs>][,...]" Specify Rankmap String
        -m <rfile>   Specify Rankmap File
        -Mf <rmap>   Specify Rankmap File or String
        -u           Use any partition independent nodes
        -t <n>       Multiply daemon and mprun timeouts by factor
                     n; n > 1
```

▼ 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 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 4-4 Options for mprun

Option	Description	Page
–A	Redirect output with an argument vector	Page 41
–B	Redirect stderr and stdout output streams to individual files	Page 40
–C	Run the program using a different working directory	Page 50
–c	Run the job on a different cluster	Page 23
–D	Redirect output to mprun	Page 40
–G	Start the program with a different group name	Page 51
–h	List the options of the mprun command	Page 52
–I	Redirect output with a custom configuration	Page 42
–i	Standard input is sent only to rank 0	Page 42
–J	Display a program's *jid* and number of processes after it finishes executing	Page 53
–j	Run a program on the same nodes as another program	Page 24
-l	Distribute processes among nodes	Page 29
-m	Distribute processes among nodes as specified in a rank map file	Page 31
–n	Read standard input from /dev/null	Page 42
–N	How to shut off all I/O connections	Page 41
–np	Run a program on multiple processes	Page 23
–Ns	Run a program with process spawning disabled	Page 25
–o	Tag each output line with the rank of the process that wrote it.	Page 53
–p	Run the program on a different partition	Page 23
–R	Distribute nodes among processes using a resource requirement spec	Page 33
–S	Settle for the available number of processes	Page 26
–U	Start the program with a different user name	Page 51
–u	Include independent nodes when you distribute processes among the nodes of a partition	Page 27

TABLE 4-4 Options for `mprun` *(Continued)*

Option	Description	Page
`-V`	Display the command's version information	Page 52
`-W`	Wrap multiple processes around available nodes	Page 25
`-Ys`	Execute the program with process spawning enabled	Page 25
`-Z`	Distribute processes among nodes by block	Page 30
`-Zt`	Distribute processes among nodes by block, but force each block to use a different node	Page 30

CHAPTER 5

Killing or Sending Signals to Programs With `mpkill`

What You Can Do

To Perform This Task	Use This Option	Described On
▼ How to kill a running program	`mpkill`	Page 58
▼ How to display a list of signals	`-l -d`	Page 58
▼ How to send a signal to a job	*-signal*	Page 58

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.

▼ 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 59
```

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

Issuing `mpkill` without specifying a signal sends a `SIGTERM` to the job.

Command Reference (`mpkill`)

TABLE 5-1 Options for `mpkill`

Command	Description	Page
none	Stop all processes associated with a particular job	Page 58
`-l`	Display a list of supported signals	Page 58
`–d`	Display a descriptive list of supported signals	Page 58
`-`*signal*	Send a signal to a job	Page 58

CHAPTER 6

Displaying Program Information With `mpps`

What You Can Do

To Perform This Task	Use this Option	Described On
▼ How to display job status	*none*	Page 62
▼ How to display information about individual jobs	`-J`	Page 63
▼ How to display information about all jobs	`-e`	Page 64
▼ How to display a job's start time	`-f`	Page 64
▼ How to display job information by partition	`-A -a`	Page 64
▼ How to display job information by process	`-P -p`	Page 65

▼ 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
JID   NPROC  UID   STATE  AOUT
41    3      slu   RUN    AAA
46    4      slu   EXNG   tmp
49    1      slu   EXIT   tmp
99    9      slu   EXNG   uname
100   9      slu   EXNG   uname
```

The status fields are described in TABLE 6-1.

TABLE 6-1 Job Status Displayed by `mpps`

`mpps` Output	Description
`CORE`	The job or process exited due to a signal and core was dumped.
`CRNG`	The job is exiting due to a signal. The first process to die dumped core.
`EXIT`	The job or process exited normally.
`EXNG`	The job is exiting. At least one process exited normally.
`FAIL`	The job or process failed while starting, or was aborted.
`FLNG`	Initialization of the job failed, or a job abort has been signaled.
`ORPHAN`	The process has been “orphaned,” that is, the node on which it exists has gone offline.
`RUN`	The job or process is running.
`SEXIT`	The job or process exited due to a signal.
`SEXNG`	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.
`STOP`	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 6-2 Job attributes for -J option to mpps

Attribute	Description
part	The name of the partition running the job
jid	The job's unique ID
nproc	The number of processes requested (the actual number of processes started may differ if the -W (Page 25) or -S (Page 26) flags were used with mprun)
uid	The user on whose behalf the job was run (normally the user who submitted the job)
gid	The group on whose behalf the job was run (normally the group of the user who submitted the job)
state	BUILD - The job is being submitted WAIT - The job is waiting to run SPAWN - The job is preparing to run RUN - The job is running RSTRT - The job has been killed because one of the nodes on which it was running went down; the job will be restarted
running	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
wkdir	The directory in which the job's processes start
aout	The name of the program
paout	The full path of the program
ctime	The time when mprun was invoked
args	The command-line arguments of the program
stime	The time the job was started
prio	The job priority (higher numbers run first)

▼ 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
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 6-3, 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 6-3, below.

TABLE 6-3 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
iod	The process ID of the I/O daemon for this process
load	The load on the node executing the process
node	The name of the node executing the process

Command Reference (mpps)

TABLE 6-4 Options for mpps

Option	Description	Page
none	Display status information about your jobs running in the default partition	Page 62
-J	Display information about a particular job	Page 63
-e	Display information about all jobs	Page 64
-f	Display the time a job started	Page 64
-A	Display information about jobs running in all partitions	Page 64
-a	Display information about jobs running in a particular partition	Page 64
-P	Display process information about a job	Page 65
-p	Display information about a particular process attribute	Page 65

CHAPTER 7

Displaying Information With `mpinfo`

What You Can Do

Displaying Cluster Information

Task	Option	Page
▼ How to display information about any cluster	`-c`	Page 68
▼ How to display information about the current cluster	`-C`	Page 69

▼ 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 7-1 on page 76.

Include the -A option to restrict the output to a particular attribute or set of attributes. (See Page 75.)

▼ 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 7-1 on page 76.

Include the -A option to restrict the output to a particular attribute or set of attributes. (See Page 75.)

Displaying Partition Information

Task	Option	Page
▼ How to display information about individual partitions	`-p`	Page 70
▼ How to display information about all partitions	`-P`	Page 71

▼ 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 7-1 on page 76.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See Page 75.)

▼ 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 7-1 on page 76.

Include the -A option to restrict the output to a particular attribute or set of attributes. (See Page 75.)

Displaying Node Information

Task	Option	Page
▼ How to display information about individual nodes	`-n`	Page 72
▼ How to display information about all nodes	`-N`	Page 73

▼ 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 LOAD1
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 7-1 on page 76.

Include the `-A` option to restrict the output to a particular attribute or set of attributes. (See Page 75.)

▼ 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 LOAD1
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 7-1 on page 76.

Include the -A option to restrict the output to a particular attribute or set of attributes. (See Page 75.)

Displaying Individual Attribute Values

Task	Option	Page
▼ How to display an online list of valid attributes	`-lc -lp -ln`	Page 74
▼ How to restrict output to individual attributes	`-A`	Page 75
▼ How to display information in verbose mode	`-v`	Page 77

▼ 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 7-1 on page 76.

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 7-1 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).
	cpu_user	Percent of time CPU spends running user's program (USER).
	domain	DNS domain.
	enabled	If set, node is available for spawning jobs on it.
	load1	Load average for the past minute (LOAD1).
	load5	Load average for the past five minutes (LOAD5).
	load15	Load average for the past 15 minutes (LOAD15).
	manufacturer	Hardware manufacturer (MANUFACTURER).
	mem_free	Node's available RAM (in Mbytes) (FMEM).
	mem_total	Node's total physical memory (in Mbytes) (MEM).
	name	Name of the node (NAME).
	ncpus	Number of CPU modules in the node (NCPU).
	os_arch_kernel	Node's kernel architecture (MACH).

TABLE 7-1 Attributes Displayed by `-A` option to `mpinfo` *(Continued)*

Object	Attribute	Description
Node (*continued*)	`os_max_proc`	Maximum number of processes allowed on the node (note that this is *all* processes, including cluster daemons) (`MPROC`).
	`os_name`	Name of the operating system running on the node (`OS`).
	`os_release`	Operating system's release number (`OSREL`).
	`os_release_maj`	The major number of the operating system release number (`MAJ`).
	`os_release_min`	The minor number of the operating system release number (`MIN`).
	`os_version`	Operating system's version (`OSVER`).
	`partition`	The partition of which the node is a member (`PARTITION`).
	`serial_number`	Hardware serial number (`SERIAL`).
	`swap_free`	Node's available swap space (in Mbytes) (`FSWP`).
	`swap_total`	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 7-2 Options for `mpinfo`

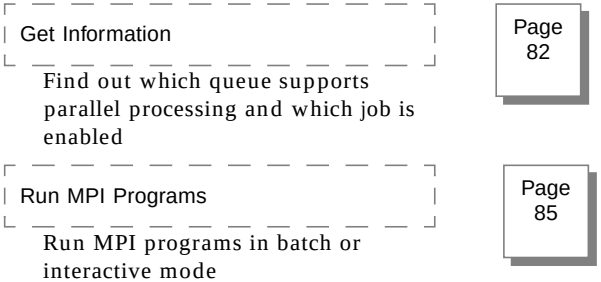
Command	Description	Page
`-c`	Display cluster-level, partition, or node information about any cluster	Page 68
`-C`	Display cluster-level information about the current cluster; equivalent to `mpinfo -c` *cluster-name* `-C`	Page 69
`-p`	Display information about individual partitions	Page 70
`-P`	Displays information about all partitions in the cluster	Page 71
`-n`	Displays information about individual nodes	Page 72
`-N`	Displays information about all nodes in the cluster	Page 73
`-lc`	List the cluster attributes that can be displayed by `-A`	Page 74
`-lp`	List the partition attributes that can be displayed by `-A`	Page 74
`-ln`	List the node attributes that can be displayed by `-A`	Page 74
`-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`	Page 75
`-v`	Display information in verbose mode—must combine with one of these options: `-c -C -p -P -n -N`	Page 77

APPENDIX **A**

Executing Programs With LSF Suite's bsub

This appendix highlights a subset of the LSF Suite's commands. For a complete description, see the LSF documentation.

What You Can Do



Getting Information with bqueues

Task	Option	Page
▼ How to Find Out Which Queues Support Parallel Jobs	`bqueues -l`	Page 82
▼ How to Find Out Which Mode is Enabled	`bqueues -l`	Page 83

▼ How to Find Out Which Queues Support Parallel Jobs (`bqueues -l`)

You can only submit distributed MPI jobs to batch queues that have been configured to handle parallel jobs. Batch queues are configured by the cluster administrator, but you can use the `bqueues -l` command to find out which of those queues support parallel jobs:

```
% bqueues -l
```

The `bqueues –l` output contains status information about all the currently defined queues. For example:

```
QUEUE: hpc
  -- Sun HPC interactive queue (uses pam as a job starter. This
     is the default queue.

           :
           :

SCHEDULING POLICIES:  NO_INTERACTIVE

USERS: all users
HOSTS: all hosts used by the LSF Batch system
JOB_STARTER: pam
PREEMPTION: PREEMPTIVE
```

Look for a queue that includes this line:

```
JOB_STARTER: pam
```

That line indicates that the queue is configured to handle parallel jobs.

Note – The `pam` entry may be followed by a `–t` or `–v`. The `–t` option suppresses printing of process status upon completion and `–v` specifies that the job is to run in verbose mode.

If no queues are currently configured for parallel job support, ask the cluster administrator to configure one.

▼ How to Find Out Which Mode Is Enabled (`bqueues -l`)

To find out whether batch or interactive mode is enabled, use the `bqueues -l` command:

```
% bqueues -l queue-name
```

The `bqueues –l` output contains status information about all the currently defined queues. For example:

```
QUEUE: hpc
  -- Sun HPC interactive queue (uses pam as a job starter). This
     is the default queue.

           :
           :

SCHEDULING POLICIES:  NO_INTERACTIVE

USERS: all users
HOSTS: all hosts used by the LSF Batch system
JOB_STARTER: pam
PREEMPTION: PREEMPTIVE
```

Examine the SCHEDULING POLICIES section of the output and look for the following entries:

Entry	Description
ONLY_INTERACTIVE	Batch mode is disabled.
NO_INTERACTIVE	Batch mode is enabled
no INTERACTIVE entry	(default) All interactive and batch modes are enabled

Running Programs Sith bsub

Task	bsub Option	Described On
▼ How to run an MPI program in batch mode	*none*	Page 85
▼ How to specify a particular job queue	`-q`	Page 86
▼ How to run an interactive MPI program	`-I`	Page 86
▼ How to force process wrapping	`-n`	Page 87
▼ How to redirect error output	`-sunhpc -e`	Page 88
▼ How to redirect standard output	`-sunhpc -o`	Page 88
▼ How to tag output by rank	`-sunhpc -t`	Page 89
▼ How to run as multiple processes	`-sunhpc -n`	Page 90
▼ How to colocate jobs	`-sunhpc -j` `-sunhpc -J`	Page 91
▼ How to restart a job	`-sunhpc -s`	Page 91

▼ How to Run an MPI Program in Batch Mode

Note – To use LSF batch commands, your PATH variable must include the directory where the LSF Base, Batch, and Parallel components were installed.

By default, the LSF Suite executes MPI programs in batch mode. However, batch mode can be disabled by the cluster administrator. To make sure batch mode is enabled, see "How to Find Out Which Mode Is Enabled (bqueues -l)" on page 83.)

To run an MPI program in the LSF Suite, you must first know the name of a queue that is configured to handle parallel jobs. (See "How to Find Out Which Queues Support Parallel Jobs (bqueues -l)" on page 82.)

Once you know which queue to use and you have verified that batch mode is enabled, use the bsub command:

```
% bsub [options] program-name
```

Here is a simple example:

```
% bsub hpc-job
```

The example above assumes the default queue is configured for parallel processing. If it is not, use the `-q` option to specify a different queue, as described below.

▼ How to Specify a Particular Job Queue (`-q`)

If the queue that supports parallel processing is not the default queue, use the `-q` option to the `bsub` command:

```
% bsub -q queue-name [options] program-name
```

For example, the following command submits `hpc-job` to the queue named `hpc-queue`:

```
% bsub -q hpc-queue hpc-job
```

▼ How to Run an Interactive MPI Program (`-I`)

Interactive batch mode makes full use of the LSF Batch system's job scheduling policies and host selection facilities, but keeps the job attached to the terminal session that submitted it. This mode is well suited to Sun MPI jobs and other resource-intensive applications.

To run an MPI program in LSF interactive mode, you must first know the name of a queue that is configured to handle parallel jobs. (See "How to Find Out Which Queues Support Parallel Jobs (bqueues -l)" on page 82.)

You must also verify that interactive mode is enabled. (See "How to Find Out Which Mode Is Enabled (bqueues -l)" on page 83.)

Once you know which queue to use and you have verified that interactive mode is enabled, use the `bsub` command. If the queue you need is not the default queue, use the `-q` option, as well:

```
% bsub -I [-q queue-name] [options] program-name
```

Here's an example:

```
% bsub -I -q hpc-queue -n 4 hpc-job
```

In the example above, the `hpc` is the queue name, the program runs as four processes, and its name is `hpc-job`.

When the queue accepts the job, it returns a job ID. You can use the job ID later as an argument to various commands that enquire about job status or that control certain aspects of job state. For example, you can suspend a job or remove it from a queue with the `bstop` *jobid* and `bkill` *jobid* commands. These commands are described in the *LSF Batch User's Guide*.

▼ How to Force Process Wrapping (`-n`)

Note – This option is being deprecated; it will not be available after this release.

The `bsub` command provides the `-n` option to specify the number of CPUs to be used for a program. If you use the `-sunhpc -n` option to run a program as multiple processes along with the `bsub -n` option, you can force process wrapping:

```
% bsub -n cpu-count [-q queue] -sunhpc -n process-count program-name
```

You can specify *cpu-count* as either an integer or a range:

integer --> *cpu-count*
range --> *min-cpu-count*`,`*max-cpu-count*

In either case, as a side effect of entering a *cpu-count*, the LSF Suite allocate any extra processes among the available CPUs, assigning as many processes to a CPU as required to complete the job.

▼ How to Redirect Error Output (-sunhpc -e)

To redirect `stderr` to a file, use the `-sunhpc -e` options to the `bsub` command, and specify the filename:

```
% bsub -I [-n processes] [-q queue] -sunhpc -e filename program-name
```

The error output from each process is redirected to a file with this nomenclature:

filename`.R`*n*

You supply the *filename* in the command, and LSF adds the `.R`*n*. The *n* indicates the rank of the process producing the error output. For example:

```
% bsub -I -n 4 -q hpc-queue -sunhpc -e myerrorfile hpc-job
```

The command above would run the `hpc-job` program as four processes on the `hpc-queue` queue, and redirect error output to four files:

```
myerrorfile.R0
myerrorfile.R1
myerrorfile.R2
myerrorfile.R3
```

▼ How to Redirect Standard Output (-sunhpc -o)

To redirect `stdout` to a file, use the `-sunhpc -o` options to the `bsub` command, and specify the filename:

```
% bsub -I [-n processes] [-q queue] -sunhpc -o filename program-name
```

The error output from each process is redirected to a file with this nomenclature:

filename`.R`*n*

You supply the *filename* in the command, and LSF adds the `.R`*n*. The *n* indicates the rank of the process producing the error output. For example:

```
% bsub -I -n 4 -q hpc-queue -sunhpc -o myoutputfile hpc-job
```

The command above would run the `hpc-job` program as four processes on the `hpc-queue` queue, and redirect error output to four files:

```
myerrorfile.R0
myerrorfile.R1
myerrorfile.R2
myerrorfile.R3
```

▼ How to Tag Output by Rank (`-sunhpc -t`)

Note – You cannot use the `–t` argument with the `–e` or `–o` options to `–sunhpc`.

To identify output by the rank of the node that produced it, use the `-sunhpc -t` options:

```
% bsub -I -n processes [-q queue] -sunhpc -t program-name
```

For example:

```
% bsub -I -n 4 -q hpcqueue -sunhpc -t myprogram
.
.
.
R0: SunOS hpc-demo3 5.8 s998_16 sun4u sparc SUNW, Ultra-2
R1: SunOS hpc-demo3 5.8 s998_16 sun4u sparc SUNW, Ultra-2
R2: SunOS hpc-demo3 5.8 s998_16 sun4u sparc SUNW, Ultra-2
R3: SunOS hpc-demo3 5.8 s998_16 sun4u sparc SUNW, Ultra-2
```

▼ How to Run As Multiple Processes (-sunhpc -n)

By default an MPI program runs as one process. To run the program as multiple processes, use the `-sunhpc -n` option:

```
% bsub [-I] [-q queue] -sunhpc -n number-of-processes program-name
```

The LSF Suite attempts to allocate each process to a CPU. If not enough CPUs are available, the job fails. (You can force process wrapping by using the `-n` option to the `bsub` command. See "How to Force Process Wrapping (-n)" on page 87.)

Here is an example:

```
% bsub -q hpc-queue -sunhpc -n 8 hpc-job
```

The example attempts to allocate eight processes among the available number of CPUs. If eight CPUs are not available, the job fails. Here is the same example with the `-n` option to the `bsub` command:

```
% bsub -n 4 -q hpc-queue -sunhpc -n 8 hpc-job
```

▼ How to Colocate Jobs (-sunhpc -j -J)

Note – This feature is deprecated; it will be removed after this release.

To colocate a job with another, use the `-sunhpc -j` or `-J` options to the `bsub` command:

```
% bsub -I [-q queue] -sunhpc -j jid program-name

% bsub -I [-q queue] -sunhpc -J job-name program-name
```

Use the `-j` option to identify the job by its *jid*. Use the `-J` option to identify the job by name. To get a job's *jid*, use the `bjobs` command, described in the *LSF Batch User's Guide*. Here is an example:

```
% bsub -I -q hpc-queue -sunhpc -j 4622 hpc-job

% bsub -I -q hpc-queue -sunhpc -J job1 hpc-job
```

The example above colocates the job being started (`hpc-job`) with the job whose jid is `4622`. The second example does the same, but identifies the job by name (`job1`)

▼ How to Restart a Job (-sunhpc -s)

To restart a job that is in the `STOPPED` state, use the `-s` option (lower case):

```
% bsub [-q queue] -sunhpc -s program-name
```

Note – Do not use the `–s` argument with the Prism debugger. It would add nothing to the Prism capabilities and is likely to interfere with the debugger's control over the debugging session.

To identify processes in the STOPPED state, use the ps command with the –el argument and look for the value under the “S” (for “state”) column. For example:

```
% ps -el
F   S  UID PID PPID C PRI NI  ADDR     SZ WCHAN TTY TIME  CMD
19  T  0   0   0    0 0   SY  f0274e38 0  ?         0:00  sched
```

When you spawn a process that was in the STOPPED state, the program’s name does not appear in the ps output. Instead, the stopped process is identified as a RES daemon.

Command Reference

TABLE 7-3 Options for `bqueues` Command

Option	Description	Page
`-l`	Find out which queues support parallel jobs	Page 82
`-l`	Find out which mode is enabled	Page 83

TABLE 7-4 Options for `bsub` Command

Option	Description	Page
none	Run a program in batch mode	Page 85
`-I`	Run a program in interactive mode	Page 86
`-n`	Force process wrapping	Page 87
`-q`	Run the job on a particular job queue	Page 86
`-sunhpc -e`	Redirect error output	Page 88
`-sunhpc -j`	Colocate jobs and identify other job by its jid	Page 91
`-sunhpc -J`	Colocate jobs and identify other job by its name	Page 91
`-sunhpc -n`	Run the job as multiple processes	Page 90
`-sunhpc -o`	Redirect standard output	Page 88
`-sunhpc -s`	Restart a job	Page 91
`-sunhpc -t`	Tag output by rank	Page 89

APPENDIX **B**

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 B-1 on page 97 (for non-I/O MPI) and TABLE B-2 on page 99 (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:

`[`*x y z*`] Error in` *function_name*`:` *errclass_string*`:`*intern*`(`*a*`):`*description*`:` *unixerrstring*

where

- `[`*x y z*`]` 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:

`[`*x y z*`] 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 B-1 Sun MPI Standard Error Classes

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

TABLE B-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_TIMEDOUT	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 exhauste.
MPI_ERR_LASTCODE	55	Last error code.

MPI I/O message are listed separately, in TABLE B-2 on page 99.

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 B-2 on page 99. 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 B-2 Sun MPI I/O Error Classes

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

TABLE B-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 46 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

A

B

C

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W