Sun HPC ClusterTools™ 3.1 User's Guide

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300   Fax 650 969-9131

Part No. 806-3733-10
March 2000, Revision A

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

Please
Recycle

Adobe PostScript™

Contents

Preface vii

1. **Introduction 1**

   Sun HPC ClusterTools 3.1 Software 1

   Sun Compilers 4

   Solaris Operating Environment 5

   Fundamental CRE Concepts 5

   Using the Sun CRE 8

2. **Logging In 9**

   Logging In 9

   After Logging In 9

   Logging Out 11

3. **Executing Programs 13**

   Choosing Where to Execute 13

   Authentication Methods 14

   Specifying Default Execution Options 14

   Executing Programs via `mprun` 15

   Network Communication 18

   Executing Programs 18

`mprun` Options 19

Specifying Where a Program Is to Run 22

Mapping MPI Ranks to Nodes 26

Expressing More Complex Resource Requirements With `-R` 29

Specifying the Behavior of I/O Streams 36

Changing the Working Directory 43

Executing With a Different User or Group Name 43

Getting Information 43

Specifying a Different Argument Vector 44

Exit Status 44

Omitting `mprun` 44

Sending a Signal to a Process 45

4. Getting Information 47

`mpps`: Finding Out Job Status 47

`mpinfo`: Configuration and Status 50

5. Debugging Programs 57

Debugging Sun MPI Programs 57

A. Using LSF With Sun HPC ClusterTools 59

About LSF Suite 3.2.3 59

Job Execution Modes 60

Starting Sun MPI Programs 60

B. Troubleshooting 69

MPI Messages 69

MPI I/O Error Handling 72

Exceeding the File Descriptor Limit 74

Exceeding the TCP Port Limit 75

Tables

TABLE 3-1 User Commands Required by Authentication Methods 14

TABLE 3-2 Options for `mprun` 19

TABLE 3-3 Standard RRS Attributes 30

TABLE 3-4 Operators Valid for Use in RRS 32

TABLE 3-5 `mprun` Shortcut Summary 42

TABLE 4-1 Job and Process States 48

TABLE 4-2 Partition Attributes Available via `mpinfo` 52

TABLE 4-3 Node Attributes Available via `mpinfo` 53

TABLE B-1 Sun MPI Standard Error Classes 71

TABLE B-2 Sun MPI I/O Error Classes 73

Preface

This manual describes how to execute Sun™ MPI jobs on systems running Sun HPC ClusterTools™ 3.1 software with the Sun HPC Cluster Runtime Environment (CRE).

Appendix A provides supplemental information for those using the LSF 3.2.3 workload management suite from Platform Computing Corporation instead of the CRE. This appendix supplements the documentation that accompanies LSF.

Before You Read This Book

Product notes for the other components in the suite are included in *Sun HPC ClusterTools Product Notes*. For information about writing MPI programs, refer to the *Sun MPI Programming and Reference Guide*.

If you are using LSF instead of the CRE, see the documentation that came withthe LSF Suite, especially the *LSF Batch User's Guide*.

Using UNIX® Commands

This document may not contain information on basic UNIX® commands and procedures such as creating directories and copying and deleting files.

See one or more of the following for this information:

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

Typographic Conventions

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	*machine_name*%
C shell superuser	*machine_name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Related Documentation

TABLE P-3 Related Documentation

Application	Title	Part Number
Sun HPC ClusterTools software	*Sun HPC ClusterTools Product Notes*	806-4182-10
Sun MPI programming	*Sun MPI Programming and Reference Guide*	806-3734-10
Scalable Coherent Interface (SCI)	*Sun HPC SCI Guide*	806-4183-10
Sun HPC ClusterTools performance tuning	*Sun HPC ClusterTools Performance Guide*	806-3732-10
Sun HPC ClusterTools administration	*Sun HPC ClusterTools Administrator's Guide*	806-3731-10
Prism™ development environment	*Prism User's Guide*	806-3736-10
Prism development environment	*Prism Reference Manual*	806-3737-10
Sun S3L	*Sun S3L Programming and Reference Guide*	806-3735-10

In addition, if you are using Platform Computing's Load Sharing Facility (LSF) Suite, you will want to consult Platform's documentation for LSF.

LSF Suite	*LSF Batch User's Guide*
LSF Suite	*LSF Parallel User's Guide*
LSF Suite	*LSF Batch Programmer's Guide*
LSF Suite	*LSF Batch User's Quick Reference*

These documents are available from Platform Computing Corporation. See their web site for more information:

`http://www.platform.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://www1.fatbrain.com/documentation/sun`

Accessing Sun Documentation Online

The `docs.sun.com`SM web site enables you to access Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at:

`http://docs.sun.com`

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

`docfeedback@sun.com`

Please include the part number (806-3733-10) of your document in the subject line of your email.

CHAPTER 1

Introduction

This manual explains how to execute Sun MPI applications on a Sun HPC cluster that is using the Sun Cluster Runtime Environment (CRE) for job management. If you are using the LSF Suite from Platform Computing instead, see Appendix A.

Sun HPC ClusterTools 3.1 Software

This section describes Sun HPC ClusterTools 3.1 software, an integrated ensemble of parallel development tools that extend Sun's network computing solutions to high-end distributed-memory applications. Sun HPC ClusterTools products can be used either with the CRE or with LSF Suite 3.2.3, Platform Computing Corporation's resource-management software.

Sun Cluster Runtime Environment

The CRE is a cluster administration and job launching facility. It provides users with an interactive command-line interface for executing jobs on the cluster and for obtaining information about job activity and cluster resources.

The CRE also performs load-balancing for programs running in shared partitions.

Partitions, load-balancing, and other related Sun HPC cluster concepts are discussed in "Fundamental CRE Concepts" on page 5.

Sun MPI and MPI I/O

Sun MPI is a highly optimized version of the Message Passing Interface (MPI) communications library. Sun MPI implements all of the MPI 1.2 standard as well as a significant subset of the MPI 2.0 feature list. For example, Sun MPI provides the following features:

- Integration with the Sun Cluster Runtime Environment (CRE).
- Integration with 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 a Scalable Coherent Interface (SCI) network, can be run without change on a cluster that has an ATM network.
- Multiprotocol support such that MPI picks the fastest available medium for each type of connection (such as shared memory, SCI, or ATM).
- Communication via shared memory for fast performance on clusters of SMPs.
- Finely tunable shared memory communication.
- Optimized collectives for symmetric multiprocessors (SMPs) and clusters of SMPs.
- MPI I/O support for parallel file I/O.
- *Prism support* – Users can develop, run, and debug programs in the Prism programming environment.
- *Implicit coscheduling* – The Sun HPC spind daemon enables certain processes of a given MPI job on a shared-memory system to be scheduled at approximately the same time as other related processes. This coscheduling reduces the load on the processors, thus reducing the effect that MPI jobs have on each other.
- Limited support of one-sided communication routines.
- Sun MPI is a dynamic library.
- MPI-2 dynamic support.

Sun MPI and MPI I/O provide full F77, C, and C^{++} support, as well as Basic F90 support.

Sun Parallel File System

The Sun Parallel File System (PFS) component of the Sun HPC ClusterTools suite of software 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 means the time required to read or write a PFS file can be reduced by an amount roughly proportional to the number of file server nodes in the PFS file system.

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

Prism

Prism is the Sun HPC graphical programming environment. It allows you to develop, execute, debug, and visualize data in multithreaded or nonthreaded message-passing programs. With Prism you can

- Control various aspects of program execution, such as:
    - Starting and stopping execution
    - Setting breakpoints and traces
    - Printing values of variables and expressions
    - Displaying 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*

Prism can be used 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 that are used widely 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.
- Autocorrelation.
- Convolution/deconvolution.

- Tridiagonal solvers.
- Banded solvers.
- Eigensolvers.
- Singular value decomposition.
- Least squares.
- One-dimensional sort.
- Multidimensional sort.
- Selected ScaLAPACK and BLACS application program interface.
- 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 provided online.

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

Sun Compilers

Sun HPC ClusterTools 3.1 software supports the following Sun compilers:

- Sun WorkShop Compilers C/C^{++} v4.2 and v5.0
- Sun WorkShop Compilers Fortran v4.2 and v5.0

Solaris Operating Environment

Sun HPC ClusterTools software uses the Solaris 2.6, or Solaris 7 or 8 (32-bit or 64-bit) operating environment. All programs that execute under Solaris 2.6, Solaris 7, or Solaris 8 execute in the Sun HPC ClusterTools environment.

Fundamental CRE Concepts

This section introduces some important concepts that you should understand in order to use the CRE.

Cluster of Nodes

As its name implies, the Sun Cluster Runtime Environment is intended to operate in a Sun HPC cluster—that is, in a collection of Sun symmetric multiprocessor (SMP) servers that are interconnected by any Sun-supported, TCP/IP-capable interconnect. An SMP attached to the cluster network is referred to as a *node*.

The CRE manages the launching and execution of both serial and parallel jobs on the cluster nodes, which are grouped into logical sets called *partitions*. (See the next section for more information about partitions.) For serial jobs, its chief contribution is to perform load-balancing in shared partitions, where multiple processes can be competing for the same node resources. For parallel jobs, the CRE provides:

- A single job-monitoring and control point
- Load-balancing for shared partitions
- Information about node connectivity
- Support for spawning of MPI processes
- Support for Prism interaction with parallel jobs

Note – A “cluster” can consist of a single Sun SMP server. However, executing MPI jobs on even a single-node cluster requires the CRE to be running on that cluster.

The CRE supports parallel jobs running on clusters of up to 64 nodes containing up to 1024 CPUs.

Partitions

The system administrator configures the nodes in a Sun HPC cluster into one or more logical sets, called *partitions.* A job is always launched on a predefined partition that is currently *enabled*, or accepting jobs. A job will run on one or more nodes in that partition, but not on nodes in any other enabled partition.

Note – The CPUs in a Sun Enterprise 10000 server can be configured into logical "nodes." These domains can be logically grouped to form partitions, which the CRE uses in the same way it deals with partitions containing other types of Sun HPC nodes.

Partitioning a cluster allows multiple jobs to execute concurrently, without the risk that jobs on different partitions will interfere with each other. This ability to isolate jobs can be beneficial in various ways. For example:

- If one job requires exclusive use of a set of nodes but other jobs need to execute at the same time, the availability of two partitions in a cluster would allow both needs to be satisfied.
- If a cluster contains a mix of nodes whose characteristics differ—such as having different memory sizes, CPU counts, or levels of I/O support—the nodes can be grouped into partitions that have similar resources. This would allow jobs that require particular resources to be run on suitable partitions, while jobs that are less resource-dependent could be relegated to less specialized partitions.

The system administrator can selectively *enable* and *disable* partitions. Jobs can be executed only on enabled partitions. This restriction makes it possible to define many partitions in a cluster, but have only a few active at any one time.

At any given time, a node can be in only one enabled partition. In other words, a node can belong to more than one partition, so long as only one of its partitions is enabled at a time.

It is also possible for a node in the cluster to be *independent*, that is, not belonging to any currently enabled partition. If you log in to one of these independent nodes and do not request a particular partition, the CRE will launch your job on the cluster's *default partition*. See "Specifying the Partition" on page 23 for more information.

Note – Although a job cannot be run across partition boundaries, it can be run on a partition plus independent nodes. (See "When Number of Processes Exceeds Number of CPUs" on page 25 for information.)

In addition to enabling and disabling partitions, the system administrator can set and unset other partition attributes that influence various aspects of how the partition functions. See "Displaying Partition Information" on page 51 for information about determining which values have been set on your local partitions.

Load-Balancing

The CRE load-balances programs when more CPUs are available than are required for a job. When you issue the `mprun` command to launch a job, the 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, the 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 won't sit idle while other nodes are heavily loaded, thereby keeping overall throughput of the partition as high as possible.

Jobs and Processes

When a serial program executes on a Sun HPC cluster, it becomes a Solaris process with a Solaris *process ID*, or *pid*.

When the CRE executes a distributed message-passing program it spawns multiple Solaris processes, each with its own pid.

The CRE also assigns a *job ID*, or *jid*, to the program. If it is an MPI job, the jid applies to the overall job. Job IDs always begin with a `j` to distinguish them from pids. Many CRE commands take jids as arguments. For example, you can issue an `mpkill` command with a signal number or name and a jid argument to send the specified signal to all processes that make up the job specified by the jid.

Parallel File System

From the user's perspective, PFS file systems closely resemble UNIX file systems. 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.

Using the Sun CRE

The balance of this manual discusses the following aspects of using the CRE:

- Logging in – see Chapter 2.
- Choosing a partition and executing programs – see Chapter 3.
- Obtaining information – see Chapter 4.
- Debugging programs – see Chapter 5.

Appendix A contains information about using LSF instead of the CRE, and Appendix B contains information about MPI error messages.

CHAPTER 2

Logging In

To start using the Sun CRE, you will log in to a Sun HPC cluster. Once logged in, you can write or edit programs, compile and link them, and run and degug them in a partition. When you have finished you will log out.

Logging In

Logging in to a node in a Sun HPC cluster is the same as logging in to any Sun server. That is, to log in on your local machine, just supply your user name at the login prompt and, if a password is required, the password. For remote logins, use `rlogin`.

You receive the standard Solaris login information, followed by a Solaris prompt:

```
Sun Microsystems Inc.   SunOS 5.6       Generic March 2000
Users: wmitty jthurb
node0%
```

You are now logged in to a node whose hostname is `node0`.

After Logging In

Once you are logged in to a Sun HPC cluster, you can issue any Solaris or Sun HPC commands, and you can execute any programs that will execute under Solaris 2.6, Solaris 7, or Solaris 8 operating environment.

Writing Programs

You can perform program development on a Sun HPC cluster node or you can do it on any computer running a compatible version of the Solaris operating environment.

Compiling and Linking Programs

If your program uses Sun HPC ClusterTools components, you must compile and link your program on a cluster that contains the 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 and Reference 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.

Issuing CRE Commands

The CRE provides commands that allow you to execute programs in partitions and obtain information about cluster resources and job activity.

- `mprun` – Executes your job according to your specified requirements. See Chapter 3 for information.
- `mpkill` – Sends a standard Solaris signal to a specified job. See “Sending a Signal to a Process” on page 45 for information.
- `mpinfo` – Displays information about the configuration of partitions and nodes, as well as node status information. See “`mpinfo`: Configuration and Status” on page 50 for information.
- `mpps` – Displays information about jobs and processes currently running on the cluster. See “`mpps`: Finding Out Job Status” on page 47 for information.

The 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.

CRE commands take options that consist of a dash followed by one or two letters. You can combine single-letter options that don’t take arguments so long as they don’t create ambiguity with multiletter options. For example, the command

```
% mprun –B –J
```

can also be written as

```
% mprun –BJ
```

Man Pages

The man pages for Sun HPC commands reside in

```
/opt/SUNWhpc/man
```

If you cannot display these man pages, you may need to add this directory to your manpath.

Logging Out

To log out of the Sun HPC cluster, issue the command

```
% logout
```

CHAPTER 3

Executing Programs

This chapter describes how to execute programs on a Sun HPC cluster. You can execute programs in any partitions to which you have access. A major difference between the Sun HPC cluster and a collection of workstations is that the Sun Cluster Runtime Environment (CRE) provides you with a simple, interactive interface for specifying where and how your program should run.

Programs written for Solaris 2.6 , Solaris 7, or Solaris 8 can run without recompilation on a Sun HPC cluster.

Note – Running parallel jobs with the CRE is supported on up to 1024 processors and up to 64 nodes.

Choosing Where to Execute

The Sun CRE provides you with considerable flexibility in choosing where you want your program to execute. For example, you can specify

- The partition in which you want to execute your program. (See "Partitions" on page 6 for introductory information.)
- The number of processes you want to start, and how you want to map them to the nodes in the partition.
- The characteristics of the node or nodes on which you want to run—for example, the minimum amount of memory required or the maximum acceptable load.

Specific instructions about specifying where a program is to run begin in the section "Specifying Where a Program Is to Run" on page 22.

You can specify default execution criteria via the MPRUN_FLAGS environment variable; see “Specifying Default Execution Options” on page 14. You can also override these criteria via options to the mprun command.

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 command beginning with mp, such as mprun or mpps.
DES	While DES authentication is in use, you must issue the keylogin command before issuing any commands beginning with mp, such as mprun or mpps.

See your system administrator for details.

Specifying Default Execution Options

You can use the environment variable MPRUN_FLAGS to specify one or more default options to the program execution command, mprun. (These options are described in more detail in the remainder of the chapter and are listed in “mprun Options” on page 19.) Then, you need not specify any option contained in MPRUN_FLAGS. The mprun command will be interpreted as if the options contained in MPRUN_FLAGS were included on the command line (preceding any options that are on the command line). You can override any default option by including a new value for the option on the mprun command line.

The setting of the environment variable can be any number of valid mprun options. If you use more than one word, enclose the list in quotation marks.

For example, the following makes part2 the default partition to be used for mprun.

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.

C shell

```
% printenv MPRUN_FLAGS
```

Bourne shell

```
# printenv MPRUN_FLAGS
```

Note – Your system administrator may have set defaults for the Sun HPC cluster by defining environment variables or CRE attributes, such as SUNHPC_PART or default_interactive_partition. The defaults that you set in MPRUN_FLAGS override defaults set by the system administrator.

Executing Programs via mprun

This section provides general information about executing programs via mprun.

Execution via mprun is similar to standard Solaris program execution. For example,

- Your environment is used as if you executed the program from a traditional shell.
- Signals are treated as they are in standard Solaris; for multiprocess programs, if one process is killed via a signal, all processes are killed.
- You can run a program in the background.

CRE commands do differ slightly from standard Solaris execution. The remainder of this section discuses these differences.

Moving mprun Processes to the Background

When you move either a process started with mprun or a script that issues mprun commands to the background, you must do one of the following:

- Redirect stdin to a file. For example:

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

- Specify the `–n` option to `mprun` so that standard in will be read from `/dev/null`. See "Specifying the Behavior of I/O Streams" on page 36 for a detailed discussion of standard I/O issues.

  ```
  % mprun –n
  ```

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

Shell-Specific Actions

If you want to perform actions that are shell specific, such as executing compound commands, you must first invoke the appropriate shell as part of the `mprun` command. For example,

```
% mprun csh –c 'echo $USER'
```

or

```
% mprun csh –c 'cd /foo ; bar'
```

File Descriptors

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

Note – The 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 7 or Solaris 8 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 Solaris 7 or Solaris 8 32-bit 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.

See “Exceeding the File Descriptor Limit” on page 74 for additional information.

Core Files

Core files are produced as they normally are in Solaris. However, if more than one process dumps core in a multiprocess program, the resulting core file may be invalid.

Standard Output and Standard Error

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 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. You can also specify other methods for handling I/O streams, including the three standard ones. See “Specifying the Behavior of I/O Streams” on page 36 for additional information.

SMP Characteristics of Sun HPC Clusters

Since your Sun HPC cluster consists of symmetric multiprocessors (SMPs), the CRE takes into consideration the number of CPUs per node by default. In general, mprun will assign more processes to larger SMPs. For information about how the CRE allocates processes to CPUs, see “When Number of Processes Exceeds Number of CPUs” on page 25 and “Option Priority Matrix” on page 21.

Network Communication

For clusters of up to four nodes, the preferred interconnect technology is the Scalable Coherent Interface (SCI), which employs the low-latency Remote Shared Memory (RSM) protocol. The Sun HPC ClusterTools software also runs on clusters of up to 64 nodes when connected via any non-SCI TCP/IP capable interconnect, such as Ethernet, high-speed Ethernet, Gigabit Ethernet, ATM OC-3, ATM OC-12, FDDI, and HiPPI.

A non-SCI TCP/IP interconnect is always used to carry standard LAN traffic for the cluster. If an SCI network is not available, a TCP/IP network can also be used for message-passing by parallel Sun MPI applications.

However, 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 75 for additional information.

Executing Programs

The basic format of the `mprun` command is

```
% mprun [options] [–] executable [args ... ]
```

Note – When the name of your program conflicts with the name of an `mprun` option, use the **–** (dash) symbol to separate the program name from the option list.

mprun Options

The following table lists and briefly describes the mprun options. Their use is described more fully in the balance of this chapter. See the “Option Priority Matrix” on page 21 for information about how the options interact when conflicting options are specified.

TABLE 3-2 Options for mprun

Option	Meaning
–A *aout*	Execute aout and use a different argument as the argv[0] argument to the program. See “Specifying a Different Argument Vector” on page 44.
–B	Send stderr and stdout output streams to files. See “Specifying the Behavior of I/O Streams” on page 36.
–C *path*	Use the specified directory as the current working directory for the job. See “Changing the Working Directory” on page 43.
–c *cluster_name*	Run on the specified cluster. See “Specifying the Cluster” on page 23.
–D	Provide separate stdout and stderr streams. See “Specifying the Behavior of I/O Streams” on page 36.
–G *group*	Execute with the specified group ID or group name. See “Executing With a Different User or Group Name” on page 43.
–h	Display help. See “Getting Information” on page 43.
–I *file_descr_string*	Use the specified I/O file descriptor string to control I/O stream handling. See “Specifying the Behavior of I/O Streams” on page 36.
–i	Standard input to mprun is sent only to rank 0, and not to all other ranks.
–J	Show the jid, cluster name, and number of processes after executing. See “Getting Information” on page 43.
–j *jid*	Run on the same node(s) as the job with job ID jid. See “Running on the Same Node(s) as Another Specified Job” on page 26.
–Mf ",*node1* [*numprocs*],*node2* [*numprocs*][,…]" | *rankmapfile*	Assigns processes in rank order to specified nodes. See “Using the –Mf Option” on page 26.
–n	Read stdin from . See “Specifying the Behavior of I/O Streams” on page 36.

TABLE 3-2 Options for `mprun` *(Continued)*

Option	Meaning
`–N`	Do not open any standard I/O connections. See “Specifying the Behavior of I/O Streams” on page 36.
`–np` *number*	Request the specified number of processes. See “Controlling Process Spawning” on page 24.
`–Ns`	Disable spawning of multiple processes from a job on SMPs. See “Option Priority Matrix” on page 21.
`–o`	Prefix each output line with the rank of the process that wrote it.
`–p` *partition*	Run in the specified partition. See “Specifying the Partition” on page 23.
`–R` "*resource_string*"	Specify conditions for choosing nodes. See “Expressing More Complex Resource Requirements With `–R`” on page 29.
`–S`	Settle for the available number of nodes (used with –np). See “When Number of Processes Exceeds Number of CPUs” on page 25.
`–U` *user*	Execute with the specified user ID or user name. See “Executing With a Different User or Group Name” on page 43.
`–u`	Allow inclusion of “independent” nodes. See “When Number of Processes Exceeds Number of CPUs” on page 25
`–V`	Display version information. See “Getting Information” on page 43.
`–W`	Wrap the requested processes on the available CPUs (used with `–np`). See “When Number of Processes Exceeds Number of CPUs” on page 25.
`–Ys`	Allow spawning on SMPs. See “Option Priority Matrix” on page 21.
`–Z` *group_size*	Group processes by the specified number, assigning ranks in groups of *group_size*. Multiple groups may be assigned to a node. The `mprun –np` value determines how many ranks are actually assigned. Explicit `–np` values should be used, but if `–np 0` is specified, as many groups of *group_size* as possible will be used without overcommitting any CPUs. See “Expressing More Complex Resource Requirements With `–R`” on page 29.
`–Zt` *processor_tiling_factor*	Group processes by the specified number, assigning ranks in groups of *processor_tiling_factor*. One group will be assigned per node. The `mprun –np` value determines how many ranks are actually assigned. Explicit `–np` values should be used, but if `–np 0` is specified, as many groups of *processor_tiling_factor* as possible will be used without overcommitting any CPUs.

Option Priority Matrix

When choosing nodes and processors to fit the requirements specified in an `mprun` request, the CRE considers the options in a certain 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 an option causes other options in the command line to be ignored, even if they appear later in the command line. The tables in this section describe how the various options interact when conflicts arise.

Rank Placement

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:

Option	Nullifies Previous	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

I/O Control

In addition to rank placement options, there are options that control the I/O for a job. The following table summarizes an interaction matrix for these options:

Option	Nullifies Previous
–B	–D –N –B –I
–D	–D –N –B –I
–I	–D –N –B –I
–N	–D –N –B –I

Other Options

Of the other basic options that control mprun attributes some can be issued multiple times, with the last one issued in control. The following table summarizes an interaction matrix for these options:

Option	Nullifies Previous
–np	–np
–Ys	–Ns
–Ns	–Ys
–W	–S
–S	–W
–U	–U
–G	–G
–A	–A
–C	–C
–c	–c
–p	–p
–r	–r

Specifying Where a Program Is to Run

The mprun command provides you with considerable flexibility in specifying where you want your job to run.

- "Specifying the Cluster" on page 23 describes how to choose the cluster on which you want your program to run.
- "Specifying the Partition" on page 23 describes how to choose the partition in which a program is to run.
- "Controlling Process Spawning" on page 24 describes how to specify how many processes are to be started and how they should be mapped to nodes.
- "Running on the Same Node(s) as Another Specified Job" on page 26 describes how to run your job on the same node as another job.
- "Expressing More Complex Resource Requirements With –R" on page 29 describes a syntax for specifying complex requirements that can't be encapsulated in the basic command-line options.

- "Mapping MPI Ranks to Nodes" on page 26 describes several methods of controlling the way your job's processes map to specific CPUs.

In cases where your specified requirements can be met by more than one node, the cluster chooses the least-loaded node, unless you have specified other sorting criteria.

Specifying the Cluster

By default, your job will run on the cluster where you are logged in. (The cluster name is defined by the system administrator in the `SUNHPC_CLUSTER` environment variable.)

If you are logged in on a machine that is connected to the Sun HPC cluster where you want to run your job, but the machine is not a node in the cluster, use

```
% mprun –c cluster_name
```

to specify the cluster.

Note – You can find the name of the cluster by running `mpinfo –C` on any node in the cluster. The cluster name is the same as the host name of the cluster's master node. See "Displaying Cluster Information" on page 55 for additional details.

Specifying the Partition

Use `mprun –p` to specify the partition in which you want your program to run. The partition must be in the enabled state. For example,

```
% mprun –p part2 a.out
```

specifies that `a.out` is to be run in the partition `part2`.

The CRE chooses the partition where your job will run by considering various methods of specification, in order of priority. If a partition is not specified by a particular method, the CRE will try the next specification method in the list to choose a partition.

1. The partition name specified in the `–p` option in the command line. If the specified partition is not valid, the command will fail.
2. The partition name specified in the `–p` option in the `MPRUN_FLAGS` environment variable. If the specified partition is not valid, the command will fail.

3. The partition name specified by the system administrator in the SUNHPC_PART environment variable, if enabled.

4. The login partition, if enabled.

5. The partition name specified by the system administrator in the default_interactive_partition attribute, if set.

6. If none of the first five criteria are met, the CRE searches for one enabled partition. If more than one enabled partition is found, the job fails.

The mpinfo command will tell you the names of enabled partitions in the cluster, along with other useful information about cluster resources. See "mpinfo: Configuration and Status" on page 50 for a description of mpinfo.

Controlling Process Spawning

Specify the Number of Processes

Use the –np option to specify the number of processes you want to start; the default is 1. This option is typically used with a Sun MPI program.

For example,

```
% mprun –p part2 –np 4 a.out
```

specifies that you want four copies of a.out to start on the nodes of the partition named part2.

You can also specify 0 as the –np value. The CRE will start one process per CPU on each available CPU. Thus, if the partition part2 has six available CPUs, the command

```
% mprun –p part2 –np 0 a.out
```

will start six copies of a.out.

Limit to One Process Per Node

Use the –Ns option to limit the number of processes to one per node. This prevents nodes from spawning more processes regardless of the number of CPUs they have.

When Number of Processes Exceeds Number of CPUs

When you request multiple processes (via the `–np` option), the CRE attempts to start one process per CPU. If you request more processes than the number of available CPUs, you must use one of three options to prevent `mprun` from failing:

- `–W`

  Use the `–W` option if you want the processes to *wrap*—that is, to allocate multiple processes to each CPU, which will execute their respective sets of processes one by one. For example, if the partition `part2` has six available CPUs and you specify

  ```
  % mprun –p part2 –np 10 –W a.out
  ```

  the CRE will start 10 processes on the six CPUs.

Note – When the CRE wraps processes, it distributes them according to load-balancing rules. Therefore, you will not be able to predict where they will execute.

- `–S`

  If you prefer to have a certain number of processes started, but are willing to settle for however many CPUs are available, use the `–S` option. The CRE will start one process on each available CPU. Thus, if you issue the same command as above, but substitute `–S` for `–W`:

  ```
  % mprun –p part2 –np 10 –S a.out
  ```

  and six CPUs are available on `part2`, then six copies of `a.out` will start, one per CPU.

- –u

  If you specify `–u`, the CRE will look for independent nodes outside the partition to make up the difference between the requested number of processes and the number of available CPUs. To be eligible, an external node must satisfy three requirements:

  - It must be enabled.
  - It cannot belong to another partition that is currently enabled.
  - It must be running the same version of the Solaris operating environment as the nodes in the partition where you are running.

  If you specify `–np 0`, `–S`, or `–W`, the search will be restricted to the partition you are in.

Running on the Same Node(s) as Another Specified Job

Use the `–j` option to specify that the program you want to execute should run on the same node or nodes as a particular job ID (jid). For example, to run `a.out` on the same node(s) as a job whose job ID is 85, issue the command

```
% mprun –j 85 a.out
```

For more control, you can combine `–j` with other options, but they must *follow* it on the command line. For example, if `–np`, together with `–W` or `–S`, follows `–j` on the command line, `–j` determines which nodes to run on, while the other options determine the number of processes to map onto these nodes.

You can use the `mpps` command to find out the job ID of any job.

Mapping MPI Ranks to Nodes

The –Mf option described in the next section is the best way to map rank processes to specific nodes. The –Z and –Zt options organize a job's processes into subsets of a specified size and group the processes in a subset on the same node. (See "Using `–Z` and `–Zt`" on page 29.) You can also use the –R option to map ranks to nodes, but its use for this purpose is discouraged, and may not be supported in future releases. (See "Using RRS to Map Ranks to Nodes" on page 34.)

Using the `–Mf` Option

The `–Mf` option allows you to control the mapping of rank processes to nodes, taking a rank map as an argument. A rank map is a rank-to-node specification that lists node names in the order in which you want your processes to run on them. It can be included in the command line as a string or saved in a file, in which case the file name is used as the argument. If you do not specify the number of process ranks to be used on a node, the default value of 1 will be used. This section describes the syntax of the `–Mf` option, its *rankmap* argument, and the conditions that must be met to use it successfully.

Syntax

Suppose you want your 7-process job to run like this:

- Ranks 0, 1, and 6 will run on `mars`.

- Rank 2 will run on venus.
- Ranks 3, 4, and 5 will run on jupiter.

Compose your rank map, with the number of process ranks assigned to each node in the list, either directly in the command line or in a file that you refer to in the command line. Your rank map must use the correct syntax:

- *Inline rank map* – Node names are listed in the order in which they will be used, and names may be repeated. The list must begin with a comma (it may *not* begin with a space), the node names must be separated by a comma, and the entire list must be enclosed within quotation marks. If you specify more than one rank to be mapped to a node, the number is separated from the node name by space or tabs. Using this syntax, you can obtain the desired result in our example like this:

  ```
  –Mf ",mars 2,venus,jupiter 3,mars"
  ```

  (You could obtain the same results using –np and –W with –Mf. See “Using –np” on page 28 for more information.)

- *The rankmap file* – Again, node names are listed in the order in which they will be used, and names may be repeated. 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 rankmap files, so you will probably find it easier to list each node on its own line. For our example, the file, rankmapfile, would include a list like this:

```
mars 2
venus
jupiter 3
mars
```

  and the –Mf syntax would look like this:

  ```
  –Mf rankmapfile
  ```

Qualifying Conditions

Number of Nodes Per CPU

For a given node, if you request a number of process ranks that is greater than the number of CPUs on that node, the CRE will nonetheless comply with your request unless the value of total_max_procs prevents it.

Using `–np`

If the value specified in 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.

Example. Suppose you want to run your job like this:

- Ranks 0, 2, 4, and 6 on `mars`
- Ranks 1, 3, 5, and 7 on `venus`

To accomplish this, your command line would include:

```
-np 8 -W -Mf ",mars,venus"
```

(Since the default number of process ranks per node is 1, you do not need to specify a the number of processes on each node.)

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`.

Example. Suppose your command line includes these options:

```
-np 5 -Mf ",mars 2,venus 3,jupiter"
```

Then the process ranks will map like this:

- Ranks 0 and 1 will run on `mars`.
- Ranks 2, 3, and 4 will run on `venus`.
- The job is done. No ranks will run on `jupiter`.

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

Example. Suppose your command line includes these options:

```
-np 0 -Mf ",mars 2,venus 3,jupiter"
```

Then 6 processes will run, and they will map like this:

- Ranks 0 and 1 will run on `mars`.
- Ranks 2, 3, and 4 will run on `venus`.
- Rank 5 will run on `jupiter`.

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 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.

Using –Z and –Zt

The –Z and –Zt options causes the CRE to organize a job’s processes into subsets of a specified size and to group all processes in a subset on the same node. You specify the subset size with a numerical argument to –Z or –Zt. For example,

```
% mprun –Z 3 –np 8 a.out
```

groups the job’s processes by threes.

With –Z, these groups *may* be distributed onto different nodes, but there is no guarantee that they will be; two or more groups may be started on the same CPU.

With –Zt, no two blocks will be mapped to the same node — three nodes will be used.

Note – The –Z option is incompatible with the –S and –W options.

Expressing More Complex Resource Requirements With –R

Use the –R option to express complex node requirements that are not accessible via the options discussed above.

The –R option takes a *resource requirement specifier* (RRS) as an argument. The RRS is enclosed in quotation marks and provides the settings for any number of attributes that you want to use to control the selection of nodes. You combine multiple attribute settings using the logical & (AND) and | (OR) operators.

The CRE parses the attribute settings in the order in which they are listed in the RRS, along with other options you specify. The CRE 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 the specified criteria. If you are starting a single process, the CRE starts the process on the CPU that's first in the list. If you are starting *n* processes, the CRE starts them on 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.

Specifying Resource Attributes

TABLE 3-3 lists predefined attributes you can include in an RRS. Your system administrator may also have defined attributes specific to your Sun HPC cluster. You can see what settings these administrator-defined attributes have with the `mpinfo` command.

TABLE 3-3 Standard RRS Attributes

Attribute	Meaning
`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 hostname.
`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.

TABLE 3-3 Standard RRS Attributes

Attribute	Meaning
`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.

The CRE recognizes two types of attributes, value and boolean.

Value-Based Attributes

Value attributes can take a literal value or a numeric value. Or, depending on the operator used, they may take no value. (Note that `–R` is also used with `mpinfo`. See Chapter 4 for more information.)

- Attributes with a literal value take a name as a setting. Use an equal sign and the name after the attribute to show the setting. For example,

  ```
  % mprun –R "name = hpc–demo" a.out
  ```

  ```
  % mpinfo –N –R "partition.name=part1"
  ```

- Attributes with a numeric value include an operator and a value. For example,

  ```
  % mprun –R "load5 < 4" a.out
  ```

  specifies that you only want nodes whose individual load averages over the previous 5 minutes were less than 4.

  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='5.7'"
  ```

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

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

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

TABLE 3-4 identifies the operators that can be used in RRS expressions.

TABLE 3-4 Operators Valid for Use in RRS

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.)
<<	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 –
=, !=, >=, <=, >, <, <<, >>
!
&, |
?
```

If you use the << or >> operator, the CRE does not provide load-balancing. In the previous example, the 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
```

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 2. 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

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
```

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

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

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

Examples

Here are some examples of the –R option in use.

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 the command

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

Using RRS to Map Ranks to Nodes

To map ranks to specific nodes, use the `-Mf` option. While you can use the `-R` option to map ranks to nodes, its use for this purpose is discouraged, and may not be supported in future releases.

That said, you can construct an RRS expression (see "Expressing More Complex Resource Requirements With `-R`" on page 29) that causes `mprun` to distribute a specified number of processes (MPI ranks) to a set of nodes in a specified order

The RRS expression assigns to each node in the set a single-character alias preceded by a number, which together make up a sequence of count/alias pairs. For example:

```
"[2a2b2c2d]:a.name=hpc-node0 & b.name=hpc-node1 & c.name=hpc-node2 &
d.name=hpc-node3"
```

The number that precedes a node's alias tells the CRE how many processes to start on that node. In this example, it assigns two processes to each of the nodes defined by the aliases `a`, `b`, `c`, and `d`. This number can be different for each node, but it *must not* exceed the number of CPUs on that node.

The CRE distributes processes to the nodes in the order in which they are listed in the RRS expression, starting the rank 0 process on the first node in the list. Once the prescribed number of processes have been started on the first node, the CRE moves to the second node and then to subsequent nodes, starting the specified number of processes on each node in turn. An alias cannot be repeated in the sequence, but one node can be defined with more than one alias.

The RRS rank-mapping expression must satisfy the following conditions:

- Up to 26 node aliases can be defined; aliases are *not* case-sensitive. Every node alias must be preceded by a number, which may have more than one digit.
- The number of processes assigned to a given node *cannot* be greater than the number of CPUs on that node.

- The `-np` value *cannot* be greater than the total number of processes allocated by the RRS expression. You cannot use use the `-W` option to get around this restriction by wrapping the processes.

The following example shows this technique being applied on a 4x4 partition. Two processes are started on each of four, four-CPU nodes.

```
% mprun -o -np 8 -R "[2a2b2c2d]:a.name=hpc-node0 & b.name=hpc-node1
& c.name=hpc-node2 & d.name=hpc-node3" uname -n
r0:hpc-node0
r1:hpc-node0
r2:hpc-node1
r3:hpc-node1
r4:hpc-node2
r5:hpc-node2
r6:hpc-node3
r7:hpc-node3
```

The `-o` option prepends each output line with the MPI rank of the process that writes it. Two CPUs on each node are not participants in this job.

The next example shows different numbers of processes being allocated to each node. One process is started on the first node, two on the second, and so forth.

```
% mprun -o -np 10 -R "[1a2b3c4d]:a.name=hpc-node0 & b.name=hpc-node1
& c.name=hpc-node2 & d.name=hpc-node3" uname -n
r0:hpc-node0
r1:hpc-node1
r2:hpc-node1
r3:hpc-node2
r4:hpc-node2
r5:hpc-node2
r6:hpc-node3
r7:hpc-node3
r8:hpc-node3
r9:hpc-node3
```

The following example shows the error message that is returned when the number of processes assigned to a node exceeds the number of CPUs on that node.

```
% mprun -o -np 6 -R "[2a1b3c]:a.name=hpc-node0 & b.name=hpc-node1 &
c.name=hpc-node0" uname -n
mprun: no_mp_jobs: No nodes in partition satisfy RRS
```

In this case, the node hpc-node0 is aliased twice — as 2a and 3c — so that it can be repeated in the sequence. This use of multiple aliases is legal, but hpc-node0 has four CPUs and the total number of processes assigned by 2a and 3c is five, which violates the second condition listed above.

The next example shows what happens when an alias does not start with a number. In this case, the alias for hpc-node0 violates the first condition listed above.

```
% mprun -o -np 6 -R "[a2b3c]:a.name=hpc-node0 & b.name=hpc-node1 &
c.name=hpc-node2" uname -n
mprun: no_mp_jobs: No nodes in partition satisfy RRS
```

Specifying the Behavior of I/O Streams

Introducing mprun I/O

By default, all standard output (stdout) and standard error (stderr) from an mprun-launched job will be merged and sent to mprun's standard output. This is ordinarily the user's terminal. Likewise, mprun's standard input (stdin) is sent to the standard input of all the processes.

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

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

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

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

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

The mprun command's own options allow you to control I/O in other ways. For example, rather than making remote processes communicate with mprun (when it may not be necessary), you can make each process write to or read from a file on the node on which it is running. 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
```

mprun also provides options that you can use to control standard output and standard error streams. For example, you can

- Use the –D option to make the standard error from each process go to the standard error of mprun, instead of its standard output. For example,

  ```
  % mprun –D a.out
  ```

  sends standard output from a.out to the standard output of mprun and sends the standard error of a.out to the standard error of mprun.
- Use the –B option to merge the standard output and standard error streams from each process and direct them to files named out.*jid*.*rank*, where *jid* is the job ID of the job and *rank* is the rank of this process within the job. The files are located in the job's working directory. There is no standard input stream.
- Use the –N option to shut off all standard I/O to all the processes. That is, with this option, you specify that there are to be no stdin, stdout, and stderr connections. Use the –N option for situations in which standard I/O is not necessary; 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.
- Use the –n option to cause stdin to be read from /dev/null. This can be useful when running mprun in the background, either directly or through a script. Without –n, mprun will block in this situation, even if no reads are posted by the remote job. When –n is specified, the user process encounters an EOF if it attempts to read from stdin. This is comparable to the behavior of the –n option to rsh.

Note – 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.

Creating a Custom Configuration

Use the –I option to specify a custom configuration for the I/O streams associated with a job, including standard input, output, and error. The –I option takes as an argument a comma-separated series of *file descriptor strings*. These strings specify what is to happen with each of the job's I/O streams.

In Solaris, each process has a numbered set of *file descriptors* associated with it. The standard I/O streams are assigned the first three file descriptors:

- 0 – standard input (stdio)
- 1 – standard output (stdout)
- 2 – standard error (stderr)

The argument list to `-I` can include a string for each file descriptor associated with a job; if any file descriptor is omitted, its stream won't be connected to any device.

Restriction: 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`.

The file descriptor strings in the `-I` argument list can be in any order. Quotation marks around the strings are optional.

File Descriptor Attributes

The file descriptor string assigns one or more of the following attributes to a file descriptor:

- `r` – File descriptor is to be read from.
- `w` – File descriptor is to be written to.
- `p` – File descriptor is to be attached to a pseudo-terminal (pty).

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`.

The following attributes are output-related and thus can only be used in conjunction with `w`:

- `l` – Line-buffered output.
- `t` – Tag the line-buffered output with process rank information.
- `a` – Stream is to be appended to the specified file.

Note – NFS does not support append operations.

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 don't 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"
Hello
Hello
```

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"
r0:Hello
r1:Hello
```

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.

File Descriptor String Syntax

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.

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

fdattr=filename

For example,

```
10w=output
```

says 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. See “Introducing `mprun` I/O” on page 36. 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.

`mprun` *Options versus 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 3-5 `mprun` 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.

These shortcuts are not exact substitutions. The CRE uses ptys correctly, whether the `–I` option is present or absent. Also, the 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.

Caution Regarding the Use of `–i` Option

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.

Changing the Working Directory

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. (Setting a path with –C does not affect where the CRE looks for executables.) If you don't specify –C, the default is the current working directory. For example,

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

changes the working directory for a.out to /home/collins/bin.

Executing With a Different User or Group Name

Use the –U option to execute with the specified user ID or user name. For example,

```
% mprun –U traveler a.out
```

executes a.out as the user traveler.

Use the –G option to execute with the specified group ID or group name.

```
% mprun –G qa–team a.out
```

executes a.out as the group qa–team.

You must have the appropriate level of permissions to use these options. For example, you must belong to the group you specify, or be the superuser.

Getting Information

Use the –h option to display a list of mprun options and their meanings.

Use the –V option to display the command's version number.

If you specify either –h or –V, it must be the only option on the command line.

Use the –J option to display the program's jid, along with the name of the cluster and the number of processes, after executing mprun.

Specifying a Different Argument Vector

By default, mprun passes the vector of a program's command-line arguments to the program in the standard way. 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 (argv[0]), and arg1 and arg2 are the second and third elements.

In cluster-level programming, it is sometimes useful to specify an argv[0] that is not the name of the program. You can use the –A option to do this. The argument to –A is the name of the program to be executed. You can then follow this with an argument of your choice in the arg0 position. For example, if you want to pass newarg as the argv[0] to the program a.out, along with arg1 and arg2, you could issue the command

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

Exit Status

The exit status of mprun specifies the number of processes that exited with nonzero exit status.

Omitting mprun

You can execute a serial program without using mprun. For example, you could simply type

```
% a.out
```

In that case, the program executes locally, on the node where you are logged in. By doing this, however, you give up the benefits of load-balancing provided by the CRE.

Note – You cannot run Sun MPI programs in this way; you must use mprun.

Sending a Signal to a Process

The mpkill command is comparable to the Solaris kill command. You use it to terminate all processes of the jobs with the specified job IDs running on the Sun HPC cluster, or to send a signal to it.

You can send any standard Solaris signal. Use the –l option to obtain a list of the supported signals, or the –d option to list them along with brief descriptions.

Specify the signal's name or number, followed by the job ID, to send that signal to the job. For example,

```
% mpkill -CONT 59
```

sends a SIGCONT to the processes that constitute job 59.

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

To find out a job's job ID, use the command mpps or the –J option to mprun.

mpkill returns the following status values:

- 0 – The command executed successfully.
- 1 – An error was encountered 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.

CHAPTER 4

Getting Information

The CRE user interface includes two commands for obtaining information about a Sun HPC cluster's configuration (mpinfo) and information about jobs running on the cluster (mpps).

mpps: Finding Out Job Status

The mpps command is comparable to the Solaris ps command. It returns information about jobs and processes currently running on the Sun HPC cluster.

By default mpps shows basic information about the user's jobs currently running in the default partition. 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
```

In the response,

- JID is the executing program's job ID.
- NPROC is the number of processes in the job.
- UID is the user ID of the person who executed the program.
- STATE is the execution status of the job's processes. (See TABLE 4-1 on page 48 for a list of possible process states.)
- AOUT is the name of the executable program.

TABLE 4-1 lists the states reported by mpps. Some states refer only to jobs, some only to processes, and some to both. (See "Displaying Process Information" on page 49.)

TABLE 4-1 Job and Process States

State	mpps Display	Meaning
CORE	CORE	The job or process exited due to a signal and core was dumped.
COREING	CRNG	The job is exiting due to a signal. The first process to die dumped core.
EXIT	EXIT	The job or process exited normally.
EXITING	EXNG	The job is exiting. At least one process exited normally.
FAIL	FAIL	The job or process failed on startup or was aborted.
FAILING	FLNG	Initialization of the job failed, or a job abort has been signaled.
ORPHAN	ORPHAN	The process has been "orphaned," that is, the node on which it exists has gone offline.
RUNNING	RUN	The job or process is running.
SEXIT	SEXIT	The job or process exited due to a signal.
SEXITING	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.
SPAWNING	SPAWN	The job or process is being spawned.
STOP	STOP	The job or process is stopped.

Use the –f option to display, in addition, the start time for each job and the job's arguments.

Use the –e option to display information on all jobs, not just your jobs.

Displaying the Partition

To show information about jobs running in all partitions, use the –A option.

To show information about jobs running in a specific partition, use the –a option, followed by the name of the partition.

Displaying Process Information

Use the –p option to also view information about the processes that make up the jobs. The process information is listed below each job. 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
```

In this example,

- RANK is the process's rank within the job.
- PID is the process's process ID.
- STATE is the process's execution status.
- NODE is the node on which the process is running.

Displaying Specific Process and Job Information

You can also use the –P option to display one or more specific process values and the –J option to display one or more job values. Separate multiple values either with spaces or with commas and no spaces.

Arguments to –P are

- rank – The rank of the process within the job.
- pid – The process's 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 on which the process is executing.
- node – The name of the node on which the process is executing.

You can list these via the –lp option.

Arguments to –J are

- part – The name of the partition in which the job will run.
- jid – The job's unique ID, which can be used as an argument to mpkill.
- nproc – The number of processes requested (the actual number of processes started may differ if the –W or –S flags were used with mprun).
- uid – The user on whose behalf the job will be run (normally the user who submitted the job; see the –U flag to mprun for details).

- gid – The group on whose behalf the job will be run (normally the group of the user who submitted the job; see the –G flag to mprun for details).
- state – There are five states:
  - 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 for this job. This is not always equal to the number of processes started for this job, since processes that have exited are not counted.
- wkdir – The directory in which the job's processes will be (or were) started.
- aout – The name of the program to be run.
- paout – The full path of the program to be run.
- ctime – The job creation time (when mprun was invoked for the job).
- args – The command-line arguments for the program to be run.
- stime – The time the job was started.
- prio – The job priority (higher numbers run first).

mpinfo: Configuration and Status

Use the mpinfo command to display information about the configuration of partitions and nodes, and status information about nodes.

Overview

You can display information on all partitions or nodes, or on any subset of them. You can either list the partitions or nodes, or you can use the –R option, along with a resource requirement specifier (RRS), to have the CRE determine which objects should be displayed. See "Expressing More Complex Resource Requirements With –R" on page 29 for information on RRSs. If you specify a partition, you must include only partition attributes in the RRS; if you specify a node, you must use only node attributes.

Use the –A option to specify an attribute whose value you want to display. If you want to display more than one attribute, separate them by commas with no spaces. Alternatively, you can issue multiple –A options on the same command line. If you omit –A, mpinfo displays values for a default set of attributes.

Use the –v option to display information about all attributes for one or more partitions or nodes. These include attributes defined by the system administrator.

When a Boolean attribute is displayed, yes indicates that the attribute is set, and no indicates that the attribute is not set.

Displaying Partition Information

Use the –P option to display information for all partitions.

Use the –p option, followed by the name of the partition, to display information about an individual partition. To display information about multiple partitions, list the names, either separating them with commas and no spaces or enclosing the list in quotation marks.

Partition attributes whose settings you can view via mpinfo are shown in TABLE 4-2 on page 52; the heading displayed for each attribute is shown in parentheses after its description.

The following summarizes various points discussed earlier.

- You can specify one or more of these attributes via the –A option, or as part of an RRS as an argument to the –R option. You can use either the attribute's real name or, in some cases, a shorter version.
- For attributes that are defined as negatives (for example, no_logins), you can specify a positive version (for example, logins) for –A.
- You can list the settings of all attributes (including any system administrator–defined attributes) on a per-partition basis via the –v option.

- You can list the names and brief descriptions of these attributes via the –lp option.

TABLE 4-2 Partition Attributes Available via mpinfo

Attribute (mpadmin form)	Description (mpinfo output heading)
enabled	Set if the partition is enabled, that is, if it is ready to accept jobs (ENA).
maxt	Maximum number of simultaneously running processes allowed on each node of the partition (MAXT).
name	Name of the partition (NAME).
login	Allow logins. When login is set, LOG is set. Note that this is the inverse of the mpadmin meaning. (LOG).
mp	Allow multinode jobs. When no_mp_jobs is unset, MP is set. Note that this is the inverse of the mpadmin meaning. (MP).
nodes	Number of nodes in the partition (NODES).

The following example illustrates the default mpinfo output for partitions:

```
% 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 following example displays the names, numbers of nodes, and enabled status for all partitions:

```
% mpinfo –A name,enabled,nodes –P
NAME         ENA NODES: Tot(cpu) Enb(cpu) Onl(cpu)
part10       no             1(  4)   1(  4)   1(  4)
part11       yes            1(  4)   1(  4)   1(  4)
```

Displaying Node Information

Use the –N option to display information about all nodes.

Use the -n option, followed by the name(s) of one or more nodes. When listing multiple node names, separate the names with commas without spaces.

The following table shows the node attributes that you can display via mpinfo. The heading that is displayed for each attribute is shown in parentheses at the end of each description.

Note these points:

- You can specify one or more of these attributes via the -A option, or as part of an RRS as an argument to the -R option. You can use either the attribute's real name or, in some cases, a shorter version.
- You can list the settings of all attributes (including any system administrator–defined attributes) on a per-node basis via the -v option.
- You can list the names and brief descriptions of these attributes via the -ln option.

TABLE 4-3 Node Attributes Available via mpinfo

Attribute	Short Form	Description (mpinfo output heading)
cpu_idle	idle	Percent of time CPU is idle (IDLE).
cpu_iowait	iowait	Percent of time CPU spends waiting for I/O (IWAIT).
cpu_kernel	kernel	Percent of time CPU spends in kernel (KERNL).
cpu_type	cpu	CPU architecture (CPU).
cpu_user	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	manuf	Hardware manufacturer (MANUFACTURER).
mem_free	memf	Node's available RAM (in Mbytes) (FMEM).
mem_total	memr	Node's total physical memory (in Mbytes) (MEM).
name		Name of the node (NAME).

TABLE 4-3 Node Attributes Available via `mpinfo` *(Continued)*

Attribute	Short Form	Description (`mpinfo` output heading)
`ncpus`	`ncpu`	Number of CPU modules in the node (`NCPU`).
`os_arch_kernel`	`mach`	Node's kernel architecture (`MACH`).
`os_max_proc`	`maxproc`	Maximum number of processes allowed on the node (note that this is *all* processes, including cluster daemons) (`MPROC`).
`os_name`	`os`	Name of the operating system running on the node (`OS`).
`os_release`	`osrel`	Operating system's release number (`OSREL`).
`os_release_maj`	`osmaj`	The major number of the operating system release number (`MAJ`).
`os_release_min`	`osmin`	The minor number of the operating system release number (`MIN`).
`os_version`	`osver`	Operating system's version (`OSVER`).
`partition`		The partition of which the node is a member (`PARTITION`).
`serial_number`	`serno`	Hardware serial number (`SERIAL`).
`swap_free`	`swapf`	Node's available swap space (in Mbytes) (`FSWP`).
`swap_total`	`swapr`	Node's total swap space (in Mbytes) (`SWAP`).

The following is an example of the `mpinfo` output for nodes:

```
% mpinfo –N
NAME  UP PARTITION OS    OSREL NCPU FMEM   FSWP    LOAD1 LOAD5 LOAD
15
node0 y  p0        SunOS 5.6   1    0.89   158.34  0.09  0.11  0.13
node1 y  p0        SunOS 5.6   1    31.41  276.12  0.00  0.01  0.01
node2 y  p1        SunOS 5.6   1    25.59  279.77  0.00  0.00  0.01
node3 y  p1        SunOS 5.6   1    25.40  279.88  0.00  0.00  0.01
```

The following example shows only the names of nodes and the partition they're in:

```
% mpinfo –N –A name,partition
NAME         PARTITION
node0        part0
node1        part0
node2        part1
node3        part1
```

Displaying Cluster Information

Use the –C option to display information about the entire cluster. For example,

```
% mpinfo –C
NAME   ADMINISTRATOR    DEF_INTER_PART
node0  wmitty           part0
```

where:

- NAME – The name of the cluster, which is the host name of the master node.
- ADMINISTRATOR – The name of its administrator
- DEF_INTER_PART – The default interactive partition

CHAPTER 5

Debugging Programs

The Prism development environment is a component in the Sun HPC ClusterTools suite of software. You can use it to debug and visualize data in serial or message-passing programs on a Sun HPC cluster. For complete information on the Prism environment , see the *Prism User's Guide* and *Prism Reference Manual*. This chapter gives a brief overview of how to start up Prism.

To use Prism, you must first log in to the Sun HPC cluster, as described in Chapter 2. If you are using the graphical version of the Prism environment, you must be running the Solaris 2.6, 7, or 8 operating environment with either OpenWindows or CDE.

You can start the Prism environment by entering

```
% prism
```

and then loading your executable program from within Prism.

Alternatively, you can specify the program's name on the command line. In this case, the environment will start up with the program already loaded. For example,

```
% prism a.out
```

Once the program is loaded in the Prism environment, you can execute it, debug it, and visualize data in it. The program executes on the same node as the Prism environment.

Debugging Sun MPI Programs

If you are going to use Prism to debug a Sun MPI program, use the `–np` option with `mprun` to specify how many processes are to be started. For example,

```
% prism –np 4 a.out
```

When you use the –np option, you can also use other Prism options, such as –p, to determine where the job's processes are to run and how they are mapped onto nodes. For example,

```
% prism –p part0 –np 4 a.out
```

starts the environment as well as the message-passing program a.out on the partition part0. Client Prism processes are also started with each of the a.out processes. They receive instructions from and return information to the master Prism daemon that is started by mprun.

You can attach to a running Sun MPI program by specifying its job ID after the name of the executable program. For example,

```
% prism –np 1 a.out 462
```

You can find out the job ID of a program by issuing the mpps command or by using the –J option to mprun.

The setting of the MPRUN_FLAGS environment variable applies to both mprun starting the Prism environment and to the Prism environment starting the parallel processes. This means that the default options are likely to be incorrect for one or the other, since you would typically want to start the Prism environment on one node in a shared partition, and the Sun MPI processes on multiple nodes, possibly in a dedicated partition.

APPENDIX **A**

Using LSF With Sun HPC ClusterTools

The Sun HPC ClusterTools products can be teamed with LSF Suite 3.2.3, Platform Computing Corporation’s resource management software. This appendix explains how to execute Sun MPI applications on a Sun HPC cluster using Platform Computing Corporation’s LSF Batch software, enhanced by the LSF Parallel facility.

Note – Users should read the *LSF Batch User’s Guide* for detailed information about the LSF Batch system’s general features.

About LSF Suite 3.2.3

LSF Suite 3.2.3 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 software are: LSF Base, LSF Batch, and LSF Parallel.

- *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.

LSF Suite 3.2.3 runs with Sun HPC ClusterTools software under Solaris 2.6 or Solaris 7 (32- or 64-bit).

Job Execution Modes

If you're 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 launched immediately.
- These queues are created by the system administrator. Each queue is defined by a set of job-launching 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 launched.

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.

Starting Sun MPI Programs

This section explains the basic steps for starting up message-passing programs on a Sun HPC cluster using LSF Batch services. It covers the following topics:

- "Using Parallel Job Queues" on page 61
- "`bsub` Overview" on page 62
- "Submitting Jobs in Batch Mode" on page 62
- "Submitting Interactive Batch Jobs" on page 63
- "Using the `–sunhpc` Option" on page 64

For information about developing, compiling, and linking Sun MPI programs, see the *Sun MPI Programming and Reference Guide*.

Note – Running parallel jobs with LSF Suite 3.2.3 is supported on up to 1024 processors and up to 64 nodes.

Using Parallel Job Queues

Distributed MPI jobs must be submitted via batch queues that have been configured to handle parallel jobs. This parallel capability is just one of the many characteristics that a system administrator can assign when setting up a batch queue.

You can use the command `bqueues –l` to find out which job queues support parallel jobs, as shown in FIGURE A-1.

The `bqueues –l` output contains status information about all the queues currently defined. Look for a queue that includes the line:

```
JOB_STARTER: pam
```

which means it is able to handle parallel (distributed MPI) jobs. In the example shown in FIGURE A-1, the queue `hpc` is defined in this way.

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.

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

           :
           :

SCHEDULING POLICIES:  INTERACTIVE

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

FIGURE A-1 Finding a Parallel Queue With `bqueues –l`

If no queues are currently configured for parallel job support, ask the system administator to set one or more up in this way.

Once you know the name of a queue that supports parallel jobs, submit your Sun MPI jobs explicitly to them. For example, the following command submits the job hpc–job to the queue named hpc for execution on four processes.

```
hpc–demo% bsub –q hpc –n 4 hpc–job
```

Additional examples are provided in "Submitting Jobs in Batch Mode" on page 62 and "Submitting Interactive Batch Jobs" on page 63.

Note – To use LSF Batch commands, your PATH variable must include the directory where the LSF Base, Batch, and Parallel components were installed. The default installation directory is /opt/SUNWlsf/bin. Likewise, your PATH variable must include the ClusterTools software installation directory; the default location for ClusterTools components is /opt/SUNWhpc/bin.

---

bsub Overview

The command for submitting Sun MPI jobs to the LSF Batch system is bsub, just as it is for submitting nonparallel batch jobs. The command syntax is essentially the same as well, except for an additional option, –sunhpc, which applies specifically to Sun MPI jobs. The bsub syntax for parallel jobs is

bsub [*basic_options*] [–sunhpc *sunhpc_args*] *job*

The *basic_options* entry refers to the set of standard bsub options that are described in the *LSF Batch User's Guide*. The –sunhpc option allows Sun HPC–specific arguments to be passed to the MPI job *job*.

"Submitting Jobs in Batch Mode" on page 62 and "Submitting Interactive Batch Jobs" on page 63 describe how to use bsub to submit jobs in batch and interactive batch modes, respectively. The –sunhpc option is discussed in "Using the –sunhpc Option" on page 64.

Refer to the *LSF Batch User's Guide* for a full discussion of bsub and associated job-submission topics.

Submitting Jobs in Batch Mode

The simplest way to submit a Sun MPI job to the LSF Batch system is in batch mode. For example, the following command submits hpc–job to the queue named hpc in batch mode and requests that the job be distributed across four processors.

```
hpc–demo% bsub –q hpc –n 4 hpc–job
```

Batch-mode is enabled by default, but can be disabled by the system administrator via the INTERACTIVE parameter.

You can check to see if a queue is able to handle batch-mode jobs by running bqueues -l *queue_name*. Then look in the SCHEDULING POLICIES: section of the bqueues output for the following entries.

- ONLY_INTERACTIVE – This entry means that batch mode is disabled; interactive and interactive batch modes are enabled.
- NO_INTERACTIVE – This entry means batch mode is enabled; interactive and interactive batch modes are disabled.
- No reference to INTERACTIVE – If there is no entry containing the term *XXX*_INTERACTIVE, all modes are enabled; this is the default condition.

The example queue shown in FIGURE A-1 on page 61 has a SCHEDULING POLICIES:setting of NO_INTERACTIVE, which allows batch-mode jobs, but not interactive batch.

As soon as hpc-job is submitted in batch mode, LSF Batch detaches it from the terminal session that submitted it.

Note – If you request more processors than are available, you must use *process wrapping* to allow multiple processes to be mapped to each processor. Otherwise, LSF Batch will wait indefinitely for the number of resources to become available and the job will never launched. Process wrapping is discussed in “Specify the Number of Processes” on page 65.

Submitting Interactive Batch Jobs

The 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.

The following example submits hpc-job to the queue named hpc in interactive batch mode. As before, this example is based on the assumption that hpc is configured to support parallel jobs.

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

The -I option specifies interactive batch mode.

The queue must not have interactive mode disabled. To check this, run

```
hpc-demo% bqueues -l hpc
```

and check the `SCHEDULING POLICIES:` section of the resulting output. If it contains either

```
SCHEDULING POLICIES:  ONLY_INTERACTIVE
```

or

```
SCHEDULING POLICIES:
```

(that is, no entry), interactive batch mode is enabled.

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 Chapter 7 of the *LSF Batch User's Guide*.

Using the `–sunhpc` Option

LSF Suite version 3.2.3 supports the `bsub` command-line option `–sunhpc`, which gives users special control over Sun MPI jobs. As mentioned earlier, the `–sunhpc` option and its arguments must be the last option on the `bsub` command line:

`bsub [`*basic_options*`] [–sunhpc` *sunhpc_args*`]` *job*

This section describes the arguments to the `–sunhpc` option.

Redirect `stderr`

Use the `–e` argument to redirect `stderr` to a file named *file*`.R`*n*, where *file* is the user-supplied name of the output file. The R*n* extension is supplied automatically and indicates the rank of the process producing the `stderr` output.

For example, to redirect `stderr` to files named `boston.R0`, `boston.R1`, and so forth, enter

```
hpc–demo% bsub –I –n 4 –q hpc –sunhpc –e boston hpc–job
```

Redirect `stdout`

Use the `–o` argument to redirect `stdout` to a file named *file*`.R`*n*, where *file* is the user-supplied name of the output file. The R*n* extension is supplied automatically and indicates the rank of the process producing the `stdout` output.

For example, to redirect `stdout` to files named `boston.R0`, `boston.R1`, and so forth, enter

```
hpc-demo% bsub -I -n 4 -q hpc -sunhpc -o boston hpc-job
```

Collocate Jobs by Specifying Job ID

Use the -j argument to specify the job ID of another job with which the new job should collocate.

For example, to cause job hpc-job to be collocated with a job whose job ID is 4622, enter

```
hpc-demo% bsub -I -n 4 -q hpc -sunhpc -j 4622 hpc-job
```

Use bjobs to find out the job ID of a job. See the *LSF Batch User's Guide* for details.

Collocate Jobs by Specifying Job Name

Use the -J argument to specify the name of another job with which the new job should collocate.

For example, to cause job hpc-job1 to be collocated with a job named hpc-job2, enter

```
hpc-demo% bsub -I -n 4 -q hpc -sunhpc -J hpc-job2 hpc-job1
```

Specify the Number of Processes

Use the -n argument to specify the number of processes to run. This argument can be used in concert with the bsub -n argument to cause process wrapping to occur. Process wrapping is the term used to describe a technique for distributing multiple processes to fewer processors than there are processes. As a result, each processor has multiple processes, which are spawned in a cyclical, wrap-around, fashion.

For example, the following will distribute 48 processes across 16 processors, resulting in a 3-process wrap per processor.

```
hpc-demo% bsub -I -n 16 -q hpc -sunhpc -n 48 hpc-job
```

If you specify a range of processors rather than a single quantity and a larger number of processes, the process wrapping ratio (number of processes per to processor) will depend on the number of processors that are actually allocated.

For example, the following will distribute 48 processes across at least 8 processors and possibly as many as 16.

```
hpc-demo% bsub -I -n 8,16 -q hpc -sunhpc -n 48 hpc-job
```

Consequently, the process-to-processor wrapping ratio may be as high as 6:1 (48 processes across 8 processors) or as low as 3:1 (48 processes across 16 processors).

Spawn a Job in the Stopped State

Use the `–s` argument to cause a job to be spawned in the `STOPPED` state. It does this by setting the `stop-on-exec` flag for the spawned process. This feature can be of value in a program monitoring or debugging tool as a way of gaining control over a parallel program. See the `proc`(4) man page for details.

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

The following example shows the `–s` argument being used to spawn an interactive batch job in the `STOPPED` state.

```
hpc–demo% bsub –I –n 1 –q hpc –sunhpc –s hpc–job
```

To identify processes in the `STOPPED` state, issue the `ps` command with the `–el` argument:

```
hpc–demo% 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
```

Here, the `sched` command is in the `STOPPED` state, as indicated by the `T` entry in the `S` (State) column.

Note that, when spawning a process 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.

Generate Rank-Tagged Output

Use the `–t` argument to cause all output to be tagged with its MPI rank.

Note – The `–t` argument *cannot* be used when output is redirected by the `–e` or `–o` options to `–sunhpc`.

For example, the following adds a rank-indicator prefix to each line of output.

```
 hpc-demo% bsub –I –n 4 –sunhpc –t uname –a
Job <10125> is submitted to default queue <hpc>.
<<Waiting for dispatch ...>>
<<Starting on hpc-demo3>>
R0: SunOS hpc-demo3 5.7 s998_16 sun4u sparc SUNW,Ultra-2
R1: SunOS hpc-demo3 5.7 s998_16 sun4u sparc SUNW,Ultra-2
R2: SunOS hpc-demo3 5.7 s998_16 sun4u sparc SUNW,Ultra-2
R3: SunOS hpc-demo3 5.7 s998_16 sun4u sparc SUNW,Ultra-2
 hpc-demo%
```

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 71 (for non-I/O MPI) and TABLE B-2 on page 73 (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.

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. 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 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 root, you can determine the current hard limit value via the `limit` function, as follows:

```
host# limit –h descriptors
```

If you are logged in to a Bourne shell as root, 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.

```
host# 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 see if 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 7 or 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 7 or Solaris 8 32-bit environment. However, this approach is not recommended. See “File Descriptors” on page 16 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:

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

the number of TCP ports may, under some conditions, 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 will only occur 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, such as SCI/RSM.

Index

SYMBOLS

– (dash), 18
&J, 40
&R, 40

A

–a, 48
a (file descriptor attribute), 38
–A (mpinfo), 50, 53
–A (mpps), 48
–A (mprun), 44
arg0, 44
arguments, specifying different vector, 44
attributes
  file descriptors, 38
  node (table), 53
  partitions (table), 52
  RRS, 30
    boolean, 33
    value-based, 31
authentication, 14

B

–B, 37, 42

C

–c, 23
–C (mpinfo), 55
–C (mprun), 43
clusters
  defined, 5
  single-node, 5
  specifying, 23
  to display, 43, 55
code, developing, 10
commands
  combining options, 10
  name conflicts, 18
  path, 10
Common Desktop Environment (CDE), 57
compilers, supported versions, 4
compiling, 10
conflicts, names, 18
Control–Z, 16
core files, 17

D

–D, 17, 37
debugging. *See* Prism development environment.
default_interactive_partition, 24, 55
defaults, specifying, 14
DES, 14
docs.sun.com, x
documentation, ix

LSF. *See* Load Sharing Facility documentation.

E

-e, 48
error classes (table), 71 to 72
  I/O, 73 to 74
error messages, 70
exit status, 44

F

-f, 48
Fatbrain.com, x
file descriptors
  attributes, 38
  strings, 37 to 41
    syntax, 39 to 41

G

-G, 43
group, changing for execution, 43

H

-h, 43

I

-I, 37
-i, 42
  and signals, 42
I/O
  mprun I/O, 36 to 42
  Sun MPI I/O, 2
  Sun PFS, 2
independent nodes. *See* nodes, independent.
information, 47 to 55
  mprun, 43

J

-J, 43, 49
-j, 26
job ID (jid), 7, 40, 47
  to display, 43
jobs, 7
  collocating, 26
  serial, 5, 44
  size limit, 5, 13
  states (table), 48
  status. *See* mpps.
  to display specific values, 49 to 50

K

Kerberos 5, 14
keylogin, 14
kill. *See* mpkill.
kinit, 14

L

l (file descriptor attribute), 38
linking, 10
-ln, 53
Load Sharing Facility (LSF), 59 to 67
  batch mode, 62 to 63
  bqueues, 61
  bsub, 62 to 67
  -e, 64, 66
  INTERACTIVE, 63
  interactive batch mode, 60, 63 to 64
  -J, 65
  -j, 65
  jobs
    collocating, 65
    stopped-state spawning, 66
  -l, 61
  LSF Base, 59
  LSF Batch, 59
  LSF Parallel, 59
  match mode, 60
  -n, 65
  NO_INTERACTIVE, 63
  -o, 64, 66

ONLY_INTERACTIVE, 63
pam, 61
path, 62
processes
specifying number, 65
wrapping, 63
queues, 60, 61
rank-tagged output, 66
-s
with Prism, 66
SCHEDULING POLICIES, 63
stderr, 64
stdout, 64
-sunhpc, 62, 64
-t, 61, 66
-v, 61
load-balancing, 7
logging in, 9
logging out, 11
-lp, 49, 52
LSF. *See* Load Sharing Facility.

M

man page path, 11
Message Passing Interface (MPI), 2
-Mf, 26 to 29
mpinfo, 50 to 55
mpkill, 45
mpps, 47 to 50
mprun
I/O
shell syntax, 41 to 42
shortcuts, 42
options
prioritization, 21 to 22
table, 19 to 20
to display, 43
mprun
omitting, 44
options, 19 to 44
MPRUN_FLAGS, 14, 21, 23
with Prism, 58

N

-N (mpinfo), 52
-n (mpinfo), 53
-N (mprun), 37
-n (mprun), 16, 37
NFS
with mprun I/O, 38
nodes
attributes (table), 53
defined, 5
domains as, 6
independent, 6
to display information, 52 to 55
-np, 24
with -Mf, 28
-Ns, 24
with RRS, 30

O

-o, 35 to 36, 42
OpenWindows, 57

P

-P, 49, 51
p (file descriptor attribute), 38
-p (mpinfo), 51
-p (mpps), 49
-p (mprun), 23
partitions
attributes (table), 52
defined, 6
disabled, 6
enabled, 6
specifying, 23
to display, 48
to display information, 51
PFS. *See* Sun Parallel File System.
Platform Computing Corporation, ix
Prism development environment, 3, 57 to 58
with bsub -s, 66
process ID (pid), 7
processes, 7

- controlling spawning, 24 to 36
- exceeding number of CPUs, 25
- grouping on a node, 29
- limiting one per node, 24
- mapping ranks to nodes, 26 to 29
  - using RRS (deprecated), 34 to 36
- number, 24
  - to display, 43
- ranks, 40
- running in background, 15
- sending signals, 45
- states (table), 48
- to display information, 49

processes, status. *See* `mpps`.

programs
- developing, 10
- name conflicts, 18
- serial. *See* jobs, serial.

`ps` *See* `mpps`.

pseudo-terminals (ptys), 38, 40, 42

R

`-R`, 29 to 36, 50, 51, 53

`r` (file descriptor attribute), 38

rankmap. *See* `-Mf`.

ranks
- mapping to nodes. *See* processes, mapping ranks to nodes.

resource requirement specifiers (RRS), 29 to 36, 50, 51, 53
- attributes, 30
- operators, 32

S

`-S`, 25

serial jobs. *See* jobs, serial.

shell-specific actions, 16

Solaris operating environment
- supported versions, 5

Starfire. *See* Sun Enterprise 10000.

`stderr`, 17, 36, 37, 42

`stdin`, 15, 36, 37, 42

`stdout`, 17, 36, 37, 42

Sun Cluster Runtime Environment (CRE)
- defined, 1
- fundamentals, 5 to 8
- path, 10

Sun Enterprise 10000, 6

Sun MPI, 2, 69
- error handlers, 69
- errors, 69 to 74

Sun Parallel File System (PFS), 2, 7

Sun S3L, 3

Sun WorkShop Compilers, 4

`SUNHPC_CLUSTER`, 23

`SUNHPC_PART`, 24

T

`t` (file descriptor attribute), 38

tiling. *See* `-Zt`.

U

`-U`, 43

`-u`, 25

user name, changing for execution, 43

V

`-V`, 43

`-v`, 51, 53

version of `mprun`, 43

W

`-W`, 25

`w` (file descriptor attribute), 38

warning messages, 70

working directory, to change, 43

Z

`-Z`, `-Zt`, 29