Sun MPI 4.0 User's Guide: With LSF

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

Part No: 805-7230-10
June 1999, Revision A

Copyright Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers .

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

Sun, Sun Microsystems, the Sun logo, SunStore, AnswerBook2, docs.sun.com, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

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

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

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

Copyright 1999 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303-4900 U.S.A. Tous droits réservés.

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

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

Sun, Sun Microsystems, le logo Sun, SunStore, AnswerBook2, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

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

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Contents

Preface

The *Sun MPI 4.0 User's Guide: With LSF* explains how to execute Sun MPI jobs on systems running Sun HPC ClusterTools 3.0 software with the LSF Suite, version 3.2.3, from Platform Computing Corporation. It is intended to be used in conjunction with the *LSF Batch User's Guide*.

Note - If your cluster uses the Sun Cluster Runtime Environment (CRE) *instead of* the LSF 3.2.3 workload management suite, read the *Sun MPI 4.0 User's Guide: With CRE* instead of this manual.

Before You Read This Book

This book supplements the *LSF Batch User's Guide*, version 3.2.3. For information about writing MPI programs, refer to the *Sun MPI 4.0 Programming and Reference Guide*. Sun MPI 4.0 is part of the Sun HPC ClusterTools 3.0 suite of software. Product notes for Sun MPI are included in *Sun HPC ClusterTools 3.0 Product Notes*.

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:

- AnswerBook™ online documentation for the Solaris™ 2.6 or Solaris 7 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*#

TABLE P–2 Shell Prompts *(continued)*

Shell	Prompt
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 3.0 Product Notes*	805-6262-10
Sun HPC ClusterTools software	*Sun HPC ClusterTools 3.0 Administrator's Guide: With LSF*	805-6280-10
SCI	*Sun HPC 3.0 SCI Guide*	805-6263-10
Installing Sun HPC ClusterTools software	*Sun HPC ClusterTools 3.0 Installation Guide*	805-6264-10
Installing LSF Suite	*LSF 3.2.3 Installation Guide*	805-6265-10
Sun MPI Programming	*Sun MPI 4.0 Programming and Reference Guide*	805-6269-10
Prism	*Prism 6.0 User's Guide*	805-6277-10
Prism	*Prism 6.0 Reference Manual*	805-6278-10
Sun S3L	*Sun S3L 3.0 Programming and Reference Guide*	805-6275-10

TABLE P–3 Related Documentation *(continued)*

Application	Title	Part Number
LSF Suite	*LSF Batch Administrator's Guide*	805-6257-10
LSF Suite	*LSF Batch User's Guide*	805-6258-10
LSF Suite	*LSF Parallel User's Guide*	805-6259-10
LSF Suite	*LSF Batch Programmer's Guide*	805-6260-10

Sun Documentation on the Web

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 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 3.0 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.

Sun HPC Cluster Overview

A Sun HPC 3.0 cluster can be a single Sun SMP (symmetric multiprocessor) server or a cluster of these SMPs running Sun HPC ClusterTools 3.0 software and LSF Base, Batch, and Parallel software.

Sun HPC Cluster Hardware

A Sun HPC cluster configuration can range from a single Sun SMP (symmetric multiprocessor) server to a cluster of SMPs connected by any Sun-supported, TCP/IP-capable interconnect.

Note - An individual SMP server within a Sun HPC cluster is referred to as a *node.*

Sun HPC ClusterTools 3.0 Software and LSF Suite 3.2.3

Sun HPC ClusterTools 3.0 software is an integrated ensemble of parallel development tools that extend Sun's network computing solutions to high-end distributed-memory applications. The Sun HPC ClusterTools products are teamed with LSF Suite 3.2.3, Platform Computing Corporation's resource management software.

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

Load Sharing Facility

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 3.0 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.

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 LSF Suite 3.2.3.
- Support for multithreaded programming.
- Seamless use of different network protocols; for example, code compiled on a Sun HPC System that has a Scalable Coherent Interface (SCI) network, can be run without change on a system 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).
- Prism support – Users can develop, run, and debug programs in the Prism programming environment.
- MPI I/O support for file I/O.
- Sun MPI is a dynamic library.

Sun MPI and MPI I/O provide full F77, C, and C++ support and Basic F90 support.

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 message-passing programs. With Prism you can

- Control program execution, such as:
  - Start and stop execution.
  - Set breakpoints and traces.
  - Print values of variables and expressions.
  - Display the call stack.
- Visualize data in various formats.
- Analyze performance of MPI programs.
- Control entire multiprocess parallel jobs, aggregating processes 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 MPI and provides the following functionality for Sun 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 3.0 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

Job Execution Modes

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, the user can specify that the job be placed on that queue. If a job does not require special execution conditions, the user 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.

CHAPTER 2

Starting Sun MPI Programs

This chapter 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.
- Using `bsub` (overview).
- Submitting an MPI job in batch mode.
- Submitting an MPI job in interactive batch mode.
- Using the Sun HPC option `--sunhpc`.

For information about developing, compiling, and linking Sun MPI programs, see the *Sun MPI 4.0 Programming and Reference Manual*.

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 2–1.

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

which means it is able to handle parallel (distributed MPI) jobs. In the example shown in Figure 2–1, 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
```

Figure 2–1 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 9 and “Submitting Interactive Batch Jobs” on page 10.

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 9 and "Submitting Interactive Batch Jobs" on page 10 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 11.

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 2–1; 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 13.

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
```

"Redirect `stderr`" on page 11 through "Spawn a Job in the Stopped State" on page 13 describe the `--sunhpc` arguments.

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 would be likely to interfere with Prism'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%
```

CHAPTER 3

Performance Tuning

Sun MPI uses a variety of techniques to deliver high-performance, robust, and memory-efficient message passing under a wide set of circumstances. In certain situations, however, applications will benefit from nondefault behaviors. The Sun MPI environment variables discussed in this section allow you to tune these default behaviors. A list of all Sun MPI environment variables, with brief descriptions, can be found in Appendix A and in the `MPI` man page.

Current Settings

User tuning of MPI environment variables can be restricted by the system administrator through a configuration file, `hpc.conf`. To determine whether such resctrictions are in place on your local cluster use the `MPI_PRINTENV` (described below) to verify settings. .

In most cases, performance will be good without tuning any environment variables. Nevertheless, here are some performance guidelines for using MPI environment variables. In some cases, diagnosis of whether environment variables would be helpful is aided by Prism profiling with TNF probes, as described in the *Prism User's Guide* and the *Sun MPI Programming and Reference Guide.*

Runtime Diagnostic Information

Certain Sun MPI environment variables cause extra diagnostic information to be printed out at run time:

```
% setenv MPI_PRINTENV 1
% setenv MPI_SHOW_INTERFACES 3
% setenv MPI_SHOW_ERRORS 1
```

Running on a Dedicated System

If your system has sufficient capacity for running your MPI job, you can commit processors aggressively to your job. At a minimum, the CPU load should not exceed the number of physical processors. The CPU load for your job is the number of MPI processes in the job, but the load is greater if your job is multithreaded. The load on the system must also be shared with any other jobs are running on the same system. You can check the current load can be checked with the `lsload` command.

To run your job more aggressively on a dedicated system, set the `MPI_SPIN` and `MPI_PROCBIND` environment variables:

```
% setenv MPI_SPIN 1
```

Use this only if you will leave at least one processor per node free to service system daemons. Profiling with Prism introduces background daemons that cause a slight but noticeable load, so you must be careful to avoid overloading when attempting to profile a code with this setting.

```
% setenv MPI_PROCBIND 1
```

Set the `MPI_PROCBIND` variable only if there are no other MPI jobs running and your job is single-threaded.

Safe Use of System Buffers

In some MPI programs, processes send large volumes of data with blocking sends before starting to receive messages. The MPI standard specifies that users must explicitly provide buffering in such cases, perhaps using `MPI_Bsend` calls. In practice, however, some users rely on the standard send routine (`MPI_Send`) to supply unlimited buffering. By default, Sun MPI prevents deadlock in such

situations through general polling, which drains system buffers even when no receives have been posted by the user code.

For best performance on typical, safe programs, you can suppress general polling should by setting `MPI_POLLALL`:

```
% setenv MPI_POLLALL 0
```

Trading Memory for Performance

Depending on message traffic, performance can stall if system buffers become congested, but it can be superior if buffers are large. Here, we examine performance for on-node messages via shared-memory buffers.

It is helpful to think of data traffic per connection, the “path” from a particular sender to a particular receiver, since many Sun MPI buffering resources are allocated on a per-connection basis. A sender may emit bursts of messages on a connection, during which time the corresponding receiver may not be depleting the buffers. For example, a sender may execute a sequence of send operations to one receiver during a period in which that receiver is not making any MPI calls whatsoever.

You may need to use profiling to diagnose such conditions. For more information on profiling, see the *Prism User’s Guide* and the *Sun MPI Programming and Reference Guide*.

Rendezvous or Eager Protocol?

Is your program sending many long, unexpected messages? Sun MPI offers message *rendezvous*, which requires a receiver to echo a ready signal to the sender before data transmission can begin. This can improve performance for the case of a pair of processes that communicate with a different order for their sends as for their receives, since receive-side buffering would be reduced. To allow rendezvous behavior for long messages, set the `MPI_EAGERONLY` environment variable:

```
% setenv MPI_EAGERONLY 0
```

The threshold message size for rendezvous behavior can be tuned independently for each protocol with `MPI_SHM_RENDVSIZE`, `MPI_TCP_RENDVSIZE`, and `MPI_RSM_RENDVSIZE`.

Note - Rendezvous will often degrade performance by coupling senders to receivers. Also, for some “unsafe” codes, it can produce deadlock.

Many Broadcasts or Reductions

Does your program include many broadcasts or reductions on large messages? Large broadcasts may benefit from increased values of `MPI_SHM_BCASTSIZE`, and large reductions from increased `MPI_SHM_REDUCESIZE`. Also, if many different communicators are involved, you may want to increase `MPI_SHM_GBPOOLSIZE`. In most cases, the default values will provide best performance.

Shared-Memory Point-to-Point Message Passing

The size of each shared-memory buffer is fixed at 1 Kbyte. Most other quantities in shared-memory message passing are settable with MPI environment variables.

A *short* message, at most `MPI_SHM_SHORTMSGSIZE` bytes long, is fit into one postbox and no buffers are used. Above that size, message data is written into buffers and controlled by postboxes.

Only starting at `MPI_SHM_PIPESTART` bytes, however, are multiple postboxes used, which is known as *pipelining*. The amount of buffer data controlled by any one postbox is at most `MPI_SHM_PIPESIZE` bytes. By default, `MPI_SHM_PIPESTART` is well below `MPI_SHM_PIPESIZE`. For the smallest pipelined messages, then, a message is broken roughly into two, and each of two postboxes controls roughly half the message.

Above `MPI_SHM_CYCLESTART` bytes, messages are fed cyclically through two sets of buffers, each set of size `MPI_SHM_CYCLESIZE` bytes. During a cyclic transfer, the footprint of the message in shared memory buffers is 2*`MPI_SHM_CYCLESIZE` bytes.

The postbox area consists of `MPI_SHM_NUMPOSTBOX` postboxes per connection. By default, each connection has its own pool of buffers, each pool of size `MPI_SHM_CPOOLSIZE` bytes.

By setting `MPI_SHM_SBPOOLSIZE`, users may specify that each sender has a pool of buffers, of `MPI_SHM_SBPOOLSIZE` bytes each, to be shared among its various connections. If `MPI_SHM_CPOOLSIZE` is also set, then any one connection may consume only that many bytes from its send-buffer pool at any one time.

Memory Considerations

In all, the size of the shared-memory area devoted to point-to-point messages is

```
n * ( n -- 1
)
  * (
      MPI_SHM_NUMPOSTBOX *
( 64 + MPI_SHM_SHORTMSGSIZE )
    + MPI_SHM_CPOOLSIZE
    )
```

bytes when per-connection pools are used (that is, when `MPI_SHM_SBPOOLSIZE` is not set) and

```
n * ( n -- 1
) * MPI_SHM_NUMPOSTBOX *
( 64 + MPI_SHM_SHORTMSGSIZE )
+
n * MPI_SHM_SBPOOLSIZE
```

bytes when per-sender pools are used (that is, when `MPI_SHM_SBPOOLSIZE` is set).

Cyclic message passing limits the size of shared memory that is needed to transfer even arbitrarily large messages.

Shared-Memory Collectives

Collective operations in Sun MPI are highly optimized and make use of a “general buffer pool” within shared memory.

`MPI_SHM_GBPOOLSIZE` sets the amount of space available on a node for the “optimized” collectives in bytes. By default, it is set to 20971520 bytes. This space is used by `MPI_Bcast`, `MPI_Reduce`, `MPI_Allreduce`, `MPI_Reduce_scatter`, and `MPI_Barrier`, provided that two or more of the MPI processes are on the node.

When a communicator is created, space is reserved in the general buffer pool for performing barriers, short broadcasts, and a few other purposes.

For larger broadcasts, shared memory is allocated out of the general buffer pool. The maximum buffer-memory footprint in bytes of a broadcast operation is set by an environment variable as

(*n* / 4) * 2 * `MPI_SHM_BCASTSIZE`

where *n* is the number of MPI processes on the node. If less memory is needed than this, then less memory is used. After the broadcast operation, the memory is returned to the general buffer pool.

For reduce operations,

n * *n* * MPI_SHM_REDUCESIZE

bytes are borrowed from the general buffer pool.

The broadcast and reduce operations are pipelined for very large messages. By increasing MPI_SHM_BCASTSIZE and MPI_SHM_REDUCESIZE, one can improve the efficiency of these collective operations for very large messages, but the amount of time it takes to fill the pipeline can also increase.

If MPI_SHM_GBPOOLSIZE proves to be too small and a collective operation happens to be unable to borrow memory from this pool, the operation will revert to slower algorithms. Hence, under certain circumstances, performance could dictate increasing MPI_SHM_GBPOOLSIZE.

Running over TCP

TCP ensures reliable dataflow, even over lossy networks, by retransmitting data as necessary. When the underlying network loses a lot of data, the rate of retransmission can be very high and delivered MPI performance will suffer accordingly. Increasing synchronization between senders and receivers by lowering the TCP rendezvous threshold with MPI_TCP_RENDVSIZE may help in certain cases. Generally, increased synchronization will hurt performance, but over a lossy network it may help mitigate catastrophic degradation.

If the network is not lossy, then lowering the rendezvous threshold would be counterproductive and, indeed, a Sun MPI safeguard may be lifted. For reliable networks, use

```
% setenv MPI_TCPSAFEGATHER 0
```

Remote Shared Memory (RSM) Point-to-Point Message Passing

The RSM protocol has some similarities with the shared memory protocol, but it also has substantial deviations, and environment variables are used differently.

The maximum size of a short message is MPI_RSM_SHORTMSGSIZE bytes, with default value of 401 bytes. Short RSM messages can span multiple postboxes, but they still do not use any buffers.

The most data that will be sent under any one postbox for pipelined messages is MPI_RSM_PIPESIZE bytes. There are MPI_RSM_NUMPOSTBOX postboxes for each RSM connection.

If MPI_RSM_SBPOOLSIZE is unset, then each RSM connection has a buffer pool of MPI_RSM_CPOOLSIZE bytes. If MPI_RSM_SBPOOLSIZE is set, then each process has a pool of buffers that is MPI_RSM_SBPOOLSIZE bytes per remote node for sending messages to processes on the remote node.

Unlike the case of the shared-memory protocol, values of the MPI_RSM_PIPESIZE, MPI_RSM_CPOOLSIZE, and MPI_RSM_SBPOOLSIZE environment variables are merely requests. Values set with the setenv or printed when MPI_PRINTENV is used may not reflect effective values. In particular, only when connections are actually established are the RSM parameters truly set. Indeed, the effective values could change over the course of program execution if lazy connections are employed.

Striping refers to passing messages over multiple links to get the speedup of their aggregate bandwidth. The number of stripes used is MPI_RSM_MAXSTRIPE or all physically available stripes, whichever is less.

Use of rendezvous for RSM messages is controlled with MPI_RSM_RENDVSIZE.

Memory Considerations

Memory is allocated on a node for each remote MPI process that sends messages to it over RSM. If np_local is the number of processes on a particular node, then the memory requirement on the node for RSM message passing from any one remote process is

np_local * (MPI_RSM_NUMPOSTBOX * 128 + MPI_RSM_CPOOLSIZE)

bytes when MPI_RSM_SBPOOLSIZE is unset, and

np_local * *MPI_RSM_NUMPOSTBOX* * 128 + MPI_RSM_SBPOOLSIZE

bytes when MPI_RSM_SBPOOLSIZE is set.

The amount of memory actually allocated may be higher or lower than this requirement:

- The memory requirement is rounded up to some multiple of 8192 bytes with a minimum of 32768 bytes.
- This memory is allocated from a 256-Kbyte (262,144-byte) segment.
  - If the memory requirement is greater than 256 Kbytes, then insufficient memory will be allocated.
  - If the memory requirement is less than 256 Kbytes, some allocated memory will go unused. (There is some, but only limited, sharing of segments.)

If less memory is allocated than is required, then requested values of `MPI_RSM_CPOOLSIZE` or `MPI_RSM_SBPOOLSIZE` may be reduced at run time. This can cause the requested value of `MPI_RSM_PIPESIZE` to be overridden as well.

Each remote MPI process requires its own allocation on the node as described above.

If multiple stripes are employed, the memory requirement increases correspondingly.

Performance Considerations

The pipe size should be at most half as big as the connection pool

2 * `MPI_RSM_PIPESIZE` <= `MPI_RSM_CPOOLSIZE`

Otherwise, pipelined transfers will proceed slowly. The library adjusts `MPI_RSM_PIPESIZE` appropriately.

Reducing striping has no performance advantage, but varying `MPI_RSM_MAXSTRIPE` can give you insight into the relationship between application performance depends and internode bandwidth.

For pipelined messages, a sender must synchronize with its receiver to ensure that remote writes to buffers have completed before postboxes are written. Long pipelined messages can absorb this synchronization cost, but performance for short pipelined messages will suffer. In some cases, raising `MPI_RSM_SHORTMSGSIZE` can mitigate this effect.

APPENDIX **A**

Environment Variables

Many environment variables are available for fine-tuning your Sun MPI environment. All 39 Sun MPI environment variables are listed here with brief descriptions. The same descriptions are also available on the `MPI` man page. If you want to return to the default setting after having set a variable, simply unset it (using `unsetenv`). The effects of the variables are explained in more detail in Chapter 3.

The environment variables are listed here in six groups:

Informational

MPI_PRINTENV

When set to 1, causes the environment variables and `hpc.conf` parameters associated with the MPI job to be printed out. The default is 0.

MPI_QUIET

If set to 1, suppresses Sun MPI warning messages. The default value is 0.

MPI_SHOW_ERRORS

If set to 1, the `MPI_ERRORS_RETURN` error handler prints the error message and returns the error. The default value is 0.

MPI_SHOW_INTERFACES

When set to 1, 2 or 3, information regarding which interfaces are being used by an MPI application prints to `stdout`. Set `MPI_SHOW_INTERFACES` to 1 to print the selected internode interface. Set it to 2 to print all the interfaces and their rankings. Set it to 3 for verbose output. The default value, 0, does not print information to `stdout`.

General Performance Tuning

MPI_POLLALL

When set to 1, the default value, all connections are polled for receives, also known as full polling. When set to 0, only those connections are polled where receives are posted. Full polling helps drain system buffers and so lessen the chance of deadlock for “unsafe” codes. Well-written codes should set `MPI_POLLALL` to 0 for best performance.

MPI_PROCBIND

Binds each MPI process to its own processor. By default, `MPI_PROCBIND` is set to 0, which means processor binding is off. To turn processor binding on, set it to 1. The system administrator may allow or disable processor binding by setting the `pbind` parameter in the `hpc.conf` file on or off. If this parameter is set, the `MPI_PROCBIND` environment variable is disabled. Performance can be enhanced with processor binding, but very poor performance will result if processor binding is used for multithreaded jobs or for more than one job at a time.

MPI_SPIN

Sets the spin policy. The default value is 0, which causes MPI processes to spin nonaggressively, allowing best performance when the load is at least as great as the number of CPUs. A value of 1 causes MPI processes to spin aggressively, leading to

best performance if extra CPUs are available on each node to handle system daemons and other background activities.

Tuning Memory for Point-to-Point Performance

MPI_RSM_CPOOLSIZE

The requested size, in bytes, to be allocated per stripe for buffers for each remote-shared-memory connection. This value may be overridden when connections are established. The default value is 16384 bytes.

MPI_RSM_NUMPOSTBOX

The number of postboxes per stripe per remote-shared-memory connection. The default is 15 postboxes.

MPI_RSM_PIPESIZE

The limit on the size (in bytes) of a message that can be sent over remote shared memory via the buffer list of one postbox per stripe. The default is 8192 bytes.

MPI_RSM_SBPOOLSIZE

If set, `MPI_RSM_SBPOOLSIZE` is the requested size in bytes of each RSM send buffer pool. An RSM send buffer pool is the pool of buffers on a node that a remote process would use to send to processes on the node. A multiple of 1024 must be used. If unset, then pools of buffers are dedicated to connections rather than to senders.

MPI_RSM_SHORTMSGSIZE

The maximum size, in bytes, of a message that will be sent via remote shared memory without using buffers. The default value is 401 bytes.

MPI_SHM_CPOOLSIZE

The amount of memory, in bytes, that can be allocated to each connection pool. When `MPI_SHM_SBPOOLSIZE` is not set, the default value is 24576 bytes. Otherwise, the default value is `MPI_SHM_SBPOOLSIZE`.

MPI_SHM_CYCLESIZE

The limit, in bytes, on the portion of a shared-memory message that will be sent via the buffer list of a single postbox during a cyclic transfer. The default value is 8192 bytes. A multiple of 1024 that is at most `MPI_SHM_CPOOLSIZE`/2 must be used.

MPI_SHM_CYCLESTART

Shared-memory transfers that are larger than `MPI_SHM_CYCLESTART` bytes will be cyclic. The default value is 24576 bytes.

MPI_SHM_NUMPOSTBOX

The number of postboxes dedicated to each shared-memory connection. The default value is 16.

MPI_SHM_PIPESIZE

The limit, in bytes, on the portion of a shared-memory message that will be sent via the buffer list of a single postbox during a pipeline transfer. The default value is 8192 bytes. The value must be a multiple of 1024.

MPI_SHM_PIPESTART

The size, in bytes, at which shared-memory transfers will start to be pipelined. The default value is 2048. Multiples of 1024 must be used.

MPI_SHM_SBPOOLSIZE

If set, `MPI_SHM_SBPOOLSIZE` is the size, in bytes, of the pool of shared-memory buffers dedicated to each sender. A multiple of 1024 must be used. If unset, then pools of shared-memory buffers are dedicated to connections rather than to senders.

MPI_SHM_SHORTMSGSIZE

The size (in bytes) of the section of a postbox that contains either data or a buffer list. The default value is 256 bytes.

Note - If MPI_SHM_PIPESTART, MPI_SHM_PIPESIZE, or MPI_SHM_CYCLESIZE is increased to a size larger than 31744 bytes, then MPI_SHM_SHORTMSGSIZE may also have to be increased. See Chapter 3 for more information.

Numerics

MPI_CANONREDUCE

Prevents reduction operations from using any optimizations that take advantage of the physical location of processors. This may provide more consistent results in the case of floating-point addition, for example. However, the operation may take longer to complete. The default value is 0, meaning optimizations are allowed. To prevent optimizations, set the value to 1.

Tuning Rendezvous

MPI_EAGERONLY

When set to 1, the default, only the eager protocol is used. When set to 0, both eager and rendez-vous protocols are used.

MPI_RSM_RENDVSIZE

Messages communicated by remote shared memory that are greater than this size will use the rendezvous protocol unless the environment variable MPI_EAGERONLY is set to 1. Default value is 16384 bytes.

MPI_SHM_RENDVSIZE

Messages communicated by shared memory that are greater than this size will use the rendezvous protocol unless the environment variable `MPI_EAGERONLY` is set. The default value is 24576 bytes.

MPI_TCP_RENDVSIZE

Messages communicated by TCP that contain data of this size and greater will use the rendezvous protocol unless the environment variable `MPI_EAGERONLY` is set. Default value is 49152 bytes.

Miscellaneous

MPI_COSCHED

Specifies the user's preference regarding use of the `spind` daemon for coscheduling. Values can be 0 (prefer no use) or 1 (prefer use). This preference may be overridden by the system administrator's policy. This policy is set in the `hpc.conf` file and can be 0 (forbid use), 1 (require use), or 2 (no policy). If no policy is set and no user preference is specified, coscheduling is not used.

Note - If no user preference is specified, the value 2 will be shown when environment variables are printed with `MPI_PRINTENV`.

MPI_FLOWCONTROL

Limits the number of unexpected messages that can be queued from a particular connection. Once this quantity of unexpected messages has been received, polling the connection for incoming messages stops. The default value, 0, indicates that no limit is set. To limit flow, set the value to some integer greater than zero.

MPI_FULLCONNINIT

Ensures that all connections are established during initialization. By default, connections are established lazily. However, you can override this default by setting the environment variable `MPI_FULLCONNINIT` to 1, forcing full-connection initialization mode. The default value is 0.

MPI_MAXFHANDLES

The maximum number of Fortran handles for objects other than requests. `MPI_MAXFHANDLES` specifies the upper limit on the number of concurrently allocated Fortran handles for MPI objects other than requests. This variable is ignored in the default 32-bit library. The default value is 1024. Users should take care to free MPI objects that are no longer in use. There is no limit on handle allocation for C codes.

MPI_MAXREQHANDLES

The maximum number of Fortran request handles. `MPI_MAXREQHANDLES` specifies the upper limit on the number of concurrently allocated MPI request handles. Users must take care to free up request handles by properly completing requests. The default value is 1024. This variable is ignored in the default 32-bit library.

MPI_OPTCOLL

The MPI collectives are implemented using a variety of optimizations. Some of these optimizations can inhibit performance of point-to-point messages for “unsafe” programs. By default, this variable is 1, and optimized collectives are used. The optimizations can be turned off by setting the value to 0.

MPI_RSM_MAXSTRIPE

Defines the maximum number of stripes that can be used during communication via remote shared memory. The default value is the number of stripes in the cluster, with a maximum default of 2.

MPI_SHM_BCASTSIZE

On SMPs, the implementation of `MPI_Bcast()` for large messages is done using a double-buffering scheme. The size of each buffer (in bytes) is settable by using this environment variable. The default value is 32768 bytes.

MPI_SHM_GBPOOLSIZE

The amount of memory available, in bytes, to the general buffer pool for use by collective operations. The default value is 20971520 bytes.

MPI_SHM_REDUCESIZE

On SMPs, calling `MPI_Reduce()` causes all processors to participate in the reduce. Each processor will work on a piece of data equal to the `MPI_SHM_REDUCESIZE` setting. The default value is 256 bytes. Care must be taken when setting this variable because the system reserves `MPI_SHM_REDUCESIZE` * *np* * *np* memory to execute the reduce.

MPI_SPINDTIMEOUT

When coscheduling is enabled, limits the length of time (in milliseconds) a message will remain in the poll waiting for the `spind` daemon to return. If the timeout occurs before the daemon finds any messages, the process re-enters the polling loop. The default value is 1000 ms. A default can also be set by a system administrator in the `hpc.conf` file.

MPI_TCP_CONNLOOP

Sets the number of times `MPI_TCP_CONNTIMEOUT` occurs before signaling an error. The default value for this variable is 0, meaning that the program will abort on the first occurrence of `MPI_TCP_CONNTIMEOUT`.

MPI_TCP_CONNTIMEOUT

Sets the timeout value in seconds that is used for an `accept()` call. The default value for this variable is 600 seconds (10 minutes). This timeout can be triggered in both full- and lazy-connection initialization. After the timeout is reached, a warning message will be printed. If `MPI_TCP_CONNLOOP` is set to 0, then the first timeout will cause the program to abort.

MPI_TCP_SAFEGATHER

Allows use of a congestion-avoidance algorithm for `MPI_Gather()` and `MPI_Gatherv()` over TCP. By default, `MPI_TCP_SAFEGATHER` is set to 1, which means use of this algorithm is on. If you know that your underlying network can handle gathering large amounts of data on a single node, you may want to override this algorithm by setting `MPI_TCP_SAFEGATHER` to 0.

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 (for non-I/O MPI) and Table B–2 (for MPI I/O) and are also available on the `MPI` man page.

Three predefined error handlers are available in Sun MPI 4.0:

- `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.

TABLE B–1 Sun MPI Standard Error Classes *(continued)*

Error Code	Value	Meaning
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.
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_LASTCODE	46	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`.

TABLE B–2 Sun MPI I/O Error Classes *(continued)*

Error Class	Value	Meaning
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.
MPI_ERR_INFO_NOKEY	39	No such key .
MPI_ERR_INFO_VALUE	40	Illegal info value.
MPI_ERR_LASTCODE	46	Last error code.