Contents

Preface xi

1. Introduction to Sun HPC ClusterTools Software 1
   Supported Configurations 1
   Open Runtime Environment (ORTE) 2
      Executing Programs With mpirun 2
   Integration With Distributed Resource Management Systems 2
   Open MPI Features 3
   Debugging With TotalView 3

2. Fundamental Concepts 5
   Clusters and Nodes 5
   Processes 6
      How Programs Are Launched 6
   How the Open MPI Environment Is Integrated With Distributed Resource Management Systems 7
      Using Sun Grid Engine With ORTE 7
      Submitting Jobs Under Sun Grid Engine Integration 8
   MCA Parameters 8
   How ORTE Works With Zones in the Solaris 10 Operating System 8
3. Before You Begin 11

   Prerequisites 11
   Command and Man Page Paths 11
   Setting Up Your Path 12

   ▼ To Set Up Your Path for the Solaris OS and the Sun Studio Compiled Linux Version 12
   ▼ To Set Up Your Path for the GNU Compiled Linux Version 13

   Core Files 13
   Setting Up a known_hosts File 13

4. Compiling MPI Programs 15

   Supported Compilers 15
   Using the Wrapper Compilers 16
   Using Non-Default Error Handlers 16
   Compiling Fortran 90 Programs 17

5. Running Programs With the mpirun Command 19

   About the mpirun Command 19
   Syntax for the mpirun Command 20

   mpirun Options 20

   Using Environment Variables With the mpirun Command 21
   Using MCA Parameters With the mpirun Command 21
   Canceling Send and Receive Operations 22

   mpirun Command Examples 22

   ▼ To Run a Program With Default Settings 23
   ▼ To Run Multiple Processes 23
   ▼ To Direct mpirun By Using an Appfile 23

   Mapping MPI Processes to Nodes 24
   Specifying Available Hosts 24
   Specifying Hosts By Using a Hostfile 25
Specifying Hosts By Using the --host Option 26
  ▼ To Specify Multiple Slots Using the --host Option 26
Excluding Hosts From Scheduling By Using the --host Option 26
Oversubscribing Nodes 27
Scheduling Policies 27
  Scheduling By Slot 27
    ▼ To Specify By-Slot Scheduling 28
  Scheduling By Node 29
    ▼ To Specify By-Node Scheduling 29
Comparing By-Slot to By-Node Scheduling 30
Controlling Input/Output 30
  ▼ To Redirect Standard I/O 31
Controlling Other Job Attributes 31
    ▼ To Change the Working Directory 31
    ▼ To Specify Debugging Output 32
    ▼ To Display Command Help (--h) 32
Submitting Jobs Under Sun Grid Engine Integration 35
  Defining Parallel Environment (PE) and Queue 35
    ▼ To Use PE Commands 36
    ▼ To Use Queue Commands 37
Submitting Jobs in Interactive Mode 37
  ▼ To Set the Interactive Display 37
  ▼ To Submit Jobs Interactively 37
  ▼ To Verify That Sun Grid Engine Is Running 38
  ▼ To Start an Interactive Session Using qrsh 39
Using MPI Client/Server Applications 39
  ▼ To Launch the Client/Server Job 39
Using Name Publishing 40
6. Running Programs With \texttt{mpirun} in Distributed Resource Management Systems 47

\texttt{mpirun} Options for Third-Party Resource Manager Integration 47

Checking Your Open MPI Configuration 48
\begin{itemize}
  \item To Check for \texttt{rsh/ssh} 48
  \item To Check for PBS/Torque 48
  \item To Check for Sun Grid Engine 48
\end{itemize}

Running Parallel Jobs in the PBS Environment 48
\begin{itemize}
  \item To Run an Interactive Job in PBS 49
  \item To Run a Batch Job in PBS 50
\end{itemize}

Running Parallel Jobs in the Sun Grid Engine Environment 51

Defining Parallel Environment (PE) and Queue 52
\begin{itemize}
  \item To Use PE Commands 52
  \item To Use Queue Commands 53
\end{itemize}

Submitting Jobs Under Sun Grid Engine Integration 53
\begin{itemize}
  \item To Set the Interactive Display 53
  \item To Submit Jobs in Batch Mode 54
  \item To See a Running Job 55
  \item To Delete a Running Job 55
\end{itemize}

\texttt{rsh} Limitations 55

Using \texttt{rsh} as the Job Launcher 56
Using Sun Grid Engine as the Job Launcher 56

For More Information 57

7. Using MCA Parameters With \texttt{mpirun} 59

About the Modular Component Architecture 60
Checking the _mpirun_ Privileges 86
  ▼ To Determine the Correct Privileges on the Cluster 86

Running DTrace with MPI Programs 87
  Running an MPI Program Under DTrace 88
    ▼ To Trace a Program Using the _mpitrace.d_ Script 88
    ▼ To Trace a Parallel Program and Get Separate Trace Files 88
  Attaching DTrace to a Running MPI Program 89
    ▼ To Attach DTrace to a Running MPI Program 89

Simple MPI Tracing 89

Tracking Down Resource Leaks 91
Using the DTrace _mpiperuse_ Provider 96
  DTrace Support in the ClusterTools Software 96
  Available _mpiperuse_ Probes 96
    Specifying an _mpiperuse_ Probe in a D Script 97
  Available Arguments 98
  How To Use _mpiperuse_ Probes to See Message Queues 98
    _mpiperuse_ Usage Examples 100
      ▼ To Count the Number of Messages To or From a Host 100
      ▼ To Count the Number of Messages To or From Specific BTLs 100
      ▼ To Obtain Distribution Plots of Message Sizes Sent or Received From a Host 101
      ▼ To Create Distribution Plots of Message Sizes By Communicator, Rank, and Send/Receive 101

A. Troubleshooting 103
  MPI Messages 103
    Standard Error Classes 103
  MPI I/O Error Handling 105
    Exceeding the File Descriptor Limit 107
      Increasing the Number of Available File Descriptors 107
Contents

▼ To View the Hard Limit from the C Shell 107
▼ To View the Hard Limit from the Bourne Shell 108
▼ To Increase the Number of File Descriptors 108

Setting File Descriptor Limits When Using Sun Grid Engine 109

B. List of Available MCA Parameters 111

Index 135
Preface

This manual explains how to use distributed resource management packages for effective resource management and utilization accounting. The following packages work in conjunction with the Open Message-Passing Interface (Open MPI) parallel applications:

- Sun Grid Engine Version 6.1 software
- Altair PBS Professional 9.2 or Torque 2.3

Before You Read This Book

The Sun HPC ClusterTools™ 8.1 Software Release Notes includes release note information for the other components in this suite. For information about writing MPI programs, refer to the Open MPI Software Programming and Reference Guide. For information about a specific distributed resource management package, refer to the documentation supplied with that package.

For more information about Open MPI and its components, see the Open MPI website at:

http://www.open-mpi.org

Using UNIX Commands

This document might not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.
See one or more of the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at
  
  http://www.sun.com/documentation
Typographic Conventions

AaBbCc123  The names of commands, files, and directories; on-screen computer output

AaBbCc123  What you type, when contrasted with on-screen computer output

AaBbCc123  Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories; on-screen computer output</td>
<td>Edit your .login file.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, when contrasted with on-screen computer output</td>
<td>Use ls -a to list all files.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new words or terms, words to be emphasized. Replae command-line variables with real names or values.</td>
<td>% su</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>% su</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>% su</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password:</td>
</tr>
</tbody>
</table>

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

Shell Prompts

<table>
<thead>
<tr>
<th>Shell</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C shell</td>
<td>machine-name%</td>
</tr>
<tr>
<td>C shell superuser</td>
<td>machine-name#</td>
</tr>
<tr>
<td>Bourne shell and Korn shell</td>
<td>$</td>
</tr>
<tr>
<td>Bourne shell and Korn shell superuser</td>
<td>#</td>
</tr>
</tbody>
</table>

Related Documentation

This book focuses on Open MPI and assumes familiarity with the MPI Standard. The following materials provide useful background about using Open MPI and about the
The Sun HPC ClusterTools documentation is available online at:

http://www.sun.com/documentation

For more information about the Sun HPC ClusterTools software, see the related Web site at:

http://www.sun.com/clustertools

For more information about Open MPI and its components, see the Open MPI web site at:

http://www.open-mpi.org

For more information about Sun Grid Engine software, see the Sun Grid Engine web site at:

http://www.sun.com/software/gridware
Documentation, Support, and Training

<table>
<thead>
<tr>
<th>Sun Function</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td><a href="http://www.sun.com/documentation/">http://www.sun.com/documentation/</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="http://www.sun.com/support/">http://www.sun.com/support/</a></td>
</tr>
<tr>
<td>Training</td>
<td><a href="http://www.sun.com/training/">http://www.sun.com/training/</a></td>
</tr>
</tbody>
</table>

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

http://www.sun.com/hwdocs/feedback

Please include the title and part number of your document with your feedback:

Sun HPC ClusterTools 8.1 Software User’s Guide, part number 820-3176-10
CHAPTER 1

Introduction to Sun HPC ClusterTools Software

Sun HPC ClusterTools™ 8.1 software is a set of parallel development tools that extend the Sun network computing solutions to high-end distributed-memory applications. This chapter summarizes its required configuration and principal components. It contains the following sections:

- “Supported Configurations” on page 1
- “Open Runtime Environment (ORTE)” on page 2
- “Integration With Distributed Resource Management Systems” on page 2
- “Open MPI Features” on page 3
- “Debugging With TotalView” on page 3

Supported Configurations

Sun HPC ClusterTools 8.1 software requires the Solaris™ 10 Operating System (Solaris 10 OS). All programs that execute under the Solaris 10 OS will execute in the Sun HPC ClusterTools environment.


Sun HPC ClusterTools 8.1 software can run MPI jobs of up to 4096 processes on as many as 1024 nodes. It also provides support for spawning MPI processes.

The Sun HPC ClusterTools 8.1 software runs on clusters connected by any TCP/IP-capable interconnect, such as high-speed Ethernet, Gigabit Ethernet, and Infiniband.
Open Runtime Environment (ORTE)

Sun HPC ClusterTools 8.1 is based on the Open MPI message-passing interface. Open MPI operates using the Open Runtime Environment (ORTE). ORTE starts jobs and provides some status information.

The Open MPI `mpirun` and `mpiexec` commands are actually symbolic links to the `orterun` command. All three commands perform the same function, which is to launch MPI jobs.

ORTE is compatible with a number of other launchers, including rsh/ssh, Sun Grid Engine, and PBS Professional/Torque.

Each of ORTE’s primary operations is summarized in the sections that follow. Subsequent chapters contain the procedures.

Executing Programs With `mpirun`

Sun HPC ClusterTools 8.1 software can start both serial and parallel jobs. The syntax and use of `mpirun` are described in Chapter 5.

Integration With Distributed Resource Management Systems

Sun HPC ClusterTools 8.1 software provides integration facilities with two select distributed resource management (DRM) systems. These systems provide proper resource allocation, parallel job control and monitoring, and proper job accounting. They are:

- Sun Grid Engine Version 6.1 and 6.2 software
- Altair PBS Professional 9.2, or Cluster Resources Torque 2.3

**Note** – Open MPI itself supports other third-party launchers supported by Open MPI, such as SLURM. However, these launchers are currently not supported in Sun HPC ClusterTools software. To use these other third-party launchers, you must download the Open MPI source, compile, and link with the libraries for the launchers.
You can launch parallel jobs directly from these distributed resource management systems. The DRM interacts closely with ORTE for proper resource description and with the multiple processes comprising the requested parallel job.

For a description of the scalable and open architecture of the DRM integration facilities, see “How the Open MPI Environment Is Integrated With Distributed Resource Management Systems” on page 7. For instructions, see Chapter 6.

Open MPI Features

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

- Integration with the Open Runtime Environment (ORTE)
- Support for MPI I/O
- Seamless use of different network protocols; for example, code compiled on a Sun HPC cluster that has fast Ethernet network can be run without change on a cluster that has an Infiniband network
- Multiprotocol support so that MPI picks the fastest available medium for each type of connection (such as shared memory, fast Ethernet, Infiniband)
- Communication via shared memory for fast performance
- Optimized collectives for multiprocessors and clusters of multiprocessors
- Full F77, C, and C++ support, and basic F90 support

Debugging With TotalView

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

- Making Sun HPC ClusterTools software compatible with the TotalView debugger
- Allowing Open MPI jobs to be debugged by TotalView using the Sun Grid Engine or the Portable Batch System (PBS)
- Allowing multiple instantiations of TotalView on a single cluster
- Supporting TotalView in Sun HPC ClusterTools software
Refer to the TotalView documentation at http://www.totalviewtech.com for more information about using TotalView.

In addition, the Open MPI Frequently Asked Questions (FAQ) contains information about how to use the TotalView debugger with Open MPI. This information is available at:

http://www.open-mpi.org/faq/?category=running#run-with-tv
CHAPTER 2

Fundamental Concepts

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

- “Clusters and Nodes” on page 5
- “Processes” on page 6
- “How the Open MPI Environment Is Integrated With Distributed Resource Management Systems” on page 7
- “MCA Parameters” on page 8
- “How ORTE Works With Zones in the Solaris 10 Operating System” on page 8

Clusters and Nodes

High performance computing clusters\(^1\) are groups of servers interconnected by any Sun-supported interconnect. Each server in a cluster is called a node. A cluster can consist of a single node.

ORTE (Open Run-Time Environment) is the runtime support system for Open MPI that allows users to execute their applications in a distributed clustering environment.

When using ORTE, you can select the cluster and nodes on which your MPI programs will run and how your processes will be distributed among them. For instructions, see Chapter 5, “Running Programs With the `mpirun` Command.”

---

1. Sun\(^{TM}\) Cluster is a completely different technology used for high availability (HA) applications.
For more information about how Open MPI allocates computing resources, see the FAQ entitled “Running MPI Jobs” at:

http://www.open-mpi.org/faq/?category=running

Processes

Open MPI allows you to control several aspects of job and process execution, such as:

- Number of processes to be launched
- Number of available slots on each node
- Process launcher to be used (such as Sun Grid Engine, PBS, rsh/ssh)
- Mapping processes to nodes

How Programs Are Launched

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

1. You can launch the job either interactively or through a script. Instructions for both are provided in Chapter 5 and Chapter 6.
2. You can enter the DRM processing environment (for example, Sun Grid Engine) before launching jobs with mpirun.
3. You can reserve resources for the parallel job and set other job control parameters from within the DRM, or use a hosts file to specify the parameters.

For tasks and instructions, see Chapter 5.
How the Open MPI Environment Is Integrated With Distributed Resource Management Systems

As described in Chapter 1, the Open MPI/Sun HPC ClusterTools 8.1 environment provides close integration between ORTE and several different DRM systems, including the following:

- Sun Grid Engine
- rsh/ssh
- PBS

The integration process is similar for all DRM systems, with some individual differences. At run time, `mpirun` calls the specified DRM system (launcher), which in turn launches the job.

For information on the ways in which `mpirun` interacts with DRM systems, see Chapter 5. In addition, see the FAQ on running MPI jobs at:

http://www.open-mpi.org/faq/?category=running

Chapter 6 provides instructions for script-based and interactive job launching.

Using Sun Grid Engine With ORTE

HPC sites use batch systems to share resources fairly and accountably, and also to guarantee that a job can obtain the resources it needs to run at maximum efficiency. To properly monitor a job’s resource consumption, the batch system must be the agent that launches the job.

Sun Grid Engine, like many other batch systems, cannot launch multiple process jobs (such as MPI applications) on its own. In Sun HPC ClusterTools 8.1, ORTE launches the multiple process jobs and sets up the environment required by Open MPI.

When Sun Grid Engine launches a parallel job in cooperation with ORTE, Sun Grid Engine “owns” the resulting launched processes. Sun Grid Engine monitors the resources for these processes, thereby creating a tightly integrated environment for resource accounting. OpenRTE allows users to execute their parallel applications.
Note – There is also an open source version of Grid Engine (GE) hosted on http://www.sunsourcenet. Although the Sun HPC ClusterTools 8.1/Open MPI integration is developed with Sun Grid Engine, this integration should work for the open source Grid Engine as well.

Submitting Jobs Under Sun Grid Engine Integration

To submit jobs under Sun Grid Engine integration in Sun HPC ClusterTools 8.1, you must first create a Sun Grid Engine (SGE) environment using qsub, qsh, and so on. Instructions about how to set up the parallel environment (PE) and queue in Sun Grid Engine are described in the Sun HPC ClusterTools 8.1 Software User’s Guide.

There are two ways to submit jobs under Sun Grid Engine integration: interactive mode and batch mode. “Running Parallel Jobs in the Sun Grid Engine Environment” on page 51 explains how to submit jobs in both modes in the Sun Grid Engine environment.

MCA Parameters

Open MPI provides MCA (Modular Component Architecture) parameters for use with the mpirun command. These parameters and their values direct mpirun to perform specified functions. To specify an MCA parameter, use the -mca flag and the parameter name and value with the mpirun command.

For more information about how to use MCA parameters, see Chapter 7.

How ORTE Works With Zones in the Solaris 10 Operating System

The Solaris 10 Operating System (Solaris 10 OS) enables you to create secure, isolated areas within a single instance of the Solaris 10 OS. These areas, called zones, provide secure environments for running applications. Applications that execute in one zone cannot monitor or affect activity in another zone. You can create multiple non-global zones to run as virtual instances of the Solaris OS on the same hardware.
The *global zone* is the default zone for the Solaris system. You install Sun HPC ClusterTools software into the global zone. Any non-global zones running under that Solaris system “inherit” that installation. This means that you may install and configure Sun HPC ClusterTools and compile/run/debug your programs in either a global or a non-global zone.

**Note** – The non-global zones do not inherit the links set up in the global zone. This means that you must either type out the full path to the Sun HPC ClusterTools executables on the command line, or run the `ctact` utility in the non-global zone to set up the links. For more information about the `ctact` utility, refer to the *Sun HPC ClusterTools 8.1 Software Installation Guide*. 
Before You Begin

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

- “Prerequisites” on page 11
- “Command and Man Page Paths” on page 11
- “Core Files” on page 13
- “Setting Up a known_hosts File” on page 13

Prerequisites

If your program uses Sun HPC ClusterTools components, compile and link it on a cluster that contains the Sun HPC ClusterTools software.

It is strongly suggested that you use the Sun HPC ClusterTools 8.1 wrapper compilers to compile applications. These wrapper compilers add the appropriate compiler and linker flags to the command line and call the underlying compiler and linker for you. Wrapper compilers are available for C, C++, Fortran 77, and Fortran 90.

For more information about compiling MPI applications, see the “Compiling MPI Applications” FAQ at:

http://www.open-mpi.org/faq/?category=mpi-apps

Command and Man Page Paths

Sun HPC ClusterTools commands typically reside in the following directories:
■ Solaris OS: /opt/SUNWhpc/HPC8.1/sun/bin
■ Linux:
  ■ /opt/SUNWhpc/HPC8.1/gnu/bin for the GNBU compiled version
  ■ /opt/SUNWhpc/HPC8.1/sun/bin for the Sun Studio compiled version

You can run the Sun HPC ClusterTools software directly from the directory in which yourClusterTools commands are installed, or you may add the directory to your PATH or set the PATH environment variable.

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

The Sun HPC ClusterTools files typically reside in /opt/SUNWhpc/HPC8.1.

**Note** – The examples in this manual refer to the default location for the Solaris and Sun Studio compiled Linux binaries. Be sure to use the path name that corresponds to your operating system and ClusterTools version.

## Setting Up Your Path

▼ **To Set Up Your Path for the Solaris OS and the Sun Studio Compiled Linux Version**

For example, if you installed the Sun HPC ClusterTools 8.1 software for the Solaris OS in the default location of /opt/SUNWhpc/HPC8.1/sun/bin, you would add this location to your PATH as shown in the following example:

```
% setenv PATH /opt/SUNWhpc/HPC8.1/sun/bin:$PATH
```

The `setenv` command prefixes the PATH on both the local and remote hosts with /opt/SUNWhpc/HPC8.1/sun/bin.

**Note** – If you are using the Solaris OS version of the ClusterTools software, you can also activate the Sun HPC ClusterTools software by running the `ctact` utility. Activation sets up the symbolic links so that you can run the software from any location on the host, not just the location where you installed it. For more information about activating the software, refer to the *Sun HPC ClusterTools 8.1 Software Installation Guide*. 
To Set Up Your Path for the GNU Compiled Linux Version

For example, if you installed the GNU compiled version of the Sun HPC ClusterTools 8.1 software for Linux in the default location of /opt/SUNWhpc/HPC8.1/gnu/bin, you would add this location to your PATH as shown in the following example:

```
% setenv PATH /opt/SUNWhpc/HPC8.1/gnu/bin:${PATH}
```

The setenv command prefixes the PATH on both the local and remote hosts with /opt/SUNWhpc/HPC8.1/gnu/bin.

Core Files

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

To disable the core dump, use the limit(1) command. You can use the following command in the C shell:

```
% limit coredumpsize 0
```

Setting Up a known_hosts File

If you are using ssh to connect to your remote nodes, you must set up your ~/.ssh/known_hosts file to contain the remote nodes' host key, especially if you try to run on a cluster with many nodes for the first time.
Setting up the known_hosts file avoids having to respond to the following prompts when running mpirun to the remote nodes:

```
% /opt/SUNWhpc/HPC8.1/sun/bin/mpirun -host host04,host05,host06 hostname

The authenticity of host ‘host04 (129.148.9.88)’ can’t be established.
RSA key fingerprint is
Are you sure you want to continue connecting (yes/no)?
The authenticity of host ‘host05 (129.148.9.84)’ can’t be established.
RSA key fingerprint is
Are you sure you want to continue connecting (yes/no)?
The authenticity of host ‘host06 (129.148.9.86)’ can’t be established.
RSA key fingerprint is
Are you sure you want to continue connecting (yes/no)?
Host key verification failed.
yes
Please type ‘yes’ or ‘no’: yes
Please type ‘yes’ or ‘no’: yes
Please type ‘yes’ or ‘no’: yes
Please type ‘yes’ or ‘no’: yes
Please type ‘yes’ or ‘no’: yes
Please type ‘yes’ or ‘no’: yes
Please type ‘yes’ or ‘no’: yes
```

...
CHAPTER 4

Compiling MPI Programs

This chapter describes the compilers that Sun HPC ClusterTools Software supports for both the Solaris OS and Linux. In addition, it describes changes you might make in your application code to recompile and run programs developed with a previous version of Sun HPC ClusterTools software in Sun HPC ClusterTools 8.1.

This chapter contains the following topics:

■ “Supported Compilers” on page 15
■ “Using the Wrapper Compilers” on page 16
■ “Using Non-Default Error Handlers” on page 16
■ “Compiling Fortran 90 Programs” on page 17

If you previously compiled your application with the `tmcc` compiler, then you must recompile your applications using the `mpicc` compiler if you want them to be compatible with Sun HPC ClusterTools 8.1 software. The `tmcc` compiler is not supported in Sun HPC ClusterTools 8.1, and there is no backward compatibility.

---

Supported Compilers

For the Solaris OS, Sun HPC ClusterTools 8.1 software supports Sun Studio 10, 11, and 12 C, C++, and Fortran compilers.

For the Linux OS, the ClusterTools 8.1 software supports the Sun Studio 12 compilers and the `gcc` Linux compiler versions 3.3.3, 3.4.6, and 4.1.2.
Using the Wrapper Compilers

Sun HPC ClusterTools 8.1 supplies wrapper compilers for you to use instead of
directly calling the compilers when compiling applications for use with the Sun HPC
ClusterTools 8.1 software. These wrapper compilers do not actually perform the
compilation and linking steps themselves, but they add the appropriate compiler
and linker flags and call the compiler and linker.

Note – Using the wrapper compilers is strongly suggested. If you decide not to use
them, the Open MPI Web site at http://www.open-mpi.org contains instructions
about how to compile without using them.

The following wrapper compilers are available:

<table>
<thead>
<tr>
<th>Language</th>
<th>Wrapper Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>mpicc</td>
</tr>
<tr>
<td>C++</td>
<td>mpiCC, mpiCXX, or mpic++ (Note: mpiCC is for use on case-sensitive file systems only)</td>
</tr>
<tr>
<td>Fortran 77</td>
<td>mpif77</td>
</tr>
<tr>
<td>Fortran 90</td>
<td>mpif90</td>
</tr>
</tbody>
</table>

For more information about the wrapper compilers, their use, and troubleshooting,
see the Open MPI FAQ at:

http://www.open-mpi.org/faq/?category=mpi-apps

Using Non-Default Error Handlers

In Open MPI, the non-default error handler does not persist, and the default error
handler is used. This causes any call used after MPI_Finalize to be aborted.
Compiling Fortran 90 Programs

When you are compiling MPI programs written in Fortran 90, you must use the `-xalias=actual` switch. Otherwise, your program could fail.

This condition is due to a known condition in the MPI standard. The standard states that “The MPI Fortran binding is inconsistent with the Fortran 90 standard in several respects.” Specifically, the Fortran 90 compiler could break MPI programs that use non-blocking operations.

For more information about this issue, see

CHAPTER 5

Running Programs With the mpirun Command

This chapter describes the general syntax of the mpirun command and lists the command’s options. This chapter also shows some of the tasks you can perform with the mpirun command. It contains the following sections:

- “About the mpirun Command” on page 19
- “Syntax for the mpirun Command” on page 20
- “mpirun Command Examples” on page 22
- “Mapping MPI Processes to Nodes” on page 24
- “Controlling Input/Output” on page 30
- “Controlling Other Job Attributes” on page 31
- “Submitting Jobs Under Sun Grid Engine Integration” on page 35
- “Using MPI Client/Server Applications” on page 39
- “mpirun Command Reference” on page 41
- “For More Information” on page 45

Note – The mpirun, mpiexec, and orterun commands all perform the same function, and they can be used interchangeably. The examples in this manual all use the mpirun command.

About the mpirun Command

The mpirun command controls several aspects of program execution in Open MPI. mpirun uses the Open Run-Time Environment (ORTE) to launch jobs. If you are running under distributed resource manager software, such as Sun Grid Engine or PBS, ORTE launches the resource manager for you.
If you are using rsh/ssh instead of a resource manager, you must use a hostfile or host list to identify the hosts on which the program will be run. When you issue the mpirun command, you specify the name of the hostfile or host list on the command line; otherwise, mpirun executes all the copies of the program on the local host, in round-robin sequence by CPU slot. For more information about hostfiles and their syntax, see “Specifying Hosts By Using a Hostfile” on page 25.

Both MPI programs and non-MPI programs can use mpirun to launch the user processes.

Some example programs are provided in the /opt/SUNWhpc/HPC8.1/examples directory for you to try to compile/run as sanity tests.

### Syntax for the mpirun Command

The following example shows the general single-process syntax for mpirun:

```bash
% mpirun [options] [program-name]
```

For a simple SPMD (Single Process, Multiple Data) job, the typical syntax is:

```bash
% mpirun -np x program-name
```

For jobs involving multiple instructions, the command syntax appears similar to the following:

```bash
% mpirun [options] [program-name] : [options2] [program-name2] ...
```

For an MPMD (Multiple Program, Multiple Data) parallel application, the syntax follows this form:

```bash
% mpirun -np x program1 : -np y program2
```

This command starts $x$ number of copies of the program program1, and then starts $y$ copies of the program program2.

### mpirun Options

The options control the behavior of the mpirun command. They might or might not be followed by arguments.
Caution – If you do not specify an argument for an option that expects to be followed by an argument (for example, the --app <filename> option), that option will read the next option on the command line as an argument. This might result in inconsistent behavior.

TABLE 5-2 on page 41 lists the options in alphabetical order, with a brief description of each.

Using Environment Variables With the mpirun Command

Use the -x args option (where args is the environment variable(s) you want to use) to specify any environment variable you want to pass during runtime. The -x option exports the variable specified in args and sets the value for args from the current environment. For example:

```
% mpirun -x LD_LIBRARY_PATH=/opt/SUNWhpc/HPC8.1/lib -np 4 a.out
```

Using MCA Parameters With the mpirun Command

The mpirun command uses MCA (Multiple Component Architecture) parameters to pass environment variables. To specify an MCA parameter, use the -mca option with the mpirun command, and then specify the parameter type, the parameter you want to pass as an environment variable, and the value you want to set. For example:

```
% mpirun --mca mpi_show_handleLeaks 1 -np 4 a.out
```

This sets the MCA parameter mpi_show_handleLeaks to the value of 1 before running the program named a.out with four processes. In general, the format used on the command line is --mca parameter_name value.

Note – There are multiple ways to specify the values of MCA parameters. This chapter discusses how to use them from the command line with the mpirun command. MCA parameters are discussed in more detail in Chapter 7.
Canceling Send and Receive Operations

Open MPI supports the canceling of receive operations. However, the canceling of sends is not supported; therefore, a send will never be successfully canceled.

For more information about canceling send and receive operations, see the MPI_Cancel(3) man page.

### mpirun Command Examples

The examples in this section show how to use the mpirun command options to specify how and where the processes and programs run.

The following table shows the process control options for the mpirun command. The procedures that follow the table explain how these options are used and show the syntax for each.

**TABLE 5-1  Program/Process Control Options**

<table>
<thead>
<tr>
<th>Task</th>
<th>mpirun option</th>
<th>Page Number (For More Information)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To run a program with default settings</td>
<td>(no need to specify an option)</td>
<td>18</td>
</tr>
<tr>
<td>To run multiple parallel processes</td>
<td>-c or -np &lt;number of processes&gt;</td>
<td>19</td>
</tr>
<tr>
<td>To display command help</td>
<td>-h or --help</td>
<td>27</td>
</tr>
<tr>
<td>To change the working directory</td>
<td>--wdir or --wdir &lt;directory&gt;</td>
<td>26</td>
</tr>
<tr>
<td>To specify the list of hosts on which to invoke processes (also known as the rankmap string)</td>
<td>-host or --host or -H</td>
<td>22</td>
</tr>
<tr>
<td>To specify the list of hosts on which to execute the program (also known as the rankmap file)</td>
<td>-hostfile &lt;filename&gt; or --hostfile &lt;filename&gt; or --machinefile &lt;filename&gt;</td>
<td>21</td>
</tr>
</tbody>
</table>
TABLE 5-1  (Continued)Program/Process Control Options

<table>
<thead>
<tr>
<th>Task</th>
<th>mpirun option</th>
<th>Page Number (For More Information)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To start up in debugging mode</td>
<td>-d or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>--debug or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-debugger or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>--debugger &lt;sequence&gt;</td>
<td></td>
</tr>
<tr>
<td>To specify verbose output</td>
<td>-v</td>
<td>27</td>
</tr>
<tr>
<td>To specify multiple executables</td>
<td>-np 2 exe1 : -np 6 exe2</td>
<td></td>
</tr>
</tbody>
</table>

▼ To Run a Program With Default Settings

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

```bash
% mpirun program-name
```

▼ To Run Multiple Processes

By default, an MPI program started with `mpirun` runs as one process.

- To run the program as multiple processes, use the `-np` option:

```bash
% mpirun -np process-count program-name
```

When you request multiple processes, ORTE attempts to start the number of processes you request, regardless of the number of CPUs available to run those processes. For more information, see “Oversubscribing Nodes” on page 27.

▼ To Direct `mpirun` By Using an Appfile

You can use a type of text file (called an appfile) to direct `mpirun`. The appfile specifies the nodes on which to run, the number of processes to launch on each node, and the programs to execute in a parallel application. When you use the `--app` option, `mpirun` takes all its direction from the contents of the appfile and ignores any other nodes or processes specified on the command line.
For example the following shows an appfile called my_appfile:

```bash
# Comments are supported; comments begin with #
# Application context files specify each sub-application in the
# parallel job, one per line. The first sub-application is the 2
# a.out processes:
-np 2 a.out
# The second sub-application is the 2 b.out processes:
-np 2 b.out
```

- To use the `--app` option with the `mpirun` command, specify the name and path of the appfile on the command line. For example:

```bash
% mpirun --app my_appfile
```

This command produces the same results as running `a.out` and `b.out` from the command line.

## Mapping MPI Processes to Nodes

When you issue the `mpirun` command from the command line, ORTE reads the number of processes to be launched from the `--np` option, and then determines where the processes will run.

To determine where the processes will run, ORTE uses the following criteria:

- Available hosts (also referred to as nodes), specified by a hostfile or by the `--host` option
- Scheduling policy (round-robin or by-slot)
- Default and maximum numbers of slots available on each host
- ORTE also checks to see whether the current environment/shell runs with any third-party launcher (such as Sun Grid Engine or PBS) to determine where the processes will launch.

## Specifying Available Hosts

You specify the available hosts to Open MPI in three ways:

- Through the batch scheduler in your resource management software. This option is described in detail in Chapter 6.
By using a hostfile with the \texttt{--hostfile} option. The hostfile is a text file that contains the names of hosts, the number of available slots on each host, and the maximum slots on each host.

By using the \texttt{--host} option. Use this option to specify which hosts to include or exclude.

**Specifying Hosts By Using a Hostfile**

The hostfile lists each node, the available number of slots, and the maximum number of slots on that node. For example, the following listing shows a simple hostfile:

```
node0
node1 slots=2
node2 slots=4 max_slots=4
node3 slots=4 max_slots=20
```

In this example file, \texttt{node0} is a single-processor machine. \texttt{node1} has two slots. \texttt{node2} and \texttt{node3} both have 4 slots, but the values of \texttt{slots} and \texttt{max_slots} are the same (4) on \texttt{node2}. This disallows the processors on \texttt{node2} from being oversubscribed. The four slots on \texttt{node3} can be oversubscribed, up to a maximum of 20 processes.

When you use this hostfile with the \texttt{--nooversubscribe} option (see “Oversubscribing Nodes” on page 27), \texttt{mpirun} assumes that the value of \texttt{max_slots} for each node in the hostfile is the same as the value of \texttt{slots} for each node. It overrides the values for \texttt{max_slots} set in the hostfile.

Open MPI assumes that the maximum number of slots you can specify is equal to infinity, unless explicitly specified. Resource managers also do not specify the maximum number of available slots.

\textbf{Note} – Open MPI includes a commented default hostfile at \\
\texttt{/opt/SUNWmpc/HPC8.1/etc/openmpi-default-hostfile}. Unless you specify a different hostfile at a different location, this is the hostfile that OpenMPI uses. It is empty by default, but you may edit this file to add your list of nodes. See the comments in the hostfile for more information.
Specifying Hosts By Using the --host Option

You can use the --host option to mpirun to specify the hosts you want to use on the command line in a comma-delimited list. For example, the following command directs mpirun to run a program called a.out on hosts a, b, and c:

```
% mpirun -np 3 --host a,b,c a.out
```

Open MPI assumes that the default number of slots on each host is one, unless you explicitly specify otherwise.

▼ To Specify Multiple Slots Using the --host Option

To specify multiple slots with the --host option for each host repeat the host name on the command line for each slot you want to use. For example:

```
% mpirun -host node1,node1,node2,node2 ...
```

If you are using a resource manager such as Sun Grid Engine or PBS, the resource manager maintains an accurate count of available slots.

Excluding Hosts From Scheduling By Using the --host Option

You can also use the --host option in conjunction with a hostfile to exclude any nodes not explicitly specified on the command line. For example, assume that you have the following hostfile called my_hosts:

```
a slots=2 max_slots=20
b slots=2 max_slots=20
c slots=2 max_slots=20
d slots=2 max_slots=20
```

Suppose you issue the following command to run program a.out:

```
% mpirun -np 1 --hostfile my_hosts --host c a.out
```

This command launches one instance of a.out on host c, but excludes the other hosts in the hostfile (a, b, and d).
**Note** – If you use these two options (`--hostfile` and `--host`) together, make sure that the host(s) you specify using the `--host` option also exist in the hostfile. Otherwise, `mpirun` exits with an error.

Oversubscribing Nodes

If you schedule more processes to run than there are available slots, this is referred to as *oversubscribing*. Oversubscribing a host is not suggested, as it might result in performance degradation.

`mpirun` has a `--nooversubscribe` option. This option implicitly sets the `max_slots` value (maximum number of available slots) to the same value as the `slots` value for each node, as specified in your hostfile. If the number of processes requested is greater than the `slots` value, `mpirun` returns an error and does not execute the command. This option overrides the value set for `max_slots` in your hostfile.

For more information about oversubscribing, see the following URL:

[http://www.open-mpi.org/faq/?category=running#oversubscribing](http://www.open-mpi.org/faq/?category=running#oversubscribing)

Scheduling Policies

ORTE uses two types of scheduling policies when it determines where processes will run:

- **By slot (default).** This scheme schedules processes to run on each successive slot on one host. When all those slots are filled, scheduling begins on the next host in the hostfile.
- **By node.** In this scheme, Open MPI schedules the processes by finding the first available slot on a host, then the first available slot on the next host in the hostfile, and so on, in a round-robin fashion.

**Scheduling By Slot**

This is the default scheduling policy for Open MPI. If you do not specify a scheduling policy, this is the policy that is used.
In by-slot scheduling, Open MPI schedules processes on a node until all of its available slots are exhausted (that is, all slots are running processes) before proceeding to the next node. In MPI terms, this means that Open MPI tries to maximize the number of adjacent ranks in MPI_COMM_WORLD on the same host without oversubscribing that host.

▼ To Specify By-Slot Scheduling

If you want to explicitly specify by-slot scheduling for some reason, there are two ways to do it:

1. Specify the --byslot option to mpirun. For example, the following command specifies the --byslot and --hostfile options:

   ```
   % mpirun -np 4 --byslot --hostfile myfile a.out
   ```

   The following example uses the -host option:

   ```
   % mpirun -np 4 --byslot -host node0,node0,node1,node1 a.out
   ```

2. Set the MCA parameter rmaps_base_schedule_policy to the value slot.
   For example:

   ```
   % mpirun --mca rmaps_base_schedule_policy slot -np 4 a.out
   ```

Note – The examples in this chapter set MCA parameters on the command line. For more information about the ways in which you can set MCA parameters, see Chapter 7. In addition, the Open MPI FAQ contains information about MCA parameters at the following URL:

   http://www.open-mpi.org/faq/?category=tuning#setting-mca-params
The following output example shows the contents of a simple hostfile called `my-hosts` and the results of the `mpirun` command using by-slot scheduling.

![Output Example]

```
% cat my-hosts
node0 slots=2 max_slots=20
node1 slots=2 max_slots=20
% mpirun --hostfile my-hosts -np 8 --byslot hello | sort
Hello World I am rank 0 of 8 running on node0
Hello World I am rank 1 of 8 running on node0
Hello World I am rank 2 of 8 running on node1
Hello World I am rank 3 of 8 running on node1
Hello World I am rank 4 of 8 running on node0
Hello World I am rank 5 of 8 running on node0
Hello World I am rank 6 of 8 running on node1
Hello World I am rank 7 of 8 running on node1
```

### Scheduling By Node

In by-node scheduling, Open MPI schedules a single process on each node in a round-robin fashion (looping back to the beginning of the node list as necessary) until all processes have been scheduled. Nodes are skipped once their default slot counts are exhausted.

▼ **To Specify By-Node Scheduling**

There are two ways to specify by-node scheduling:

- Specify the `--bynode` option to `mpirun`. For example:

  ```bash
  % mpirun -np 4 --bynode --hostfile my-hosts a.out
  ```

- Set the MCA parameter `rmaps_base_schedule_policy` to the value `node`. For example:

  ```bash
  % mpirun --mca rmaps_base_schedule_policy node -np 4 a.out
  ```
The following output example shows the contents of the same hostfile used in the previous example and the results of the `mpirun` command using by-node scheduling.

```
% cat my-hosts
node0 slots=2 max_slots=20
node1 slots=2 max_slots=20
% mpirun --hostfile my-hosts -np 8 --bynode hello | sort
Hello World I am rank 0 of 8 running on node0
Hello World I am rank 1 of 8 running on node1
Hello World I am rank 2 of 8 running on node0
Hello World I am rank 3 of 8 running on node1
Hello World I am rank 4 of 8 running on node0
Hello World I am rank 5 of 8 running on node1
Hello World I am rank 6 of 8 running on node0
Hello World I am rank 7 of 8 running on node1
```

Comparing By-Slot to By-Node Scheduling

In the examples in this section, node0 and node1 each have two slots. The diagrams show the differences in scheduling between the two methods.

By-slot scheduling for the two nodes can be represented as follows:

```
 node0  node1
  0    2
  1    3
  4    6
  5    7
```

By-node scheduling for the same two nodes can be represented this way:

```
 node0  node1
  0    1
  2    3
  4    5
  6    7
```

Controlling Input/Output

Open MPI directs UNIX standard input to `/dev/null` on all processes except the rank 0 process of `MPI_COMM_WORLD`. The `MPI_COMM_WORLD` rank 0 process inherits standard input from `mpirun`. The node from which you invoke `mpirun`
need not be the same as the node where the MPI_COMM_WORLD rank 0 process resides. Open MPI handles the redirection of the `mpirun` standard input to the rank 0 process.

Open MPI directs UNIX standard output and standard error from remote nodes to the node that invoked `mpirun`, and then prints the information from the remote nodes on the standard output/error of `mpirun`. Local processes inherit the standard output/error of `mpirun` and transfer to it directly.

▼ To Redirect Standard I/O

To redirect standard I/O for Open MPI applications, use the typical shell redirection procedure on `mpirun`. For example:

```
% mpirun -np 2 my_app < my_input > my_output
```

In this example, only the MPI_COMM_WORLD rank 0 process will receive the stream from `my_input` on `stdin`. The `stdin` on all the other nodes will be tied to `/dev/null`. However, the `stdout` from all nodes will be collected into the `my_output` file.

### Controlling Other Job Attributes

<table>
<thead>
<tr>
<th>To Perform This Task</th>
<th>Use This Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>To change the working directory</td>
<td><code>-wdir</code> or <code>--wdir</code></td>
</tr>
<tr>
<td>To display debugging output</td>
<td><code>-d</code></td>
</tr>
<tr>
<td>To display command help</td>
<td><code>-h</code></td>
</tr>
</tbody>
</table>

▼ To Change the Working Directory

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

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

```
% mpirun --wdir /home/mystuff/bin a.out
```

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

▼ To Specify Debugging Output

Use this syntax to specify debugging output. For example:

```
% mpirun -d a.out
```

The -d option shows the user-level debugging output for all of the ORTE modules used with mpirun. To see more information from a particular module, you can set additional MCA debugging parameters. The availability of the additional debugging information depends on how the module of interest is implemented.

For more information on MCA parameters, see Chapter 7. For more information about whether a module provides additional verbose or debug mode, run the ompi_info command on that module.

▼ To Display Command Help (–h)

To display a list of mpirun options, use the -h option (alone). The following example shows the output from mpirun -h:

```
% ./mpirun -h
mpirun (Open MPI) 1.3r19845-ct8.1-b06a-r21
Usage: mpirun [OPTION]... [PROGRAM]...
Start the given program using Open RTE

  -am <arg0>         Aggregate MCA parameter set file list
  --app <arg0>       Provide an appfile; ignore all other command line options
  -bynode|--bynode   Whether to allocate/map processes round-robin by node
  -byslot|--byslot   Whether to allocate/map processes round-robin by
```

-c|--np|--np <arg0> Number of processes to run
-cf|--cartofile <arg0> Provide a cartography file
-d|--debug-devel|--debug-devel
   Enable debugging of OpenRTE
-debug|--debug
   Invoke the user-level debugger indicated by the orte_base_user_debugger MCA parameter
-debug-daemons|--debug-daemons
   Enable debugging of any OpenRTE daemons used by this application
-debug-daemons-file|--debug-daemons-file
   Enable debugging of any OpenRTE daemons used by this application, storing output in files
-debugger|--debugger <arg0>
   Sequence of debuggers to search for when "--debug" is used
-default-hostfile|--default-hostfile <arg0>
   Provide a default hostfile
-display-allocation|--display-allocation
   Display the allocation being used by this job
-display-devel-allocation|--display-devel-allocation
   Display a detailed list (mostly intended for developers) of the allocation being used by this job
-display-devel-map|--display-devel-map
   Display a detailed process map (mostly intended for developers) just before launch
-display-map|--display-map
   Display the process map just before launch
-do-not-launch|--do-not-launch
   Perform all necessary operations to prepare to launch the application, but do not actually launch it
-do-not-resolve|--do-not-resolve
   Do not attempt to resolve interfaces
-gmca|--gmca <arg0> <arg1>
   Pass global MCA parameters that are applicable to all contexts (arg0 is the parameter name; arg1 is the parameter value)
-h|--help
   This help message
-H|--host|--host <arg0>
   List of hosts to invoke processes on
   --hetero
   Indicates that multiple app_contexts are being provided that are a mix of 32/64 bit binaries
-hostfile|--hostfile <arg0>
   Provide a hostfile
-launch-agent|--launch-agent <arg0>
   Command used to start processes on remote nodes (default: orted)
-leave-session-attached|--leave-session-attached
  Enable debugging of OpenRTE
-loadbalance|--loadbalance
  Balance total number of procs across all allocated nodes
-machinefile|--machinefile <arg0>
  Provide a hostfile
-mca|--mca <arg0> <arg1>
  Pass context-specific MCA parameters; they are considered
  global if --gmca is not used and only one context is specified
  (arg0 is the parameter name; arg1 is the parameter value)
-n|--n <arg0>
  Number of processes to run
-nolocal|--nolocal
  Do not run any MPI applications on the local node
-nooversubscribe|--nooversubscribe
  Nodes are not to be oversubscribed, even if the system
  supports such operation
-noprefix
  Disable automatic --prefix behavior
-npernode|--npernode <arg0>
  Launch n processes per node on all allocated nodes
-ompi-server|--ompi-server <arg0>
  Specify the URI of the Open MPI server, or the name of the file
  (specified as file:filename) that contains that info
-path|--path <arg0>
  PATH to be used to look for executables to start processes
-pernode|--pernode
  Launch one process per available node on the specified number of
  nodes [no -np => use all allocated nodes]
-prefix <arg0>
  Prefix where Open MPI is installed on remote nodes
--preload-files <arg0>
  Preload the comma separated list of files to the remote machines
  current working directory before starting the remote process.
--preload-files-dest-dir <arg0>
  The destination directory to use in conjunction with --preload-files.
  By default the absolute and relative paths provided by
  --preload-files are used.
-q|--quiet
  Suppress helpful messages
-rf|--rankfile <arg0>
  Provide a rankfile
-s|--preload-binary
  Preload the binary on the remote machine before starting the remote process.
-server-wait-time|--server-wait-time <arg0>
  Time in seconds to wait for ompi-server (default: 10 sec)
-slot-list|--slot-list <arg0>
  List of processor IDs to bind MPI processes to
Submitting Jobs Under Sun Grid Engine Integration

There are two ways to submit jobs under Sun Grid Engine integration: interactive mode and batch mode. The instructions in this chapter describe how to submit jobs interactively. For information about how to submit jobs in batch mode, see Chapter 6.

Defining Parallel Environment (PE) and Queue

A PE needs to be defined for all the queues in the Sun Grid Engine cluster to be used as ORTE nodes. Each ORTE node should be installed as an Sun Grid Engine execution host. To allow the ORTE to submit a job from any ORTE node, configure each ORTE node as a submit host in Sun Grid Engine.
Each execution host must be configured with a default queue. In addition, the default queue set must have the same number of slots as the number of processors on the hosts.

▼ To Use PE Commands

● To display a list of available PEs (parallel environments), type the following:

```
% qconf -spl
make
```

● To define a new PE, you must have Sun Grid Engine manager or operator privileges. Use a text editor to modify a template for the PE. The following example creates a PE named orte.

```
% qconf -ap orte
```

● To modify an existing PE, use this command to invoke the default editor:

```
% qconf -mp orte
```

● To show a particular PE that has been defined, type this command:

```
% qconf -sp orte
```

<table>
<thead>
<tr>
<th>pe_name</th>
<th>orte</th>
</tr>
</thead>
<tbody>
<tr>
<td>slots</td>
<td>8</td>
</tr>
<tr>
<td>user_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>xuser_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>start_proc_args</td>
<td>/bin/true</td>
</tr>
<tr>
<td>stop_proc_args</td>
<td>/bin/true</td>
</tr>
<tr>
<td>allocation_rule</td>
<td>$round_robin</td>
</tr>
<tr>
<td>control_slaves</td>
<td>TRUE</td>
</tr>
<tr>
<td>job_is_first_task</td>
<td>FALSE</td>
</tr>
<tr>
<td>urgency_slots</td>
<td>min</td>
</tr>
</tbody>
</table>

The value NONE in user_lists and xuser_lists mean enable everybody and exclude nobody.

The value of control_slaves must be TRUE; otherwise, qrsh exits with an error message.

The value of job_is_first_task must be FALSE or the job launcher consumes a slot. In other words, mpirun itself will count as one of the slots and the job will fail, because only n-1 processes will start.
▼ To Use Queue Commands

- To show all the defined queues, type the following command:

```bash
% qconf -sql
all.q
```

The queue `all.q` is set up by default in Sun Grid Engine.

- To configure the `orte` PE from the example in the previous section to the existing queue, type the following:

```bash
% qconf -mattr queue pe_list "orte" all.q
```

You must have Sun Grid Engine manager or operator privileges to use this command.

### Submitting Jobs in Interactive Mode

▼ To Set the Interactive Display

Before you submit a job, you must have your DISPLAY environment variable set so that the interactive window will appear on your desktop, if you have not already done so.

For example, if you are working in the C shell, type the following command:

```bash
% setenv DISPLAY desktop:0.0
```

▼ To Submit Jobs Interactively

1. Use the `source` command to set the Sun Grid Engine environment variables from a file:

```bash
mynode4% source /opt/sge/default/common/settings.csh
```
2. Use the `qsh` command to start the interactive X Windows session, and specify the parallel environment (in this example, ORTE) and the number of slots to use:

```
mynode4% qsh -pe orte 2
waiting for interactive job to be scheduled...
Your interactive job 324 has been successfully scheduled.
```

3. On a different node in the cluster, use the `cd` command to switch to the directory where your executable is located.

```
mynode5% cd /workspace/joeuser/ompi/trunk/builds/sparc32-g/bin
```

4. Issue the `mpirun` command.

```
mynode5% /opt/SUNWhpc/HPC8.1/sun/bin/mpirun -np 4 hostname
```

In the above example, Sun Grid Engine starts the user executable `hostname` with 4 processes on the two Sun Grid Engine assigned slots. The following example shows the output from the `mpirun` command with the specified options.

```
mynode5%/opt/SUNWhpc/HPC8.1/sun/bin/mpirun -np 4 --hostname mynode5
```

▼ To Verify That Sun Grid Engine Is Running

The following is not required for normal operation, but if you want to verify that Sun Grid Engine is being used, add `--mca ras_gridengine_verbose` to the `mpirun` command line. For example:

```
% ./mpirun -np 4 --mca ras_gridengine_verbose 100 hostname
mynode6
mynode6
mynode7
mynode7
%
```
To Start an Interactive Session Using qrsh

An alternate way to start an interactive session is by using qrsh instead of qsh. For example:

```
% qrsh -V -pe orte 8 mpirun -np 4 -byslot hostname
```

Using MPI Client/Server Applications

The instructions in this section explain how to get best results when starting Open MPI client/server applications.

To Launch the Client/Server Job

1. Type the following command to launch the server application. Substitute the name of your MPI job’s universe for `univ1`:

```
% ./mpirun -np 1 --universe univ1 t_accept
```

2. Type the following command to launch the client application, substituting the name of your MPI job’s universe for `univ1`:

```
% ./mpirun -np 4 --universe univ1 t_connect
```

If the client and server jobs span more than 1 node, the first job (that is, the server job) must specify on the mpirun command line all the nodes that will be used. Specifying the node names allocates the specified hosts from the entire universe of server and client jobs.

For example, if the server runs on `node0` and the client job runs on `node1` only, the command to launch the server must specify both nodes (using the `--host node0,node1` flag) even it uses only one process on `node0`.

Assuming that the persistent daemon is started on `node0`, the command to launch the server would look like this:

```
node0% ./mpirun -np 1 --universe univ1 -host node0,node1 t_accept
```

The command to launch the client is:

```
node0% ./mpirun -np 4 --universe univ1 -host node1 t_connect
```
Using Name Publishing

If you are planning on using name publishing, you must perform some additional tasks. You need to start up an omni-server process on your server so that both the clients and servers can exchange information using that server.

For information about how to start the omni-server process, type the following command on your server:

```
% man omni-server
```

Troubleshooting Client/Server Jobs

If the MPI client/server job fails to start, you might see error messages similar to this:

```
node0% ./orted --persistent --seed --scope public --universe univ4 --debug
  [node0:21760] procdir: (null)
  [node0:21760] jobdir: (null)
  [node0:21760] unidir: /tmp/openmpi-sessions-joeuser@node0_0/univ4
  [node0:21760] top: openmpi-sessions-joeuser@node0_0
  [node0:21760] tmp: /tmp
  [node0:21760] orte_init: could not contact the specified
  universe name univ4
  [node0:21760] [NO-NAME] ORTE_ERROR_LOG: Unreachable in file
  /opt/SUNW/hpc/HPC8.1/sun/bin/orted/runtime/orte_init_stage1.c
  at line 221
```

These messages indicate that there is residual data left in the /tmp directory. This can happen if a previous client/server job has already run from the same node.

To empty the /tmp directory, use the orte-clean utility. For more information about orte-clean, see the orte-clean man page.

You might also need to run orte-clean if you see error messages similar to the following:

```
node0% ./orted --persistent --seed --scope public --universe univ4 --debug
  [node0:21760] procdir: (null)
  [node0:21760] jobdir: (null)
  [node0:21760] unidir: /tmp/openmpi-sessions-joeuser@node0_0/univ4
  [node0:21760] top: openmpi-sessions-joeuser@node0_0
```

40 Sun HPC ClusterTools 8.1 Software User's Guide • November 2008
mpirun Command Reference

This section provides a quick reference for the mpirun command options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-am list-name</td>
<td>Use the MCA parameter set file list called list-name.</td>
</tr>
<tr>
<td>--app appfile</td>
<td>Directs mpirun to use the appfile specified by appfile and to ignore other programs specified on the command line</td>
</tr>
<tr>
<td>-bynode</td>
<td>Allocates (maps) the processes specified in a round-robin scheme by node. -byslot is the default (see below).</td>
</tr>
<tr>
<td>--bynode</td>
<td>Allocates (maps) the processes specified in a round-robin scheme by node. -byslot is the default (see below).</td>
</tr>
<tr>
<td>-byslot</td>
<td>Allocates (maps) the processes specified in a round-robin scheme by slot (processor) This is the default.</td>
</tr>
</tbody>
</table>

It looks like orte_init failed for some reason; your parallel process is likely to abort. There are many reasons that a parallel process can fail during orte_init; some of which are due to configuration or environment problems. This failure appears to be an internal failure; here’s some additional information (which may only be relevant to an Open MPI developer):

orte_sds_base_contact_universe failed  
--> Returned value -12 instead of ORTE_SUCCESS

Open RTE was unable to initialize properly. The error occurred while attempting to orte_init(). Returned value -12 instead of ORTE_SUCCESS.
TABLE 5-2  mpirun Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c number</td>
<td>Same as the -np &lt;number&gt; option. Directs mpirun to run the number of copies (specified in number) of the specified program on the selected nodes. See the description of the -np option for more information.</td>
</tr>
<tr>
<td>-cf</td>
<td>Run using the cartography file filename. Cartography files describe the layout of and connections between components in a cluster. For more information about cartography files, see the mpirun(1) man page.</td>
</tr>
<tr>
<td>--cartofile filename</td>
<td></td>
</tr>
<tr>
<td>-debug</td>
<td>Invokes the user-level debugger specified in the MCA parameter orte_base_user_debugger. The default value for the MCA parameter is totalview. To change the specified debugger, change the value of the MCA parameter. (See Chapter 7 for more information.)</td>
</tr>
<tr>
<td>--debug-daemons</td>
<td>Enable debugging of any ORTE daemons used by this application.</td>
</tr>
<tr>
<td>--debug-daemons-file</td>
<td>Enable debugging of any OpenRTE daemons used by this application, storing output in files.</td>
</tr>
<tr>
<td>--debug-devel</td>
<td>Enable debugging of OpenRTE.</td>
</tr>
<tr>
<td>--debugger</td>
<td>Specifies the sequence of debuggers you want to use with mpirun. This option is a synonym for the orte_base_user_debugger, and has the same default value. If you use this option, the value you specify overrides any value set in orte_base_user_debugger.</td>
</tr>
<tr>
<td>--debugger</td>
<td></td>
</tr>
<tr>
<td>--default-hostfile</td>
<td>Run using the provided default hostfile filename.</td>
</tr>
<tr>
<td>--default-hostfile</td>
<td></td>
</tr>
<tr>
<td>--display-allocation</td>
<td>Display the allocation being used by this job.</td>
</tr>
<tr>
<td>--display-allocation</td>
<td></td>
</tr>
<tr>
<td>--display-devel-allocation</td>
<td>Intended for Open MPI/OpenRTE developers. Display a detailed list of the allocation being used by this job.</td>
</tr>
<tr>
<td>--display-map</td>
<td>Display the process map just before launch.</td>
</tr>
<tr>
<td>--display-map</td>
<td></td>
</tr>
<tr>
<td>--display-devel-map</td>
<td>Intended for Open MPI/OpenRTE developers. Display a detailed process map just before launch.</td>
</tr>
<tr>
<td>--do-not-launch</td>
<td>Perform all necessary operations to prepare to launch the application, but do not actually launch it.</td>
</tr>
<tr>
<td>--do-not-resolve</td>
<td>Do not attempt to resolve interfaces.</td>
</tr>
<tr>
<td>--do-not-resolve</td>
<td></td>
</tr>
</tbody>
</table>
TABLE 5-2  mpirun Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-gmca</td>
<td>Specifies global MCA parameters. param is the name of the specified MCA parameter. value is the value for that parameter.</td>
</tr>
<tr>
<td>--gmca param value</td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>Displays help for the mpirun command. When this option is specified on the command line, it overrides any other options and displays the command help.</td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>-H host1, host2, ...hostn</td>
<td>Specifies the list of hosts on which to invoke processes. This is a synonym for -host.</td>
</tr>
<tr>
<td>--hetero</td>
<td>Indicates that multiple app_contexts are being provided that are a mix of 32 - amd 64-bit binaries.</td>
</tr>
<tr>
<td>-host</td>
<td>Specifies the list of hosts on which to invoke processes. This is a synonym for -H.</td>
</tr>
<tr>
<td>--host &lt;host1,host2,...hostn&gt;</td>
<td></td>
</tr>
<tr>
<td>-hostfile</td>
<td>Directs mpirun to use the specified hostfile. If -hostfile is specified without using filename, mpirun uses the default hostfile located at /opt/SUNWhpcc/HPC8.1/etc/openmpi-default-hostfile.</td>
</tr>
<tr>
<td>--hostfile filename</td>
<td></td>
</tr>
<tr>
<td>--launch-agent command-name</td>
<td>Command used to start processes on remote nodes (default: orted)</td>
</tr>
<tr>
<td>--leave-session-attached</td>
<td>Enable debugging of OpenRTE.</td>
</tr>
<tr>
<td>--loadbalance</td>
<td>Balance total number of processes across all allocated nodes.</td>
</tr>
<tr>
<td>--machinefile filename</td>
<td>Synonymous with -hostfile.</td>
</tr>
<tr>
<td>--machinefile filename</td>
<td></td>
</tr>
<tr>
<td>-mca</td>
<td>Specifies an MCA parameter, where param is the name of the desired MCA parameter and value is the desired value for that parameter. These parameters and values are considered to be global parameters unless the -gmca option appears on the same command line.</td>
</tr>
<tr>
<td>--mca param value</td>
<td></td>
</tr>
<tr>
<td>-n, --n number</td>
<td>Specifies the number of processes to run. Synonymous with -np.</td>
</tr>
<tr>
<td>--no-daemonize</td>
<td>Keeps the ORTE daemons used by this application from being detached and used by other processes.</td>
</tr>
<tr>
<td>--nolocal, --nolocal</td>
<td>Specifies that MPI applications should not be run on the local node (the same node on which mpirun is running).</td>
</tr>
</tbody>
</table>

Chapter 5  Running Programs With the mpirun Command  43
### TABLE 5-2  mpirun Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-nooversubscribe</td>
<td>Never oversubscribe the nodes, even if the system supports such operations.</td>
</tr>
<tr>
<td>--nooversubscribe</td>
<td>This option sets the effective value of max_slots to equal the value of slots, and overrides the settings for that node in the hostfile.</td>
</tr>
<tr>
<td>--noprefix value</td>
<td>Cancels any previously specified directory options specified by the --prefix option.</td>
</tr>
<tr>
<td>-npernode</td>
<td>Launch number processes per node on all allocated nodes.</td>
</tr>
<tr>
<td>--npernode number</td>
<td>Launch number processes per node on all allocated nodes.</td>
</tr>
<tr>
<td>-ompi-server name</td>
<td>Specify the URL of the Open MPI server, or the name of the file (specified as file:filename) that contains that information needed to run the job.</td>
</tr>
<tr>
<td>-path path name</td>
<td>Specifies to mpirun that the executables to be used for the current job are stored in pathname.</td>
</tr>
<tr>
<td>-pernode</td>
<td>Launch one process per available node on the number of nodes specified in the -np option. If no -np option is used, then use all allocated nodes.</td>
</tr>
<tr>
<td>--pernode number</td>
<td>Launch one process per available node on the number of nodes specified in the -np option. If no -np option is used, then use all allocated nodes.</td>
</tr>
<tr>
<td>--prefix path name</td>
<td>Specifies the path to the directory where Open MPI is located on remote node(s). This option is used to run Open MPI on remote nodes (as opposed to running on the local node).</td>
</tr>
<tr>
<td>--preload-files filename</td>
<td>Preload the comma separated list of files (specified by filename) to the remote machine’s current working directory before starting the remote process.</td>
</tr>
<tr>
<td>--preload-files dest-dir directory</td>
<td>Specifies the destination directory (specified by directory) that contains the list of files (specified by --preload-files filename) to be used with the --preload-files option. By default, this option uses both absolute and relative paths.</td>
</tr>
<tr>
<td>-q, -quiet</td>
<td>Suppresses output messages from Open MPI.</td>
</tr>
<tr>
<td>-rf, --rankfile filename</td>
<td>Provide a rankfile file.</td>
</tr>
<tr>
<td>-s, --preload-binary</td>
<td>Preload the binary on the remote machine before starting the remote process.</td>
</tr>
<tr>
<td>--server-wait-time seconds</td>
<td>Time in seconds to wait for ompi-server (default: 10 sec).</td>
</tr>
<tr>
<td>--slot-list id-list</td>
<td>List of processor IDs to which you want to bind MPI processes (for example, a list of processors used in conjunction with rankfile files)</td>
</tr>
<tr>
<td>--tmpdir path name</td>
<td>Specifies the root for the session directory tree for mpirun only. This applies only to the current job.</td>
</tr>
</tbody>
</table>
For More Information

For more information about the `mpirun` command and its options, see the following:

- **Chapter 7, “Using MCA Parameters With `mpirun`” on page 59**
- the `mpirun(3)` man page
- Open MPI FAQ at [http://www.open-mpi.org](http://www.open-mpi.org)
Running Programs With `mpirun` in Distributed Resource Management Systems

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

- “`mpirun` Options for Third-Party Resource Manager Integration” on page 47
- “Running Parallel Jobs in the PBS Environment” on page 48
- “Running Parallel Jobs in the Sun Grid Engine Environment” on page 51

`mpirun` Options for Third-Party Resource Manager Integration

ORTE is compatible with a number of other launchers, including `rsh/ssh`, Sun Grid Engine, and PBS.

**Note** – Open MPI itself supports other third-party launchers supported by Open MPI, such as SLURM and Torque. However, these launchers are currently not supported in Sun HPC ClusterTools software. To use these other third-party launchers, you must download the Open MPI source, compile, and link with the libraries for the launchers.
Checking Your Open MPI Configuration

To see whether your Open MPI installation has been configured for use with the third-party resource manager you want to use, issue the `ompi_info` command and pipe the output to `grep`. The following examples show how to use `ompi_info` to check for the desired third-party resource manager.

▼ To Check for `rsh/ssh`

To see whether your Open MPI installation has been configured to use the `rsh/ssh` launcher:

```
% ompi_info | grep rsh
MCA plm: rsh (MCA v2.0, API v2.0, Component v1.3)
```

▼ To Check for PBS/Torque

To see whether your Open MPI installation has been configured to use the PBS/Torque launcher:

```
% ompi_info | grep tm
MCA ras: tm (MCA v2.0, API v2.0, Component v1.3)
MCA plm: tm (MCA v2.0, API v2.0, Component v1.3)
```

▼ To Check for Sun Grid Engine

To see whether your Open MPI installation has been configured to use Sun Grid Engine:

```
% ompi_info | grep gridengine
MCA ras: gridengine (MCA v2.0, API v2.0, Component v1.3)
```

Running Parallel Jobs in the PBS Environment

If your Open MPI environment is set up to include PBS, Open MPI automatically detects when `mpirun` is running within PBS, and will execute properly.
First reserve the number of resources by invoking the `qsub` command with the `-l` option. The `-l` option specifies the number of nodes and the number of processes per node. For example, this command sequence reserves four nodes with four processes per node for the job `myjob.sh`:

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

When you enter the PBS environment, you can launch an individual job or a series of jobs with `mpirun`. The `mpirun` command launches the job using the nodes and processes information from PBS. The resource information is accessed using the `tm` calls provided by PBS; hence, `tm` is the name used to identify the module in ORTE. The job ranks are children of PBS, not ORTE.

You can run an ORTE job within the PBS environment in two different ways: interactive and scripted.

▼ To Run an Interactive Job in PBS

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

   ```
   % qsub -l nodes=2:ppn=2 -I
   ```

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

   ```
   qsub: waiting for job 20.mynode to start
   qsub: job 20.mynode ready
   Sun Microsystems Inc. SunOS 5.10 Generic June 2006
   pbs%
   ```

2. Launch the `mpirun` command.
   Here is an example that launches the `hostname` command with a verbose output:

   ```
pbs% /opt/SUNWhpc/HPC8.1/sun/bin/mpirun -np 4 -mca plm_tm_verbose
np 1 hostname
```
The output shows the `hostname` program being run on ranks r0 and r1:

```
% /opt/SUNWhpcc/HPC8.1/sun/bin/mpirun -np 4 -mca plm_tm_verbose 1 hostname
[hostname1:09064] plm:tm: launching on node mynode1
[hostname2:09064] plm:tm: launching on node mynode2
hostname2
hostname1
hostname2
hostname1
```

▼ To Run a Batch Job in PBS

1. Write a script that calls `mpirun`.
   
   In the following examples, the script is called `myjob.csh`. The system is called `mynode`. Here is an example of the script.

```
#!/bin/csh
/opt/SUNWhpcc/HPC8.1/sun/bin/mpirun -np 2 -mca plm_tm_verbose 1 hostname
```

2. Enter the PBS environment and use the `-l` option to `qsub` to reserve resources for the job.
   
   Here is an example of how to use the `-l` option with the `qsub` command.

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

This command enters the PBS environment and reserves one node with two processes for the job that will be launched by the script named `myjob.csh`. 
Here is the output to the script `myjob.csh`.

```bash
% more myjob.csh.*
::::::::::::::
myjob.csh.e2365
::::::::::::::
::::::::::::::
myjob.csh.o2365
::::::::::::::
Warning: no access to tty (Bad file number).
The shell has no job control.
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
hostname5
hostname4
hostname5
hostname4
```

After the job finishes, it generates two output files:

- `name_of_job.ejob_id` is the file that shows the error outputs (for example, `myjob.csh.e2365` in the above example).
- `name_of_job.0job_id` is the file that shows the standard outputs (for example, `myjob.csh.o2365` in the above example).

As you can see, the `pbsrun` command calls `mpirun`, which forks into two calls of the `hostname` program, one for each node.

---

### Running Parallel Jobs in the Sun Grid Engine Environment

Sun Grid Engine 6.1 is the supported version of Sun Grid Engine for Sun HPC ClusterTools 8.1.

Before you can run parallel jobs, make sure that you have defined the parallel environment and queue before running the job.
Defining Parallel Environment (PE) and Queue

A PE needs to be defined for all the queues in the Sun Grid Engine cluster to be used as ORTE nodes. Each ORTE node should be installed as a Sun Grid Engine execution host. To allow the ORTE to submit a job from any ORTE node, configure each ORTE node as a submit host in Sun Grid Engine.

Each execution host must be configured with a default queue. In addition, the default queue set must have the same number of slots as the number of processors on the hosts.

▼ To Use PE Commands

- To display a list of available PEs (parallel environments), type the following:

  % qconf -spl
  make

- To define a new PE, you must have Sun Grid Engine manager or operator privileges. Use a text editor to modify a template for the PE. The following example creates a PE named orte.

  % qconf -ap orte

- To modify an existing PE, use this command to invoke the default editor:

  % qconf -mp orte

- To show a particular PE that has been defined, type this command:

  % qconf -sp orte

<table>
<thead>
<tr>
<th>pe_name</th>
<th>orte</th>
</tr>
</thead>
<tbody>
<tr>
<td>slots</td>
<td>8</td>
</tr>
<tr>
<td>user_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>xuser_lists</td>
<td>NONE</td>
</tr>
<tr>
<td>start_proc_args</td>
<td>/bin/true</td>
</tr>
<tr>
<td>stop_proc_args</td>
<td>/bin/true</td>
</tr>
<tr>
<td>allocation_rule</td>
<td>$round_robin</td>
</tr>
<tr>
<td>control_slaves</td>
<td>TRUE</td>
</tr>
<tr>
<td>job_is_first_task</td>
<td>FALSE</td>
</tr>
<tr>
<td>urgency_slots</td>
<td>min</td>
</tr>
</tbody>
</table>

The value NONE in user_lists and xuser_lists mean enable everybody and exclude nobody.
The value of control_slaves must be TRUE; otherwise, qrsh exits with an error message.

The value of job_is_first_task must be FALSE or the job launcher consumes a slot. In other words, mpirun itself will count as one of the slots and the job will fail, because only n-1 processes will start.

▼ To Use Queue Commands

- To show all the defined queues, type the following command:

```
% qconf -spl
all.q
```

The queue all.q is set up by default in Sun Grid Engine.

- To configure the orte PE from the example in the previous section to the existing queue, type the following:

```
% qconf -mattr queue pe_list *orte* all.q
```

You must have Sun Grid Engine manager or operator privileges to use this command.

Submitting Jobs Under Sun Grid Engine Integration

There are two ways to submit jobs under Sun Grid Engine integration: interactive mode and batch mode. The instructions in this section describe how to submit jobs in batch mode. For information about how to use interactive mode, see Chapter 5.

▼ To Set the Interactive Display

Before you submit a job, you must have your DISPLAY environment variable set so that the interactive window will appear on your desktop, if you have not already done so.

For example, if you are working in the C shell, type the following command:

```
% setenv DISPLAY desktop:0.0
```
To Submit Jobs in Batch Mode

**Note** – Before you can use the parallel environment, make sure that you have set it up before running the job. See “Defining Parallel Environment (PE) and Queue” on page 52 for more information.

1. Create the script. In this example, `mpirun` is embedded within a script to `qsub`.

    ```bash
    mynode4% cat SGE.csh
    #!/usr/bin/csh

    # set PATH: including location of MPI program to be run
    setenv PATH /opt/SUNWhpc/HPC8.1/examples/connectivity:${PATH}

    mpirun -np 4 -mca ras_gridengine_verbose 100 connectivity.sparc -v
    ```

    **Note** – The `--mca ras_gridengine_verbose 100` setting is used in this example only to show that Sun Grid Engine is being used. This would not be needed for normal operation.

2. Next, source the Sun Grid Engine environment variables from a `settings.csh` file where `$SGE_ROOT` is set to `/opt/sgf`.

    ```bash
    % source $SGE_ROOT/default/common/settings.csh
    ```

3. To start the batch (or scripted) job, specify the parallel environment, slot number and the user executable.

    ```bash
    % qsub -pe orte 2 sge.csh
    your job 305 ("sge.csh") has been submitted
    ```

Since this is submitted as a batch job, you would not expect to see output at the terminal. If no indication is given for where the output should go, Sun Grid Engine redirects to your home directory and creates `<job_name>.o<job_number>`.

The job creates the output files. The file name with the format `name_of_job.ojob_id` contains the standard output. The file name with the format `name_of_job.ejob_id` contains the error output. If the job executes normally, the error output files will be empty.
The following example lists the files produced by a job called *sge.csh* with the job ID number 866:

```
% ls -rlt ~ | tail
-rw-r--r-- 1 joeuser mygroup 0 Jan 16 16:42 sge.csh.po866
-rw-r--r-- 1 joeuser mygroup 0 Jan 16 16:42 sge.csh.pe866
-rw-r--r-- 1 joeuser mygroup 0 Jan 16 16:42 sge.csh.e866
-rw-r--r-- 1 joeuser mygroup 194 Jan 16 16:42 sge.csh.o866
```

By default, the output files are located in your home directory, but you can use Sun Grid Engine software to change the location of the files, if desired.

**Note** – In most cases, you do not need to change the values set in the *gridengine MCA* parameters. If you run into difficulty and want to change the values for debugging purposes, the option is available. For more information about MCA parameters, see Chapter 7.

▼ To See a Running Job

● Type the following command:

```
% qstat -f
```

▼ To Delete a Running Job

● Type the following command:

```
% qdel job-number
```

where *job-number* is the number of the job you want to delete.

For more information about Sun Grid Engine commands, refer to the Sun Grid Engine documentation.

*rsh* Limitations

**Note** – This issue affects both *rsh* and the Sun Grid Engine program *qrsh*. *qrsh* uses *rsh* to launch jobs.
If you are using `rsh` or `qrsh` as the job launcher on a large cluster with hundreds of nodes, `rsh` might show the following error messages when launching jobs on the remote nodes:

```
rcmd: socket: Cannot assign requested address
rcmd: socket: Cannot assign requested address
rcmd: socket: Cannot assign requested address
[node0:00749] ERROR: A daemon on node m2187 failed to start as expected.
[node0:00749] ERROR: There may be more information available from
[node0:00749] ERROR: the ’qstat -t’ command on the Grid Engine tasks.
[node0:00749] ERROR: If the problem persists, please restart the
[node0:00749] ERROR: Grid Engine PE job
```

This indicates that `rsh` is running out of sockets when launching the job from the head node.

**Using rsh as the Job Launcher**

If you are using `rsh` as your job launcher, use `ssh` instead. Add the following to your command line:

```
-mca plm_rsh_agent ssh
```

**Using Sun Grid Engine as the Job Launcher**

If you are using Sun Grid Engine version 6.1 or earlier as your job launcher, you can modify the Sun Grid Engine configuration to allow Sun Grid Engine to use `ssh` instead of `rsh` to launch tasks on the remote nodes. The following web site describes how to perform this workaround:

http://gridengine.sunsource.net/howto/qrsh_qlogin_ssh.html

Note that this workaround does not properly track resource usage, nor does it allow proper job accounting. Sun Grid Engine tracks resource usage by attaching an extra groupid when launching tasks as a user of the remote connection.

Sun Grid Engine version 6.2 fixes this issue by not using `rsh` to start jobs on remote nodes. Instead Sun Grid Engine version 6.2 makes use of a native Interactive Job Support (IJS), which removes any dependencies on `rsh`, `ssh`, or `telnet`. It is recommended that you upgrade to the latest available version of Sun Grid Engine.
For More Information

For more information about using the `mpirun` command to perform batch processing, see the following:

- `mpirun(3)` man page
- Running MPI Jobs section of the Open MPI FAQ at [http://www.open-mpi.org/faq/?category=running#plm-available](http://www.open-mpi.org/faq/?category=running#plm-available)
Open MPI uses Modular Component Architecture (MCA) parameters to provide a way to tune your runtime environment. Each parameter corresponds to a specific function. You change the value of the parameter in order to change the function. Appendix B contains the complete list of MCA parameters.

Developing an Open MPI application that uses MCA parameters poses a number of advantages. Developers and administrators can customize the Open MPI environment to suit the specific needs of hardware or the operating environment. For example, a system administrator might use MCA parameters to optimize an Open MPI installation on a network so that users only need to run with the default values to obtain the best performance.

This chapter contains the following topics:

- “About the Modular Component Architecture” on page 60
- “Open MPI Frameworks” on page 60
- “The ompi_info Command” on page 63
- “Using MCA Parameters” on page 72
- “For More Information” on page 83

In order to understand how MCA parameters fit within Open MPI, you must understand how the Modular Component Architecture is constructed.
About the Modular Component Architecture

The Modular Component Architecture (MCA) is the backbone for much of Open MPI’s functionality. It is a series of frameworks, components, and modules that are assembled at runtime to create an MPI implementation.

An MCA framework manages a specific Open MPI task (such as process launching for ORTE). Each MCA framework supports a single component type, but can support multiple versions of that type. The framework uses the services from the MCA base functionality to find and/or load components.

An MCA component is an implementation of a framework’s interface. It is a standalone collection of code that can be bundled into a plug-in that can be inserted into the Open MPI code base, either at runtime and/or at compile time.

An MCA module is an instance of a component. For example, if a node running an Open MPI application has multiple Ethernet NICs, the Open MPI application will contain one TCP MPI point-to-point component, but two TCP point-to-point modules.

For more information about the Open MPI Modular Component Architecture, see the Open MPI FAQ on runtime tuning at:

http://www.open-mpi.org/faq/?category=tuning

Open MPI Frameworks

There are three types of frameworks in Open MPI:

- In the MPI layer (OMPI)
- In the run-time layer (ORTE)
- In the operating system/platform layer (OPAL)

You might think of these frameworks as ways to group MCA parameters by function. For example, the OMPI btl framework controls the functions in the byte transfer layer, or BTL (point-to-point byte movement) in the network. All of the MCA parameters that are grouped under btl affect the BTL layer.
In addition to the parameters that are grouped under the individual frameworks, there are top-level MCA parameters that affect the frameworks themselves and specify values to your Open MPI installation.

**TABLE 7-1** Top-Level MCA Parameters

<table>
<thead>
<tr>
<th>Parameter Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mca</td>
<td>Specify paths or functions for MCA parameters</td>
</tr>
<tr>
<td>mpi</td>
<td>Specify MPI behavior at runtime</td>
</tr>
<tr>
<td>orte</td>
<td>Specify debugging functions and components for ORTE</td>
</tr>
<tr>
<td>opal</td>
<td>Specify stack trace information</td>
</tr>
</tbody>
</table>

To view the available top-level parameters in each group, type the following command:

```bash
% ompi_info --param groupname groupname
```

where `groupname` stands for the parameter group you want to view. For example, to view the available MPI parameters, you would type:

```bash
% ompi_info --param mpi mpi
```

**OMPI Frameworks**

The following table lists the frameworks in the MPI layer.

**TABLE 7-2** OMPI Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocator</td>
<td>Memory allocator</td>
</tr>
<tr>
<td>bml</td>
<td>BTL management layer (managing multiple devices)</td>
</tr>
<tr>
<td>btl</td>
<td>Byte transfer layer (point-to-point byte movement)</td>
</tr>
<tr>
<td>coll</td>
<td>MPI collective algorithms</td>
</tr>
<tr>
<td>io</td>
<td>MPI-2 I/O functionality</td>
</tr>
<tr>
<td>mpool</td>
<td>Memory pool management</td>
</tr>
<tr>
<td>mtl</td>
<td>Messaging transport layer</td>
</tr>
<tr>
<td>osc</td>
<td>One-sided communication</td>
</tr>
</tbody>
</table>
Currently, there is no simple way to get a list of the available components in a framework. You can use the `grep` command to search for components. For example, the following command searches for a list of components in the `btl` framework:

```
% ompi_info | grep btl
```

### ORTE Frameworks

The following table lists the ORTE frameworks.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errmgr</td>
<td>Error manager</td>
</tr>
<tr>
<td>gpr</td>
<td>General purpose registry</td>
</tr>
<tr>
<td>iof</td>
<td>I/O forwarding</td>
</tr>
<tr>
<td>ns</td>
<td>Name server</td>
</tr>
<tr>
<td>oob</td>
<td>Out-of-band communication</td>
</tr>
<tr>
<td>plm</td>
<td>Process launch module (was pls).</td>
</tr>
<tr>
<td>ras</td>
<td>Resource allocation subsystem</td>
</tr>
<tr>
<td>rds</td>
<td>Resource discovery subsystem</td>
</tr>
<tr>
<td>rmaps</td>
<td>Resource mapping subsystem</td>
</tr>
<tr>
<td>rmgr</td>
<td>Resource manager (upper meta layer for all other Resource frameworks)</td>
</tr>
<tr>
<td>rml</td>
<td>Remote messaging layer (routing of OOB messages)</td>
</tr>
<tr>
<td>schema</td>
<td>Name schemas</td>
</tr>
<tr>
<td>sds</td>
<td>Startup discovery services</td>
</tr>
<tr>
<td>soh</td>
<td>State of health</td>
</tr>
</tbody>
</table>
OPAL Frameworks

The following table lists the OPAL frameworks.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtrace</td>
<td>Stack trace framework for debugging</td>
</tr>
<tr>
<td>maffinity</td>
<td>Memory affinity</td>
</tr>
<tr>
<td>memory</td>
<td>Memory hooks</td>
</tr>
<tr>
<td>paffinity</td>
<td>Processor affinity</td>
</tr>
<tr>
<td>timer</td>
<td>High-resolution timers</td>
</tr>
</tbody>
</table>

A complete list of MCA parameters, grouped under each of these frameworks, appears in Appendix B.

The `ompi_info` Command

The `ompi_info` command returns information about your Sun HPC ClusterTools/Open MPI installation. When you issue the command without any modifiers, `ompi_info` returns the following information:

- Revision information for Open MPI, ORTE, and OPAL
- Installed compilers
- Architecture of the node on which Open MPI is installed
- Version information for the installed frameworks
Command Options

The `ompi_info` command has the following options:

### TABLE 7-5   Options for the `ompi_info` Command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-a</code> or <code>--all</code></td>
<td>Shows all configuration options and MCA parameters</td>
</tr>
<tr>
<td><code>-c</code> or <code>--config</code></td>
<td>Shows configuration options</td>
</tr>
<tr>
<td><code>-gmca</code> or</td>
<td>Passes global MCA parameters that apply to all contexts. <code>param-name</code> is the parameter name; <code>value</code> is the value of the parameter</td>
</tr>
<tr>
<td><code>--gmca param-name value</code></td>
<td></td>
</tr>
<tr>
<td><code>-h</code> or <code>--help</code></td>
<td>Shows the <code>ompi_info</code> help message</td>
</tr>
<tr>
<td><code>--hostname</code></td>
<td>Shows the name of the host on which Open MPI was configured and built</td>
</tr>
<tr>
<td><code>--internal</code></td>
<td>Shows internal MCA parameters (not meant to be modified by users)</td>
</tr>
<tr>
<td><code>-mca</code> or <code>--mca</code></td>
<td>Passes context-specific MCA parameters; they are considered global if <code>--gmca</code> is not used. <code>param-name</code> is the name of the parameter; <code>value</code> is the value for that parameter.</td>
</tr>
<tr>
<td><code>--param arg1 arg2</code></td>
<td>Shows MCA parameters. <code>arg1</code> can be a specific framework name or all. <code>arg2</code> can be a specific parameter name or all.</td>
</tr>
<tr>
<td><code>--parsable or </code></td>
<td>Displays output in parsable format</td>
</tr>
<tr>
<td><code>--parseable</code></td>
<td></td>
</tr>
<tr>
<td><code>--path pathname</code></td>
<td>Shows the paths with which Open MPI was configured.</td>
</tr>
<tr>
<td><code>--pretty</code></td>
<td>Displays output in “prettyprint” format (default)</td>
</tr>
<tr>
<td><code>-v</code> or <code>--version arg0 arg1</code></td>
<td>Shows version of Open MPI or a component. <code>arg0</code> can be the name of a specific framework or all. <code>arg1</code> can be the name of a specific component or all.</td>
</tr>
</tbody>
</table>

The output from the `ompi_info` command appears similar to the following:

```bash
% omni_info
Open MPI: 1.2r13978-ct7b027r1708
Open MPI SVN revision: 0
```
Open RTE: 1.2r13978-ct7b027r1708
Open RTE SVN revision: 0
OPAL: 1.2r13978-ct7b027r1708
OPAL SVN revision: 0
Prefix: /opt/SUNWphc/HPC8.1

Configured architecture: i386-pc-solaris2.10
  Configured by: root
  Configured on: Thu Mar  8 16:47:40 EST 2008
  Configure host: burpen-csx10-0
    Built by: root
    Built on: Thu Mar  8 17:04:51 EST 2008
    Built host: burpen-csx10-0
  C bindings: yes
  C++ bindings: yes
  Fortran77 bindings: yes (all)
  Fortran90 bindings: yes
  Fortran90 bindings size: trivial
    C compiler: cc
    C compiler absolute: /ws/ompi-tools/SUNWspro/SOS12/bin/cc
    C++ compiler: CC
    C++ compiler absolute: /ws/ompi-tools/SUNWspro/SOS12/bin/CC
    Fortran77 compiler: f77
    Fortran77 compiler abs: /ws/ompi-tools/SUNWspro/SOS12/bin/f77
    Fortran90 compiler: f95
    Fortran90 compiler abs: /ws/ompi-tools/SUNWspro/SOS12/bin/f95
    C profiling: yes
    C++ profiling: yes
    Fortran77 profiling: yes
    Fortran90 profiling: yes
    C++ exceptions: yes
    Thread support: no
    Internal debug support: no
    MPI parameter check: runtime
    Memory profiling support: no
    Memory debugging support: no
    libltdl support: yes
    Heterogeneous support: yes

mpirun default --prefix: yes
  MCA backtrace: printstack (MCA v1.0, API v1.0, Component v1.2)
  MCA paffinity: solaris (MCA v1.0, API v1.0, Component v1.2)
    MCA maffinity: first_use (MCA v1.0, API v1.0, Component v1.2)
    MCA timer: solaris (MCA v1.0, API v1.0, Component v1.2)
    MCA allocator: basic (MCA v1.0, API v1.0, Component v1.0)
    MCA allocator: bucket (MCA v1.0, API v1.0, Component v1.0)
      MCA alloc: basic (MCA v1.0, API v1.0, Component v1.2)
      MCA alloc: self (MCA v1.0, API v1.0, Component v1.2)
      MCA alloc: sm (MCA v1.0, API v1.0, Component v1.2)
      MCA alloc: tuned (MCA v1.0, API v1.0, Component v1.2)
      MCA io: romio (MCA v1.0, API v1.0, Component v1.2)
| MCA mpool: sm  (MCA v1.0, API v1.0, Component v1.2) |
| MCA mpool: udapl (MCA v1.0, API v1.0, Component v1.2) |
| MCA pml: cm   (MCA v1.0, API v1.0, Component v1.2) |
| MCA pml: obl  (MCA v1.0, API v1.0, Component v1.2) |
| MCA bml: r2   (MCA v1.0, API v1.0, Component v1.2) |
| MCA rcache: rb (MCA v1.0, API v1.0, Component v1.2) |
| MCA rcache: vma (MCA v1.0, API v1.0, Component v1.2) |
| MCA bt1: self (MCA v1.0, API v1.0, Component v1.2) |
| MCA bt1: sm   (MCA v1.0, API v1.0, Component v1.2) |
| MCA bt1: tcp  (MCA v1.0, API v1.0, Component v1.2) |
| MCA bt1: udapl (MCA v1.0, API v1.0, Component v1.2) |
| MCA topo: unity (MCA v1.0, API v1.0, Component v1.2) |
| MCA osc: pt2pt (MCA v1.0, API v1.0, Component v1.2) |
| MCA errmgr: hnp (MCA v1.0, API v1.3, Component v1.2) |
| MCA errmgr: orted (MCA v1.0, API v1.3, Component v1.2) |
| MCA errmgr: proxy (MCA v1.0, API v1.3, Component v1.2) |
| MCA gpr: null (MCA v1.0, API v1.0, Component v1.2) |
| MCA gpr: proxy (MCA v1.0, API v1.0, Component v1.2) |
| MCA gpr: replica (MCA v1.0, API v1.0, Component v1.2) |
| MCA iof: proxy (MCA v1.0, API v1.0, Component v1.2) |
| MCA iof: svc  (MCA v1.0, API v1.0, Component v1.2) |
| MCA ns: proxy (MCA v1.0, API v2.0, Component v1.2) |
| MCA ns: replica (MCA v1.0, API v2.0, Component v1.2) |
| MCA oob: tcp  (MCA v1.0, API v1.0, Component v1.0) |
| MCA ras: dash_host (MCA v1.0, API v1.3, Component v1.2) |
| MCA ras: gridengine (MCA v1.0, API v1.3, Component v1.2) |
| MCA ras: localhost (MCA v1.0, API v1.3, Component v1.2) |
| MCA ras: tm   (MCA v1.0, API v1.3, Component v1.2) |
| MCA rds: hostfile (MCA v1.0, API v1.3, Component v1.2) |
| MCA rds: proxy (MCA v1.0, API v1.3, Component v1.2) |
| MCA rds: resfile (MCA v1.0, API v1.3, Component v1.2) |
| MCA rmaps: round_robin (MCA v1.0, API v1.3, Component v1.2) |
| MCA rmgr: proxy (MCA v1.0, API v2.0, Component v1.2) |
| MCA rmgr: urm  (MCA v1.0, API v2.0, Component v1.2) |
| MCA rml: oob  (MCA v1.0, API v1.0, Component v1.2) |
| MCA plm: gridengine (MCA v1.0, API v1.3, Component v1.2) |
| MCA plm: proxy (MCA v1.0, API v1.3, Component v1.2) |
| MCA plm: rsh  (MCA v1.0, API v1.3, Component v1.2) |
| MCA plm: tm   (MCA v1.0, API v1.3, Component v1.2) |
| MCA sds: env  (MCA v1.0, API v1.0, Component v1.2) |
| MCA sds: pipe (MCA v1.0, API v1.0, Component v1.2) |
| MCA sds: seed (MCA v1.0, API v1.0, Component v1.2) |
| MCA sds: singleton (MCA v1.0, API v1.0, Component v1.2) |
Using the *ompi_info* Command With MCA Parameters

The *ompi_info* command can list the parameters for a given component, all the parameters for a specific framework, or all parameters. The *ompi_info* output for most parameters contains a description of the parameter. The output for any parameter shows the current value of that parameter.

▼ To List All MCA Parameters

● Type the following command at the system prompt:

```
% ompi_info --param all all
```

The output from *ompi_info* lists all of the installed frameworks, their MCA parameters, and their current values. To see the complete list of MCA parameters, see “List of Available MCA Parameters” on page 111.

▼ To List All MCA Parameters For a Framework

● Type the following command at the system prompt:

```
% ompi_info --param btl all
```

In this example, the command lists all of the available MCA parameters for the *btl* framework. The output from *ompi_info* looks similar to the following:

```
MCA btl: parameter "btl_base_debug" (current value: "0")
  If btl_base_debug is 1 standard debug is output, if > 1 verbose debug is output
MCA btl: parameter "btl" (current value: <none>)
  Default selection set of components for the btl framework (<none> means "use all components that can be found")
MCA btl: parameter "btl_base_verbose" (current value: "0")
  Verbosity level for the btl framework (0 = no verbosity)
MCA btl: parameter "btl_self_free_list_num" (current value: "0")
  Number of fragments by default
MCA btl: parameter "btl_self_free_list_max" (current value: ":-1")
  Maximum number of fragments
```
MCA btl: parameter "btl_self_free_list_inc" (current value: "32")
    Increment by this number of fragments
MCA btl: parameter "btl_self_eager_limit" (current value: "131072")
    Eager size fragment (before the rendez-vous protocol)
MCA btl: parameter "btl_self_min_send_size" (current value: "262144")
    Minimum fragment size after the rendez-vous
MCA btl: parameter "btl_self_max_send_size" (current value: "262144")
    Maximum fragment size after the rendez-vous
MCA btl: parameter "btl_self_min_rdma_size" (current value: "2147483647")
    Maximum fragment size for the RDMA transfer
MCA btl: parameter "btl_self_max_rdma_size" (current value: "2147483647")
    Maximum fragment size for the RDMA transfer
MCA btl: parameter "btl_self_exclusivity" (current value: "65536")
    Device exclusivity
MCA btl: parameter "btl_self_flags" (current value: "10")
    Active behavior flags
MCA btl: parameter "btl_self_priority" (current value: "0")
MCA btl: parameter "btl_sm_free_list_num" (current value: "8")
MCA btl: parameter "btl_sm_free_list_max" (current value: "-1")
MCA btl: parameter "btl_sm_free_list_inc" (current value: "64")
MCA btl: parameter "btl_sm_exclusivity" (current value: "65535")
MCA btl: parameter "btl_sm_latency" (current value: "100")
MCA btl: parameter "btl_sm_max_procs" (current value: "-1")
MCA btl: parameter "btl_sm_extra_procs" (current value: "2")
MCA btl: parameter "btl_sm_mpool" (current value: "sm")
MCA btl: parameter "btl_sm_eager_limit" (current value: "4096")
MCA btl: parameter "btl_sm_max_frag_size" (current value: "32768")
MCA btl: parameter "btl_sm_size_of_cb_queue" (current value: "128")
MCA btl: parameter "btl_sm_cb_lazy_free_freq" (current value: "120")
MCA btl: parameter "btl_sm_priority" (current value: "0")
MCA btl: parameter "btl_tcp_if_include" (current value: <none>)
MCA btl: parameter "btl_tcp_if_exclude" (current value: "lo")
MCA btl: parameter "btl_tcp_free_list_num" (current value: 
MCA btl: parameter "btl_tcp_free_list_max" (current value: "-1")
MCA btl: parameter "btl_tcp_free_list_inc" (current value: "32")
MCA btl: parameter "btl_tcp_sndbuf" (current value: "131072")
MCA btl: parameter "btl_tcp_rcvbuf" (current value: "131072")
MCA btl: parameter "btl_tcp_endpoint_cache" (current value: "30720")
MCA btl: parameter "btl_tcp_exclusivity" (current value: "0")
MCA btl: parameter "btl_tcp_eager_limit" (current value: "65536")
MCA btl: parameter "btl_tcp_min_send_size" (current value: "65536")
MCA btl: parameter "btl_tcp_max_send_size" (current value: "131072")
MCA btl: parameter "btl_tcp_min_rdma_size" (current value: "131072")
MCA btl: parameter "btl_tcp_max_rdma_size" (current value: "2147483647")
MCA btl: parameter "btl_tcp_flags" (current value: "122")
MCA btl: parameter "btl_tcp_priority" (current value: "0")
MCA btl: parameter "btl_udapl_free_list_num" (current value: "8")
  Initial size of free lists (must be >= 1).
MCA btl: parameter "btl_udapl_free_list_max" (current value: "-1")
  Maximum size of free lists (-1 = infinite, otherwise must be >= 1).
MCA btl: parameter "btl_udapl_free_list_inc" (current value: "8")
  Increment size of free lists (must be >= 1).
MCA btl: parameter "btl_udapl_mpool" (current value: "udapl")
  Name of the memory pool to be used.
MCA btl: parameter "btl_udapl_max_modules" (current value: "8")
  Maximum number of supported HCAs.
MCA btl: parameter "btl_udapl_num_recv" (current value: "8")
  Total number of receive buffers to keep posted per endpoint (must be >= 1).
MCA btl: parameter "btl_udapl_num_sends" (current value: "7")
  Maximum number of sends to post on an endpoint (must be >= 1).
MCA btl: parameter "btl_udapl_sr_win" (current value: "4")
  Window size at which point an explicit credit message will be generated (must be >= 1).
MCA btl: parameter "btl_udapl_eager_rdma_num" (current value: "32")
  Number of RDMA buffers to allocate for small messages.
MCA btl: parameter "btl_udapl_max_eager_rdma_peers" (current value: "16")
Maximum number of peers allowed to use RDMA for short messages (indeedepndantly RDMA will still be used for large messages, (must be >= 0; if zero then RDMA will not be used for short messages).

MCA btl: parameter "btl_udapl_eager_rdma_win" (current value: "28")
Window size at which point an explicit credit message will be generated (must be >= 1).

MCA btl: parameter "btl_udapl_timeout" (current value: "1000000")
Connection timeout, in microseconds.

MCA btl: parameter "btl_udapl_conn_priv_data" (current value: "1")
Use connect private data to establish connections (not supported by all uDAPL implementations).

MCA btl: parameter "btl_udapl_async_events" (current value: "1000000000")
The asynchronous event queue will only be checked after entering progress this number of times.

MCA btl: parameter "btl_udapl_buffer_alignment" (current value: "256")
Preferred communication buffer alignment, in bytes (must be >= 1).

MCA btl: parameter "btl_udapl_evd_qlen" (current value: "256")
The event dispatcher queue length is a function of the number of connections as well as the maximum number of outstanding data transfer operations.

MCA btl: parameter "btl_udapl_max_request_dtos" (current value: "44")
Maximum number of outstanding submitted sends and rdma operations per endpoint, (see Section 6.6.6 of uDAPL Spec.).

MCA btl: parameter "btl_udapl_max_recv_dtos" (current value: "8")
Maximum number of outstanding submitted receive operations per endpoint, (see Section 6.6.6 of uDAPL Spec.).

MCA btl: parameter "btl_udapl_exclusivity" (current value: "1014")
uDAPL BTL exclusivity (must be >= 0).

MCA btl: parameter "btl_udapl_eager_limit" (current value: "8192")
Eager send limit, in bytes (must be >= 1).

MCA btl: parameter "btl_udapl_min_send_size" (current value: "16384")
Minimum send size, in bytes (must be >= 1).

MCA btl: parameter "btl_udapl_max_send_size" (current value: "65536")
To Display All MCA Parameters For a Selected Component

- Type the following command at the system prompt:

```bash
% ompi_info --param btl tcp
```

The `ompi_info` output looks similar to the following:

```
MCA btl: parameter "btl_base_debug" (current value: "0")
If btl_base_debug is 1 standard debug is output, if > 1 verbose debug is output
MCA btl: parameter "btl" (current value: <none>)
Default selection set of components for the btl framework (<none> means "use all components that can be found")
MCA btl: parameter "btl_base_verbose" (current value: "0") Verbosity level for the btl framework (0 = no verbosity)
MCA btl: parameter "btl_tcp_if_include" (current value: <none>)
MCA btl: parameter "btl_tcp_if_exclude" (current value: "lo")
MCA btl: parameter "btl_tcp_free_list_num" (current value:
```
Using MCA Parameters

There are three ways to use MCA parameters with Open MPI:

1. Setting the parameter from the command line using the `mpirun --mca` command. This method assumes the highest precedence; values set for parameters using this method override any other values specified for the same parameter.

2. Using the parameter as an environment variable. Values for parameters set in this fashion assume the next highest priority.

3. Setting the parameter values in a text file. Parameter values specified using this method have the lowest priority.
To Set MCA Parameters From the Command Line

- Type the following command at the system prompt:

```
% mpirun --mca param-name value
```

In this example, `param-name` stands for the name of the MCA parameter you want to set, and `value` stands for the new value you want to specify for the parameter. For example, the following command sets the value of the `mpi_show_handleLeaks` parameter to 1 for the specified job:

```
% mpirun --mca mpi_show_handleLeaks 1 -np 4 a.out
```

This sets the value of MCA parameter `mpi_show_handleLeaks` to 1 before running the program `a.out` with four processes.

Using MCA Parameters As Environment Variables

As with other types of environment variables, the syntax for setting MCA parameters as environment variables varies with the type of command shell.

To Set MCA Parameters in the `sh` Shell

1. Type the following command at the prompt:

```
% OMPI_MCA_param-name=value
```

where `param-name` is the name of the MCA parameter you want to set, and `value` is the desired value for the parameter. For example, the following command sets the `mpi_show_handleLeaks` parameter to 1:

```
% OMPI_MCA_mpi_show_handleLeaks=1
```
2. Type the following command:

```
% export OMPI_MCA_param-name
```

For example, an export command using the parameter used in the previous step would look like this:

```
% export OMPI_MCA_mpi_show_handleLeaks
```

3. Issue the `mpirun` command with the desired options. For example:

```
% mpirun -np 4 a.out
```

▼ To Set MCA Parameters in the C Shell

1. Use the `setenv` command to set the MCA parameter.

```
% setenv OMPI_MCA_param-name value
```

   where `param-name` is the name of the MCA parameter you want to set, and `value` is the desired value for the parameter. The following example shows how to set the `mpi_show_handle_leaks` parameter to 1.

```
% setenv OMPI_MCA_mpi_show_handle_leaks 1
```

2. Issue the `mpirun` command for the program (in this example, `a.out`).

```
% mpirun -np 4 a.out
```
To Specify MCA Parameters Using a Text File

1. Create a text file, specifying each parameter/value pair on a separate line. Comments are allowed. For example:

```
# This is a comment
# Set the same MCA parameter as in previous examples
mpi_show_handle_leaks = 1

# Default to rsh always
plm_rsh_agent = rsh

mpi_preconnect_all = 1
mpi_param_check = 0
#
# udapl parameters - comment or uncomment as needed
#
# btl = self,tcp,sm
# btl = self,udapl,sm
btl = ^tcp
```

2. Name the file `mca-params.conf` and save it.

You can save the file either to your home directory under
$HOME/.openmpi/mca-params.conf, where the parameter values in the file will only affect your jobs, or you can save it to
/opt/SUNWhpcc/HPC8.1/lib/openmpi-mca-params.conf, where the parameter values in the file affect all users.

The following example shows the output from the `ompi_info` command for `mca_param_files`.

```
% ompi_info --param mca mca_param_files
MCA mca: parameter "mca_param_files" (current value: 
"/home/joeuser/.openmpi/mca-params.conf:
/opt/SUNWhpcc/HPC8.1/etc/openmpi-mca-params.conf")
Path for MCA configuration files containing default parameter values
MCA mca: parameter "mca_component_path" (current value: 
"/opt/SUNWhpcc/HPC8.1/lib/openmpi:/home/joeuser/.openmpi/components")
Path where to look for Open MPI and ORTE components
MCA mca: parameter "mca_verbose" (current value: <none>)
Top-level verbosity parameter
MCA mca: parameter "mca_component_show_load_errors" (current value: "1")
Whether to show errors for components that failed to load or not
MCA mca: parameter "mca_component_disable_dlopen" (current value: "0")
Whether to attempt to disable opening dynamic components or not
```
The MCA parameter `mca_param_files` specifies a colon-delimited path of files to search for MCA parameters. Files to the left of the colon have lower precedence; files to the right of the colon have higher precedence. At runtime, `mpirun` searches the following two files in order when the `mca_param_files` parameter is set:

1. `$HOME/.openmpi/mca-params.conf`: The user-supplied set of values takes the highest precedence.
2. `$prefix/etc/openmpi-mca-params.conf`: The system-supplied set of values has a lower precedence.

In the above example, Open MPI first searches `/home/joeuser/.openmpi/mca-params.conf` for MCA parameters, and then searches `/opt/SUNW/hpc/HPC8.1/etc/openmpi-mca-params.conf`. If a parameter appears in both locations, the value set in the second file (the file to the right of the colon) is used.

### Including and Excluding Components

Each MCA framework has a top-level MCA parameter that you can use to select which components are to be used at runtime. In other words, there is an MCA parameter of the same name as each MCA framework (for example, `btl`) that you can use to include or exclude components from a given run.

You can use top-level parameters in the same way you would use other MCA parameters (for example, you can set them from the command line, as environment variables, or in text files).

For example, the `btl` MCA parameter is used to control which byte transfer layer (BTL) components are used with `mpirun`. The value for the `btl` parameter is a list of components separated by commas, with the optional prefix `^` (caret symbol).

**Note** – Do not mix “include” instructions with “exclude” instructions in the same command; otherwise, `mpirun` returns an error.
▼ To Include and Exclude Components Using the Command Line

- Type the following command at the system prompt:

```
% mpirun --mca framework comp1, comp2 ^comp3
```

In this example, the components *comp1* and *comp2* are included for the framework specified by --mca framework. Component *comp3* is excluded, since it is preceded by the ^ (caret) symbol.

For example, the following command excludes the *tcp* and *openib* components from the BTL framework, and implicitly includes all the other components:

```
% mpirun --mca btl ^tcp,openib ...
```

The use of the caret followed by the ellipsis in the command means “Perform the opposite action with the rest of the components.” When the `mpirun -- mca` command specifies components to be excluded, the caret followed by the ellipsis implicitly includes the rest of the components in that framework. When the `mpirun --mca` command specifically includes components, the caret followed by the ellipsis means “and exclude the components not specified.”

For example, the following command includes only the *self*, *sm*, and *gm* components of *btl* and implicitly excludes the rest:

```
% mpirun --mca btl self,sm,gm ...
```

Processor and Memory Affinity

Using Processor Affinity

The term *processor affinity* refers to the state where the operating system allows only that process to run on a specific processor. On multi-processor machines, this can help improve performance by not allowing the operating system to move processes between processors. This can eliminate the "jitter" from performance characteristics due to the OS moving processes, which means that performance characteristics should be consistent among multiple runs. This approach can dramatically improve performance.
**Note** – Processor affinity should not be used when a node is over-subscribed (that is, when more processes are launched than there are processors). This can lead to a serious degradation in performance (even more than simply oversubscribing the node). Open MPI usually detects this situation and automatically disables the use of processor affinity (and displays run-time warnings to this effect). For more information about oversubscribing nodes, see “Oversubscribing Nodes” on page 27.

### Using Memory Affinity

Memory affinity is only relevant for Non-Uniform Memory Access (NUMA) machines, such as many models of multi-processor Opteron™ machines. In a NUMA architecture, memory is physically distributed throughout the machine, even though it is virtually treated as a single address space. That is, memory may be physically local to one or more processors; therefore, the memory is remote to other processors. This means that some memory can be accessed more quickly by a process than other memory.

Open MPI supports general and specific memory affinity, which means that it generally tries to allocate all memory local to the processor that asked for it. When shared memory is used for communication, Open MPI uses memory affinity to make certain pages local to specific processes in order to minimize memory network/bus traffic.

#### To Find Out Whether Memory Affinity Is Supported

Open MPI supports memory affinity on a variety of systems.

- **To find out which systems are supported, type the `ompi_info` command and look for `maffinity` components to see if your system is supported. For example:**

  ```
  % ompi_info | grep maffinity
  MCA maffinity: first_use (MCA v1.0, API v1.0, Component v1.2)
  ```

**Note** – Memory affinity support is enabled only when processor affinity is enabled. This is because processes might allocate local memory and then move to a different processor, and the second processor might be remote from the memory that the process just allocated. This negates the purpose of specifying memory affinity.
Running MPI Jobs With Processor and Memory Affinity

If your system supports processor and memory affinity as shown using the *ompi_info* command, you can explicitly tell Open MPI to use affinity when running MPI jobs.

**Note** — Processor and memory affinity function only on multi-processor machines.

Currently, Open MPI only offers coarse-grained controls for processor affinity. For this reason, you can obtain the best results if processes in an Open MPI job using processor affinity are the only intensive processes running on the nodes being used for the job. Since most schedulers do not provide information on which processors should be used for specific processes, Open MPI assumes that its processes are “alone” on the node. Open MPI then exclusively claims CPUs, starting with the first one.

This means that if two processor-affinity-enabled jobs are running on the same node, they will both attempt to claim the first processor(s) on the node, resulting in CPU thrashing (and severely degraded performance).

**Note** — When running with processor affinity, all processors must be operational. Otherwise, processor affinity will not function because all the processors must be accessed in sequence.

▼ To Enable Affinity Using the Command Line

- To enable processor (and potentially memory) affinity, set the MCA parameter *opal_paffinity_alone* to 1.

For example, the following command enables processor affinity while running the program *a.out* on four processors:

```
% mpirun --mca opal_paffinity_alone 1 -np 4 a.out
```

The command shown in this example assumes that this job is running on a single 4-processor machine or two 2-processor machines. Setting *opal_paffinity_alone* tells Open MPI to bind each process to a specific processor. If memory affinity is supported, Open MPI also attempts to use memory affinity for this job.

You set values for *opal_paffinity_alone* in the same way you set other MCA parameters. For more information about setting MCA parameters, see “Using MCA Parameters” on page 72.
Note – Open MPI automatically disables processor affinity on any node that is oversubscribed (that is, where more Open MPI processes are launched in a single job on a node than it has processors) and returns warning messages. However, you may use processor affinity with degraded performance mode if the nodes are not oversubscribed.

Using MCA Parameters With Sun Grid Engine

The ras_gridengine parameters enable you to specify the output from the Open MPI RAS (Resource Allocation Subsystem). The rsh PLM (Process Launch Module) contains the gridengine parameters.

The following example shows the mpirun command with a specified MCA parameter.

```
% mpirun -np 4 -mca plm_gridengine_debug 100 connectivity.sparc -v
```

The following table shows the available MCA parameters and their default values.

<table>
<thead>
<tr>
<th>MCA Parameter</th>
<th>Default Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ras_gridengine_debug</td>
<td>0</td>
<td>Enable debugging output for the gridengine ras component</td>
</tr>
<tr>
<td>ras_gridengine_verbose</td>
<td>0</td>
<td>Enable verbose output for the gridengine ras component</td>
</tr>
<tr>
<td>ras_gridengine_show_jobid</td>
<td>0</td>
<td>Show the JOB_ID of the Grid Engine job</td>
</tr>
<tr>
<td>ras_gridengine_priority</td>
<td>100</td>
<td>Priority of the gridengine ras component</td>
</tr>
<tr>
<td>plm_base_reuse_daemons</td>
<td>0</td>
<td>Specifies whether to reuse daemons to launch dynamically spawned processes</td>
</tr>
<tr>
<td>plm_gridengine_debug</td>
<td>0</td>
<td>Enable debugging of gridengine plm component</td>
</tr>
<tr>
<td>plm_gridengine_verbose</td>
<td>0</td>
<td>Enable verbose output of the gridengine qsh -inherit command</td>
</tr>
<tr>
<td>plm_gridengine_priority</td>
<td>100</td>
<td>Priority of the gridengine plm component</td>
</tr>
<tr>
<td>plm_gridengine_orted</td>
<td>orted</td>
<td>The command name that the gridengine plm component will invoke for the ORTE daemon</td>
</tr>
</tbody>
</table>
To view a list of the RAS parameters from the command line, use the `ompi_info` command. The following example shows how to specify the RAS parameters and the output from the `ompi_info` command.

```bash
% ompi_info -param ras gridengine
MCA ras: parameter "ras" (current value: <none>)
  Default selection set of components for the ras
  framework (<none> means "use all components that can
  be found")
MCA ras: parameter "ras_gridengine_debug" (current value: "0")
  Enable debugging output for the gridengine ras
  component
MCA ras: parameter "ras_gridengine_priority" (current value: "100")
  Priority of the gridengine ras component
MCA ras: parameter "ras_gridengine_verbose" (current value: "0")
  Enable verbose output for the gridengine ras
  component
MCA ras: parameter "ras_gridengine_show_jobid" (current value: "0")
  Show the JOB_ID of the Grid Engine job
```
This example shows the output from the `ompi_info` command when the PLM parameters are specified:

```
% ompi_info -param plm gridengine
MCA plm: parameter "plm_base_reuse_daemons" (current value: "0")
   If nonzero, reuse daemons to launch dynamically
   spawned processes. If zero, do not reuse daemons
   (default)
MCA plm: parameter "plm" (current value: <none>)
   Default selection set of components for the plm
   framework (<none> means "use all components that can
   be found")
MCA plm: parameter "plm_base_verbose" (current value: "0")
   Verbosity level for the plm framework (0 = no
   verbosity)
MCA plm: parameter "plm_gridengine_debug" (current value: "0")
   Enable debugging of gridengine plm component
MCA plm: parameter "plm_gridengine_verbose" (current value: "0")
   Enable verbose output of the gridengine qrsh -inherit
   command
MCA plm: parameter "plm_gridengine_priority" (current value: "100")
   Priority of the gridengine plm component
MCA plm: parameter "plm_gridengine_orted" (current value: "orted")
   The command name that the gridengine plm component
   will invoke for the ORTE daemon
```

Changing the Default Values in MCA Parameters

**Note** – In most cases, you do not need to change the default values in the gridengine MCA parameters. If you encounter a difficulty and want to change the values for debugging purposes, the options are available.

There are options available in the MCA PLM and RAS components and modules to allow changes of the default values.

For more information about how to change the values in MCA parameters, see the General Run-time Tuning FAQ on the Open MPI Web site at:

http://www.open-mpi.org/faq/?category=tuning#setting-mca-params
For More Information

For more information about the Modular Component Architecture and MCA parameters, refer to the following sources:

- Chapter 5, “Running Programs With the `mpirun` Command” on page 19
- Appendix B, “List of Available MCA Parameters” on page 111
- Open MPI FAQ about runtime tuning: http://www.open-mpi.org/faq/?category=tuning
- The `ompi_info` man page
- The `ompi_info --help` command
Using the DTrace Utility With Open MPI

This chapter describes how to use the Solaris™ Dynamic Tracing (DTrace) utility with Open MPI. DTrace is a comprehensive dynamic tracing utility that you can use to monitor the behavior of applications programs as well as the operating system itself. You can use DTrace on live production systems to understand those systems' behavior and to track down any problems that might be occurring.

The D language is the programming language used to create the source code for DTrace programs.

The content of this chapter assumes knowledge of the D language and how to use DTrace.

The following topics are covered in this chapter:

■ “Checking the mpirun Privileges” on page 86
■ “Running DTrace with MPI Programs” on page 87
■ “Tracking Down ResourceLeaks” on page 91
■ “Using the DTrace mpiperuse Provider” on page 96

For more information about the D language and DTrace, refer to the Solaris Dynamic Tracing Guide (Part Number 817-6223). This guide is part of the Solaris 10 OS Software Developer Collection.
Solaris 10 OS documentation can be found on the web at the following location:

http://www.sun.com/documentation

Follow these links to the Solaris Dynamic Tracing Guide:
Solaris Operating Systems -> Solaris 10 -> Solaris 10 Software Developer Collection

**Note** – The programs and script mentioned in the sections that follow are located at:

/opt/SUNWhpc/examples/mpi/dtrace

---

### Checking the `mpirun` Privileges

Before you run a program under DTrace, you need to make sure that you have the correct `mpirun` privileges.

In order to run the script under `mpirun`, make sure that you have `dtrace_proc` and `dtrace_user` privileges. Otherwise, DTrace will return the following error because it does not have sufficient privileges:

```bash
mkdir: failed to initialize dtrace: DTrace requires additional privileges
```

**V** To Determine the Correct Privileges on the Cluster

To determine whether you have the appropriate privileges on the entire cluster, perform the following steps:

1. **Use your favorite text editor to create the following shell script, called `mpppriv.sh`:**

   ```bash
   #!/bin/sh
   # mpppriv.sh - run ppriv under a shell so you can get the privileges
   # of the process that mpirun creates
   ppriv $$
   ```

2. **Type the following command, replacing `host1` and `host2` with the names of hosts in your cluster:**

   ```bash
   % mpirun -np 2 --host host1,host2 mpppriv.sh
   ```
If the output of `ppriv` shows that the E privilege set has the `dtrace` privileges, then you will be able to run `dtrace` under `mpirun` (see the following two examples). Otherwise, you must adjust your system to get `dtrace` access.

The following example shows the output from `ppriv` when the privileges have not been set:

```
% ppriv $$
4084: -csh
flags = <none>
  E: basic
  I: basic
  P: basic
  L: all
```

This example shows `ppriv` output when the privileges have been set:

```
% ppriv $$
2075: tcsh
flags = <none>
  E: basic,dtrace_proc,dtrace_user
  I: basic,dtrace_proc,dtrace_user
  P: basic,dtrace_proc,dtrace_user
  L: all
```

**Note** – To update your privileges, ask your system administrator to add the `dtrace_user` and `dtrace_proc` privileges to your account in the `/etc/user_attr` file.

After the privileges have been changed, you can use the `ppriv` command to view the changed privileges.

## Running DTrace with MPI Programs

There are two ways to use dynamic tracing with MPI programs:

- Run the MPI program directly under DTrace
- Attach DTrace to a running MPI program
Running an MPI Program Under DTrace

For illustration purposes, assume you have a program named mpiapp.

▼ To Trace a Program Using the mpitrace.d Script

● Type the following command:

```bash
% mpirun -np 4 dtrace -s mpitrace.d -c mpiapp
```

The advantage of tracing an MPI program in this way is that all the processes in the job will be traced from the beginning. This method is probably most useful in doing performance measurements, when you need to start at the beginning of an application and you need all the processes in a job to participate in collecting data.

This approach also has some disadvantages. One disadvantage of running a program like the one in the above example is that all the tracing output for all four processes is directed to standard output (stdout). One way around this problem is to create a script similar to the script in the following section:

▼ To Trace a Parallel Program and Get Separate Trace Files

1. Create a shell script (called partrace.sh in this example) similar to the following:

```bash
#!/bin/sh
# partrace.sh - a helper script to dtrace Open MPI jobs from the
# start of the job.
dtrace -s $1 -c $2 -o $2.$OMPI_COMM_WORLD_RANK.trace
```

2. Type the following command to run the partrace.sh shell script:

```bash
% mpirun -np 4 partrace.sh mpitrace.d mpiapp
```

This will run mpiapp under dtrace using the mpitrace.d script. The script saves the trace output for each process in a job under a separate file name, based on the program name and rank of the process. Note that subsequent runs will append the data into the existing trace files.

Note – The status of the OMPI_COMM_WORLD_RANK.trace variable is unstable and subject to change. Use this variable with caution.
Attaching DTrace to a Running MPI Program

The second way to use dtrace with Open MPI is to attach dtrace to a running MPI program.

To Attach DTrace to a Running MPI Program

Perform the following procedure:

1. Log in to the node in which you are interested.

2. Type commands similar to the following command to get the process ID (PID) of the running program on the node of interest.

   ```
   % prstat 0 1 | grep mpiapp
   24768 joeuser  526M  3492K  sleep  59  0 0:00:08 0.1% mpiapp/1
   24770 joeuser  518M  3228K  sleep  59  0 0:00:08 0.1% mpiapp/1
   %
   ```

3. Decide which rank you want to use to attach dtrace.
   The lower PID number is usually the lower rank on the node.

4. Type the following command to attach to the rank 1 process (identified by its process ID, which is 24770 in the example) and run the DTrace script mpitrace.d:

   ```
   % dtrace -p 24770 -s mpitrace.d
   ```

Simple MPI Tracing

DTrace enables you to easily trace programs. When used in conjunction with MPI and the more than 200 functions defined in the MPI standard, DTrace provides an easy way to determine which functions might be in error during the debugging process, or those functions that might be of interest. After you determine the function showing the error, it is easy to locate the desired job, process, and rank on which to run your scripts. As demonstrated above, DTrace allows you to perform these determinations while the program is running.

Although the MPI standard provides the MPI profiling interface, using DTrace does provide a number of advantages. The advantages of using DTrace include the following:

- The PMPI interface requires you to restart a job every time you make changes to the interposing library.
DTrace allows you to define probes that let you capture tracing information on MPI without having to code the specific details for each function you want to capture.

The DTrace scripting language D has several built-in functions that help in debugging problematic programs.

The following example shows a simple script that traces the entry and exit into all the MPI API calls.

```d
mpitrace.d:
    pid$target:libmpi:MPI_*:entry
    {
      printf("Entered %s...", probefunc);
    }

    pid$target:libmpi:MPI_*:return
    {
      printf("exiting, return value = %d\n", arg1);
    }
```

When you use this example script to attach DTrace to a job that performs `send` and `recv` operations, the output looks similar to the following:

```
% dtrace -q -p 24770 -s mpitrace.d
Entered MPI_Send...exiting, return value = 0
Entered MPI_Rcv...exiting, return value = 0
Entered MPI_Send...exiting, return value = 0
Entered MPI_Rcv...exiting, return value = 0
Entered MPI_Send...exiting, return value = 0 ...
```
You can easily modify the `mpitrace.d` script to include an argument list. The resulting output resembles `truss` output. For example:

```d
mpitruss.d:
    pid$target:libmpi:MPI_Send:entry,  
    pid$target:libmpi:MPI_*send:entry,  
    pid$target:libmpi:MPI_Recv:entry,  
    pid$target:libmpi:MPI_*recv:entry  
    { 
      printf("%s(0x%x, %d, 0x%x, %d, %d, 0x%x)\n", probedesc, arg0, arg1, 
      arg2, arg3, arg4, arg5); 
    } 
    pid$target:libmpi:MPI_Send:return,  
    pid$target:libmpi:MPI_*send:return,  
    pid$target:libmpi:MPI_Recv:return,  
    pid$target:libmpi:MPI_*recv:return  
    { 
      printf("\t\t = %d\n", arg1); 
    }
```

The `mpitruss.d` script shows how you can specify wildcard names to match the functions. Both probes will match all send and receive type function calls in the MPI library. The first probe shows the usage of the built-in `arg` variables to print out the `arglist` of the function being traced.

Take care when wildcarding the entrypoint and the formatting argument output, because you could end up printing either too many arguments, or not enough arguments, for certain functions. For example, in the above case, the `MPI_Irecv` and `MPI_Isend` functions will not have their Request handle parameters printed out.

The following example shows a sample output of the `mpitruss.d` script:

```d
% dtrace -q -p 24770 -s mpitruss.d
MPI_Send(0x80470b0, 1, 0x8060f48, 0, 1,0x8060d48) = 0
MPI_Recv(0x80470a8, 1, 0x8060f48, 0, 0, 0x8060d48) = 0
MPI_Send(0x80470b0, 1, 0x8060f48, 0, 1, 0x8060d48) = 0
MPI_Recv(0x80470a8, 1,0x8060f48, 0, 0, 0x8060d48) = 0 ...
```

---

**Tracking Down Resource Leaks**

One of the biggest issues with programming is the unintentional leaking of resources (such as memory). With MPI, tracking and repairing resource leaks can be somewhat more challenging because the objects being leaked are in the middleware, and thus are not easily detected by the use of memory checkers.
DTrace helps with debugging such problems using variables, the profile provider, and a callstack function. The `mpicommcheck.d` script (shown in the example below) probes for all the MPI communicator calls that allocate and deallocate communicators, and keeps track of the stack each time the function is called. Every 10 seconds the script dumps out the current count of MPI communicator calls and the total calls for the allocation and deallocation of communicators. When the dtrace session ends (usually by pressing Ctrl-C, if you attached to a running MPI program), the script will print out the totals and all the different stack traces, as well as the number of times those stack traces were reached.

In order to perform these tasks, the script uses DTrace features such as variables, associative arrays, built-in functions (`count, ustack`) and the predefined variable `probefunc`. 
The following example shows the mpicommcheck.d script.

```c
mpicommcheck.d:
BEGIN
{
allocations = 0;
deallocations = 0;
prcnt = 0;
}

pid$target:libmpi:MPI_Comm_create:entry,
pid$target:libmpi:MPI_Comm_dup:entry,
pid$target:libmpi:MPI_Comm_split:entry
{
++allocations;
@counts[probefunc] = count();
@stacks[ustack()] = count();
}

pid$target:libmpi:MPI_Comm_free:entry
{
++deallocations;
@counts[probefunc] = count();
@stacks[ustack()] = count();
}

profile:::tick-1sec
//++prcnt > 10/
{
printf("=======================================================
==============");
printa(@counts);
printf("Communicator Allocations = %d \n", allocations);
printf("Communicator Deallocations = %d\n", deallocations);
prcnt = 0;
}

END
{
printf("Communicator Allocations = %d, Communicator\nDeallocations = %d\n", allocations, deallocations);
}
```

This script attaches dtrace to a suspect section of code in your program (that is, a section of code that might contain a resource leak). If, during the process of running the script, you see that the printed totals for allocations and deallocations are starting to steadily diverge, you might have a resource leak. Depending on how your
program is designed, it might take some time and observation of the allocation/deallocation totals in order to definitively determine that the code contains a resource leak. Once you do determine that a resource leak is definitely occurring, you can press Ctrl-C to break out of the dtrace session. Next, using the stack traces dumped, you can try to determine where the issue might be occurring.

The following example shows code containing a resource leak, and the output that is displayed using the mpicommcheck.d script.

The sample MPI program containing the resource leak is called mpicommleak. This program performs three MPI_Comm_dup operations and two MPI_Comm_free operations. The program thus “leaks” one communicator operation with each iteration of a loop.

When you attach dtrace to mpicommleak using the mpicommcheck.d script above, you will see a 10-second periodic output. This output shows that the count of the allocated communicators is growing faster than the count of deallocations.

When you finally end the dtrace session by pressing Ctrl-C, the session will have output a total of five stack traces, showing the distinct three MPI_Comm_dup and two MPI_Comm_free call stacks, as well as the number of times each call stack was encountered.
For example:

```
% prstat 0 1 | grep mpicommleak
24952 joeuser  518M  3212K sleep   59  0 0:00:01 1.8% mpicommleak/1
24950 joeuser  518M  3212K sleep   59  0 0:00:00 0.2% mpicommleak/1
% dtrace -q -p 24952 -s mpicommcheck.d

=====================================================================  
MPI_Comm_free 4                                               
MPI_Comm_dup 6                                               
Communicator Allocations = 6 
Communicator Deallocations = 4                              
=====================================================================  
MPI_Comm_free 8                                               
MPI_Comm_dup 12                                              
Communicator Allocations = 12 
Communicator Deallocations = 8                              
=====================================================================  
MPI_Comm_free 12                                              
MPI_Comm_dup 18                                              
Communicator Allocations = 18 
Communicator Deallocations = 12                              
^C
Communicator Allocations = 21, Communicator Deallocations = 14
```

```
libmpi.so.0.0.0 `MPI_Comm_free
  mpicommleak `deallocate_comms+0x19
  mpicommleak `main+0x6d
  mpicommleak `0x805081a
    7

libmpi.so.0.0.0 `MPI_Comm_free
  mpicommleak `deallocate_comms+0x26
  mpicommleak `main+0x6d
  mpicommleak `0x805081a
    7

libmpi.so.0.0.0 `MPI_Comm_dup
  mpicommleak `allocate_comms+0x1e
  mpicommleak `main+0x5b
  mpicommleak `0x805081a
    7

libmpi.so.0.0.0 `MPI_Comm_dup
  mpicommleak `allocate_comms+0x30
  mpicommleak `main+0x5b
  mpicommleak `0x805081a
    7

libmpi.so.0.0.0 `MPI_Comm_dup
  mpicommleak `allocate_comms+0x42
  mpicommleak `main+0x5b
  mpicommleak `0x805081a
    7
```
Using the DTrace mpiperuse Provider

PERUSE is an MPI interface that allows you to obtain detailed information about the performance and interactions of processes, software, and MPI. PERUSE provides a greater level of detail about process performance than does the standard MPI profiling interface (PMPI).

For more information about PERUSE and the current PERUSE specification, see:

http://www.mpi-peruse.org

Open MPI includes a DTrace provider named mpiperuse. This provider enables you to configure Open MPI to support DTrace probes into the Open MPI shared library libmpi.

DTrace Support in the ClusterTools Software

In Sun HPC ClusterTools 8.1 software, there are preconfigured executables and libraries with the mpiperuse provider probes built in. They are located in the /opt/SUNWhpc/HPC8.1/sun/instrument directory. Use the wrappers and utilities located in this directory to access the mpiperuse provider.

**Note** – No recompilation is necessary in order to use the mpiperuse provider. Just run the application to be DTraced using /opt/SUNWhpc/HPC8.1/sun/instrument/bin/mpirun.

Available mpiperuse Probes

The DTrace mpiperuse probes expose the events specified in the current PERUSE specification. These events track the life cycle of requests within the MPI library. For more information about this life cycle and the actual events provided by PERUSE, see Section 4 of the PERUSE Specification.

Sections 4.3.1 and 4.4 of the PERUSE Specification list and describe the individual events exposed by PERUSE.

The mpiperuse provider makes these events available to DTrace. The probe names correspond to the event names listed in Sections 4.3.1 and 4.4 of the PERUSE specification. For each event, the corresponding probe name is similar, except that
the leading PERUSE is removed, the probe name is all lowercase, and underscores are replaced with hyphens. For example, the probe for PERUSE_COMM_MSG_ARRIVED is comm-msg-arrived.

All of the probes are classified under the mpiperuse provider. This means that to find the probe names, you would look under the mpiperuse name. It also means that when you make a DTrace statement, you can include a wildcard for all probes simply by using the mpiperuse classification.

**Specifying an mpiperuse Probe in a D Script**

In the D scripting language, specifying an mpiperuse provider takes the following form:

```
mpiperuse$target:::probe-name
```

where `probe-name` is the name of the mpiperuse probe you want to use.

For example, to specify a probe to capture a PERUSE_COMM_REQ_ACTIVATE event, add the following line to a D script:

```
mpiperuse$target:::comm-req-activate
```

This alerts DTrace that you want to use the mpiperuse provider to capture the PERUSE_COMM_REQ_ACTIVATE event. In this example, the optional object and function fields in the probe description are omitted. This directs DTrace to find all occurrences of the comm-req-activate probes in the MPI library and its plugins instead of a specific probe. This is necessary because certain probes can happen in multiple places in the MPI library.

For more information about the D language and its syntax, refer to the *Solaris Dynamic Tracing Guide* (Part Number 817-6223). This guide is part of the Solaris 10 OS Software Developer Collection.
Available Arguments

All of the mpiperuse probes receive the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>args[0] = mpiconninfo_t *i</td>
<td>This provides a basic source and destination for the request and which protocol is expected to be used for the transfer. This typedef is defined in /usr/lib/dtrace/mpi.d.</td>
</tr>
<tr>
<td>args[1] = uintptr_t uid</td>
<td>This is the PERUSE unique id for the request that fired the probe (as defined by the PERUSE specifications). For OMPI this is the address of the actual request.</td>
</tr>
<tr>
<td>args[2] = uint_t op</td>
<td>This value indicates whether the probe is for a send == 0 or recv == 1 request.</td>
</tr>
<tr>
<td>args[3] = mpicomm_spec_t *cs</td>
<td>This structure is defined in /usr/lib/dtrace/mpi.d and mimics the spec structure, as defined on page 22 of the PERUSE specification.</td>
</tr>
</tbody>
</table>

How To Use mpiperuse Probes to See Message Queues

To use the mpiperuse provider, make reference to the appropriate mpiperuse provider probes and arguments in a DTrace script, as you would for any other provider (such as the pid provider).

The procedure for running scripts with mpiperuse probes follows the same steps as those shown in “Running an MPI Program Under DTrace” on page 88 and “Attaching DTrace to a Running MPI Program” on page 89, except that you must edit the partrace.sh script before you run it.

Change partrace.sh to include a -Z switch after the dtrace command, as shown in the following example.

```bash
#!/bin/sh
# partrace.sh - a helper script to dtrace Open MPI jobs from the
# start of the job.
dtrace -Z -s $1 -c $2 -o $2.$OMPI_COMM_WORLD_RANK.trace
```

This change allows probes that do not exist at initial load time to be used in a script (that is, the probes are in plugins that have not been dlopened).

The following example shows how to use the mpiperuse probes when running a DTrace script. Use the example script provided in /opt/SUNWhp/HPC8.1/sun/examples/dtrace/mpistat.d
1. Compile and run a script against a program.
   In this example, the script file is called \texttt{dtest.c}. Substitute the name and path of your script for \texttt{dtest.c}.

   \begin{verbatim}
% /opt/SUNWhpc/HPC8.1/sun/instrument/bin/mpicc -myhomedir/scraps/usdt/examples/dtest.c -o dtest
% /opt/SUNWhpc/HPC8.1/sun/instrument/bin/mpirun -np 2 dtest
Initing MPI...
Initing MPI...
Do communications...
Do communications...
attach to pid 13371 to test tracing.
   \end{verbatim}

2. In another window, type the following command:

\begin{verbatim}
% dtrace -q -p 13371 -s /opt/SUNWhpc/HPC8.1/sun/examples/dtrace/mpistat.d
   \texttt{input(Total)} & \texttt{Q-sizes} & \texttt{Q-Matches} & \texttt{output}
   \texttt{bytes} & \texttt{active} & \texttt{posted} & \texttt{unexp} & \texttt{posted} & \texttt{unexp} & \texttt{bytes} & \texttt{active} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 10 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 10 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 10 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 10 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 10 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 15 & 0
   \end{verbatim}
mpiperuse Usage Examples

The examples in this section show how to perform the described DTrace operations from the command line.

▼ To Count the Number of Messages To or From a Host

● Issue the following DTrace command, substituting the process ID of the process you want to monitor for `pid`:

```
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]->ci_remote] = count();}'
```

DTrace returns a result similar to the following. In this example, the process ID is 25428 and the host name is `joe-users-host2`.

```
% dtrace -p 25428 -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]->ci_remote] = count();}''
dtrace: description `mpiperuse$target:::comm-req-xfer-end ' matched 17 probes
^C
joe-users-host2 recv 3
joe-users-host2 send 3
```

▼ To Count the Number of Messages To or From Specific BTLs

● Issue the following DTrace command, substituting the process ID of the process you want to monitor for `pid`:

```
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]->ci_protocol] = count();}'
```

DTrace returns a result similar to the following. In this example, the process ID is 25445.

```
% dtrace -p 25445 -n 'mpiperuse$target:::comm-req-xfer-end {@[args[0]->ci_protocol] = count();}''
dtrace: description `mpiperuse$target:::comm-req-xfer-end ' matched 17 probes
^C
sm 60
```
To Obtain Distribution Plots of Message Sizes Sent or Received From a Host

- Issue the following DTrace command, substituting the process ID of the process you want to monitor for `pid`:

```bash
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end { @[args[0]->ci_remote] = quantize(args[3]->mcs_count); }'
```

DTrace returns a result similar to the following. In this example, the process ID is 25445.

```bash
% dtrace -p 25445 -n 'mpiperuse$target:::comm-req-xfer-end { @[args[0]->ci_remote] = quantize(args[3]->mcs_count); }'
dtrace: description 'mpiperuse$target:::comm-req-xfer-end ' matched 17 probes
^C
```

```
myhost
value  ------------- Distribution ------------- count
2 |                                         0
4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 4
8 |                                         0
```

To Create Distribution Plots of Message Sizes By Communicator, Rank, and Send/Receive

- Issue the following DTrace command, substituting the process ID of the process you want to monitor for `pid`:

```bash
dtrace -p pid -n 'mpiperuse$target:::comm-req-xfer-end { @[args[3]->mcs_comm, args[3]->mcs_peer, args[3]->mcs_op] = quantize(args[3]->mcs_count); }'
```

DTrace returns a result similar to the following. In this example, the process ID is 24937.

```bash
% dtrace -p 24937 -n 'mpiperuse$target:::comm-req-xfer-end { @[args[3]->mcs_comm, args[3]->mcs_peer, args[3]->mcs_op] = quantize(args[3]->mcs_count); }'
dtrace: description 'mpiperuse$target:::comm-req-xfer-end ' matched 19 probes
^C
134614864        1  recv
```

```
value  ------------- Distribution ------------- count
2 |                                         0
4 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 9
8 |                                         0
```
<table>
<thead>
<tr>
<th>Value</th>
<th>Distribution</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
| 4     | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

134614864  1 send
value -------- Distribution -------- count
2           |
4 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Troubleshooting

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

MPI Messages

Standard Error Classes

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_SUCCESS</td>
<td>0</td>
<td>Successful return code.</td>
</tr>
<tr>
<td>MPI_ERR_BUFFER</td>
<td>1</td>
<td>Invalid buffer pointer.</td>
</tr>
<tr>
<td>MPI_ERR_COUNT</td>
<td>2</td>
<td>Invalid count argument.</td>
</tr>
<tr>
<td>MPI_ERR_TYPE</td>
<td>3</td>
<td>Invalid datatype argument.</td>
</tr>
<tr>
<td>MPI_ERR_TAG</td>
<td>4</td>
<td>Invalid tag argument.</td>
</tr>
<tr>
<td>MPI_ERR_COMM</td>
<td>5</td>
<td>Invalid communicator.</td>
</tr>
</tbody>
</table>
### TABLE A-1  Open MPI Standard Error Classes  (Continued)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_ERR_RANK</td>
<td>6</td>
<td>Invalid rank.</td>
</tr>
<tr>
<td>MPI_ERR_ROOT</td>
<td>7</td>
<td>Invalid root.</td>
</tr>
<tr>
<td>MPI_ERR_GROUP</td>
<td>8</td>
<td>Null group passed to function.</td>
</tr>
<tr>
<td>MPI_ERR_OP</td>
<td>9</td>
<td>Invalid operation.</td>
</tr>
<tr>
<td>MPI_ERR_TOPOLOGY</td>
<td>10</td>
<td>Invalid topology.</td>
</tr>
<tr>
<td>MPI_ERR_DIMS</td>
<td>11</td>
<td>Illegal dimension argument.</td>
</tr>
<tr>
<td>MPI_ERR_ARG</td>
<td>12</td>
<td>Invalid argument.</td>
</tr>
<tr>
<td>MPI_ERR_UNKNOWN</td>
<td>13</td>
<td>Unknown error.</td>
</tr>
<tr>
<td>MPI_ERR_TRUNCATE</td>
<td>14</td>
<td>Message truncated on receive.</td>
</tr>
<tr>
<td>MPI_ERR_OTHER</td>
<td>15</td>
<td>Other error; use Error_string.</td>
</tr>
<tr>
<td>MPI_ERR_INTERN</td>
<td>16</td>
<td>Internal error code.</td>
</tr>
<tr>
<td>MPI_ERR_IN_STATUS</td>
<td>17</td>
<td>Look in status for error value.</td>
</tr>
<tr>
<td>MPI_ERR_PENDING</td>
<td>18</td>
<td>Pending request.</td>
</tr>
<tr>
<td>MPI_ERR_REQUEST</td>
<td>19</td>
<td>Illegal MPI_Request() handle.</td>
</tr>
<tr>
<td>MPI_ERR_KEYVAL</td>
<td>36</td>
<td>Illegal key value.</td>
</tr>
<tr>
<td>MPI_ERR_INFO</td>
<td>37</td>
<td>Invalid info object.</td>
</tr>
<tr>
<td>MPI_ERR_INFO_KEY</td>
<td>38</td>
<td>Illegal info key.</td>
</tr>
<tr>
<td>MPI_ERR_INFO_NOKEY</td>
<td>39</td>
<td>No such key.</td>
</tr>
<tr>
<td>MPI_ERR_INFO_VALUE</td>
<td>40</td>
<td>Illegal info value.</td>
</tr>
<tr>
<td>MPI_ERR_TIMEOUT</td>
<td>41</td>
<td>Timed out.</td>
</tr>
<tr>
<td>MPI_ERR_SYSRESOURCES</td>
<td>42</td>
<td>Out of resources.</td>
</tr>
<tr>
<td>MPI_ERR_SPAWN</td>
<td>45</td>
<td>Error spawning.</td>
</tr>
<tr>
<td>MPI_ERR_WIN</td>
<td>46</td>
<td>Invalid window.</td>
</tr>
<tr>
<td>MPI_ERR_BASE</td>
<td>47</td>
<td>Invalid base.</td>
</tr>
<tr>
<td>MPI_ERR_SIZE</td>
<td>48</td>
<td>Invalid size.</td>
</tr>
<tr>
<td>MPI_ERR_DISP</td>
<td>49</td>
<td>Invalid displacement.</td>
</tr>
<tr>
<td>MPI_ERR_LOCKTYPE</td>
<td>50</td>
<td>Invalid locktype.</td>
</tr>
</tbody>
</table>
MPI I/O Error Handling

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

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

<table>
<thead>
<tr>
<th>Error Class</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_ERR_ASSERT</td>
<td>51</td>
<td>Invalid assert.</td>
</tr>
<tr>
<td>MPI_ERR_RMA_CONFLICT</td>
<td>52</td>
<td>Conflicting accesses to window.</td>
</tr>
<tr>
<td>MPI_ERR_RMA_SYNC</td>
<td>53</td>
<td>Erroneous RMA synchronization.</td>
</tr>
<tr>
<td>MPI_ERR_NO_MEM</td>
<td>54</td>
<td>Memory exhausted.</td>
</tr>
<tr>
<td>MPI_ERR_LASTCODE</td>
<td>55</td>
<td>Last error code.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_ERR_FILE</td>
<td>20</td>
<td>Bad file handle.</td>
</tr>
<tr>
<td>MPI_ERR_NOT_SAME</td>
<td>21</td>
<td>Collective argument not identical on all processes.</td>
</tr>
<tr>
<td>MPI_ERR_AMODE</td>
<td>22</td>
<td>Unsupported amode passed to open.</td>
</tr>
<tr>
<td>MPI_ERR_UNSUPPORTED_DATAREP</td>
<td>23</td>
<td>Unsupported datarep passed to MPI_File_set_view().</td>
</tr>
<tr>
<td>MPI_ERR_UNSUPPORTED_OPERATION</td>
<td>24</td>
<td>Unsupported operation, such as seeking on a file that supports only sequential access.</td>
</tr>
<tr>
<td>MPI_ERR_NO_SUCH_FILE</td>
<td>25</td>
<td>File (or directory) does not exist.</td>
</tr>
<tr>
<td>MPI_ERR_FILE_EXISTS</td>
<td>26</td>
<td>File exists.</td>
</tr>
</tbody>
</table>
### TABLE A-2  Open MPI I/O Error Classes (Continued)

<table>
<thead>
<tr>
<th>Error Class</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_ERR_BAD_FILE</td>
<td>27</td>
<td>Invalid file name (for example, path name too long).</td>
</tr>
<tr>
<td>MPI_ERR_ACCESS</td>
<td>28</td>
<td>Permission denied.</td>
</tr>
<tr>
<td>MPI_ERR_NO_SPACE</td>
<td>29</td>
<td>Not enough space.</td>
</tr>
<tr>
<td>MPI_ERR_QUOTA</td>
<td>30</td>
<td>Quota exceeded.</td>
</tr>
<tr>
<td>MPI_ERR_READ_ONLY</td>
<td>31</td>
<td>Read-only file system.</td>
</tr>
<tr>
<td>MPI_ERR_FILE_IN_USE</td>
<td>32</td>
<td>File operation could not be completed, as the file is currently open by some process.</td>
</tr>
<tr>
<td>MPI_ERR_DUP_DATAREP</td>
<td>33</td>
<td>Conversion functions could not be registered because a data representation identifier that was already defined was passed to MPI_REGISTER_DATAREP.</td>
</tr>
<tr>
<td>MPI_ERR_CONVERSION</td>
<td>34</td>
<td>An error occurred in a user-supplied data-conversion function.</td>
</tr>
<tr>
<td>MPI_ERR_IO</td>
<td>35</td>
<td>I/O error.</td>
</tr>
<tr>
<td>MPI_ERR_INFO</td>
<td>37</td>
<td>Invalid info object.</td>
</tr>
<tr>
<td>MPI_ERR_INFO_KEY</td>
<td>38</td>
<td>Illegal info key.</td>
</tr>
<tr>
<td>MPI_ERR_INFO_NOKEY</td>
<td>39</td>
<td>No such key.</td>
</tr>
<tr>
<td>MPI_ERR_INFO_VALUE</td>
<td>40</td>
<td>Illegal info value.</td>
</tr>
<tr>
<td>MPI_ERR_LASTCODE</td>
<td>55</td>
<td>Last error code.</td>
</tr>
</tbody>
</table>
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:

```
mynode@ mpirun -np 61 hello_c
mpirun noticed that job rank 0 with PID 0 on node burl-ct-v440-1
exited on signal 15 (Terminated).
59 additional processes aborted (not shown)
```

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

The Solaris OS default file descriptor limit is 256. When you start an MPI job, a program called an orted (for ORTE daemon) spawns the user processes. For each user process spawned, the orted takes up four file descriptors. In addition, the job takes 12 additional file descriptors regardless of the number of processes spawned.

To calculate the number of file descriptors needed to run a certain job, use the following formula:

\[
\text{file descriptors} = 12 + 4 \times np
\]

where \(np\) is the number of processes launched.

If the number of file descriptors needed is greater than 256, you must increase the number of available descriptors to a value equal to or greater than the number you calculated. Otherwise, the processes fail and the error message is displayed.

Increasing the Number of Available File Descriptors

▼ To View the Hard Limit from the C Shell

1. Log in to a C shell as superuser.
2. Determine the current hard limit value for your Solaris implementation. Type the following command:

```
# limit -h descriptors
```

▼ To View the Hard Limit from the Bourne Shell

1. Log in to a Bourne shell as superuser.
2. Use the `ulimit` function. Type the following command:

```
# ulimit -Hn
```

Each function returns the file descriptor hard limit that was in effect. The new value you set for the number of available file descriptors must be less than or equal to this number. The usual default value for the hard limit in the Solaris OS is 64000 (64K).

▼ To Increase the Number of File Descriptors

**Note** – You must perform this procedure on each of the nodes on which you plan to run.

1. Open the `/etc/system` file in a text editor.
2. Add the following line to the file:

```
set rlim_fd_cur=value
```

where `value` is the new maximum number of file descriptors. For example, the following line added to the `/etc/system` file increases the maximum number of file descriptors to 1024:

```
set rlim_fd_cur=1024
```

3. Save the file and exit the text editor.
4. Reboot the system.
Setting File Descriptor Limits When Using Sun Grid Engine

If you are using Sun Grid Engine to launch your jobs on very large multi-processor nodes, you might see an error message about exceeding your file descriptor limit, and your jobs might fail. This can happen because Sun Grid Engine cannot set the file descriptor limit in its queue.

There are three ways in which you can adjust the number of available file descriptors when you use Sun Grid Engine:

1. Set the file descriptor limit in your login shell (.cshrc, .tcshrc, .bashrc, and so on).

2. Modify the /etc/shell file for each of the nodes on your cluster as described in the previous section, “To Increase the Number of File Descriptors” on page 108. Remember that you must reboot all of the nodes in the cluster once you have finished modifying the files.

3. On a Sun Grid Engine execution host, modify the $SGE_ROOT/default/common/sgeexecd startup script to increase the file descriptor limit to the same value as the hard limit (as described in the previous section). You must restart the sgeexecd daemon on the host. Since this script is shared among the Sun Grid Engine execution hosts in the cluster using NFS, you may make the change on one host, and it will be propagated to the other Sun Grid Engine hosts in the cluster.
List of Available MCA Parameters

This section lists the MCA parameters available in the Solaris OS implementation of Open MPI 1.3. The parameters are grouped under the framework to which they correspond.

To view all MCA parameters available in your implementation of Open MPI, type the following command:

```
% omni_info --param all all
```

Note that the list of available MCA parameters is different for each operating system and implementation of Open MPI. For more information, see the Open MPI Web site at:

http://www.open-mpi.org

Note – The list of MCA parameters is constantly changing due to ongoing development. See the Open MPI Web site for more information.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mca</td>
<td>mca_param_files</td>
<td>Path for MCA configuration files containing default parameter values</td>
<td>$HOME/.openmpi/mca-params.conf:/opt/SUNWhpc/HPC8.1/etc/openmpi-mca-params.conf</td>
</tr>
<tr>
<td></td>
<td>mca_component_path</td>
<td>Path where to look for Open MPI and ORTE components</td>
<td>/opt/SUNWhpc/HPC8.1/lib/openmpi:$HOME/openmpi/components</td>
</tr>
<tr>
<td></td>
<td>mcaVerbose</td>
<td>Top-level verbosity parameter</td>
<td>none</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mca_component_show_load_errors</td>
<td>Whether or not to show errors for components that failed to load</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>mca_component_disable_dlopen</td>
<td>Whether to attempt to disable opening dynamic components or not</td>
<td>0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi_param_check</td>
<td>Whether you want MPI API parameters checked at runtime or not. Possible values are 0 (no checking) and 1 (perform checking at runtime)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>mpi_yield_when_idle</td>
<td>Yield the processor when waiting for MPI communication (for MPI processes, will default to 1 when oversubscribing nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpi_event_tick_rate</td>
<td>How often to progress TCP communications (-1 = leave default value. 0 = never, otherwise specified in microseconds)</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>mpi_show_handle_leaks</td>
<td>Whether or not MPI_FINALIZE shows all MPI handles that were not freed</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpi_no_free_handles</td>
<td>Whether to actually free MPI objects when their handles are freed</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpi_show_mca_params</td>
<td>Whether to show all MCA parameter value during MPI_INIT or not (good for reproducibility of MPI jobs)</td>
<td>0 (don’t show)</td>
</tr>
<tr>
<td></td>
<td>mpi_show_mca_params_file</td>
<td>If mpi_show_mca_params is true, setting this string to a valid file name tells Open MPI to dump all the MCA parameter values into a file suitable for reading using the mca_param_files parameter (good for reproducibility of MPI jobs)</td>
<td>none</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>opal_paffinity_alone</td>
<td>If nonzero, assume that this job is the only (set of) process(es) running on each node and bind processes to processors, starting with processor ID 0.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpi_keep_peer_hostnames</td>
<td>If nonzero, save the string hostnames of all MPI peer processes (mostly for error / debugging output messages). This can add quite a bit of memory usage to each MPI process.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>mpi_abort_delay</td>
<td>If nonzero, print out an identifying message when MPI_ABORT is invoked (hostname, PID of the process that called MPI_ABORT) and delay for that many seconds before exiting (a negative delay value means to never abort). This allows attaching of a debugger before quitting the job.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpi_abort_print_stack</td>
<td>If nonzero, print out a stack trace when MPI_ABORT is invoked</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpi_preconnect_all</td>
<td>Whether to force MPI processes to create connections/ warmup with all peers during MPI_INIT (vs. making connections lazily -- upon the first MPI traffic between each process/peer pair)</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: this is an information parameter. Its value is returned by Open MPI. This value cannot be adjusted manually.
### TABLE B-1 Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi</td>
<td>mpi_leave_pinned</td>
<td>Whether to use the &quot;leave pinned&quot; protocol or not. Enabling this setting can help bandwidth performance when repeatedly sending and receiving large messages with the same buffers over RDMA-based networks. Use this parameter with care; setting it to too high a value can lead to resource exhaustion.</td>
<td>0</td>
</tr>
<tr>
<td>mpi</td>
<td>mpi_leave_pinned_pipeline</td>
<td>Whether to use the &quot;leave pinned pipeline&quot; protocol or not.</td>
<td>0</td>
</tr>
<tr>
<td>orte</td>
<td>orte_base_user_debugger</td>
<td>Sequence of user-level debuggers to search for in orterun</td>
<td>totalview @mpirun@ -a @mpirun_args@ : fxp @mpirun@ -a @mpirun_args@</td>
</tr>
<tr>
<td>orte</td>
<td>orte_debug</td>
<td>Whether or not to enable debugging output for all ORTE components (0 or 1)</td>
<td>0</td>
</tr>
<tr>
<td>orte</td>
<td>orte_no_daemonize</td>
<td>Whether to properly daemonize the ORTE daemons or not</td>
<td>0</td>
</tr>
<tr>
<td>orte</td>
<td>orte_abort_timeout</td>
<td>Time to wait [in seconds] before giving up on aborting an ORTE operation</td>
<td>10</td>
</tr>
<tr>
<td>orte</td>
<td>orte_timing</td>
<td>Request that critical timing loops be measured</td>
<td>0</td>
</tr>
<tr>
<td>opal</td>
<td>opal_signal</td>
<td>If a signal is received, display the stack trace frame</td>
<td>6,10,8,11</td>
</tr>
<tr>
<td>backtrace</td>
<td>backtrace</td>
<td>Default selection set of components for the backtrace framework (&lt;none&gt; means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>backtrace_base_verbose</td>
<td>Verbosity level for the backtrace framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td>memory</td>
<td>memory_printstack_priority</td>
<td>Verbosity level for the memory framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td>memory</td>
<td>memory_base_verbose</td>
<td>Verbosity level for the memory framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td>paffinity</td>
<td>paffinity</td>
<td>Default selection set of components for the paffinity framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td>maffinity</td>
<td>maffinity</td>
<td>Default selection set of components for the maffinity framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>maffinity_first_use_priority</td>
<td>Priority of the first_use maffinity component</td>
<td>10</td>
</tr>
<tr>
<td>timer</td>
<td>timer</td>
<td>Default selection set of components for the timer framework (&lt;none&gt; means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>timer_base_verbose</td>
<td>Verbosity level for the timer framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>timer_solaris_priority</td>
<td>Priority of the Solaris OS maffinity component</td>
<td>0</td>
</tr>
<tr>
<td>allocator</td>
<td>allocator</td>
<td>Default selection set of components for the allocator framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
</tbody>
</table>
## Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allocator_base_verbose</td>
<td>Verbosity level for the allocator framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allocator_basic_priority</td>
<td>Priority of the basic allocator component</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allocator_bucket_num_buckets</td>
<td>Number of buckets in the allocator bucket component</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>allocator_bucket_priority</td>
<td>Priority of the allocator bucket component</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>coll</td>
<td>Default selection set of components for the coll framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>coll_base_verbose</td>
<td>Verbosity level for the coll framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>coll_basic_priority</td>
<td>Priority of the basic coll component</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>coll_basic_crossover</td>
<td>Minimum number of processes in a communicator before using the logarithmic algorithms</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>coll_self_priority</td>
<td>Priority of the self coll component</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>coll_sm_priority</td>
<td>Priority of the sm coll component</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>coll_sm_control_size</td>
<td>Length of the control data -- should usually be either the length of a cache line on most multiprocessors, or the size of a page on machines that support direct memory affinity page placement (in bytes)</td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td>coll_sm_bootstrap_filename</td>
<td>File name (in the Open MPI session directory) of the coll_sm component bootstrap rendezvous mmap file</td>
<td>shared_mem_sm_bootstrap</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>coll_sm_bootstrap_num_segments</td>
<td>Number of segments in the bootstrap file</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>coll_sm_fragment_size</td>
<td>Fragment size (in bytes) used for passing data through shared memory (will be rounded up to the nearest control_size size)</td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td>coll_sm_mpool</td>
<td>Name of the mpool component to use</td>
<td>sm</td>
</tr>
<tr>
<td></td>
<td>coll_sm_comm_in_use_flags</td>
<td>Number of “in use” flags, used to mark a message passing area segment as currently being used or not (must be &gt;= 2 and &lt;= comm_num_segments)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>coll_sm_comm_num_segments</td>
<td>Number of segments in each communicator’s shared memory message passing area (must be &gt;= 2, and must be a multiple of comm_in_use_flags)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>coll_sm_tree_degree</td>
<td>Degree of the tree for tree-based operations (must be &gt; 1 and &lt;= min(control_size, 255))</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>coll_sm_shared_mem_used_bootstrap</td>
<td>Amount of shared memory used in the shared memory bootstrap area (in bytes)</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: this is an information parameter. Its value is returned by Open MPI. This value cannot be adjusted manually.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>coll_sm_info_num_procs</td>
<td>Number of processes to use for the calculation of the shared_mem_size MCA information parameter (must be =&gt; 2)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>coll_sm_shared_mem_used_data</td>
<td>Amount of shared memory used in the shared memory data area for info_num_procs processes (in bytes)</td>
<td>548864</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: this is an information parameter. Its value is returned by Open MPI. This value cannot be adjusted manually.</td>
<td></td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td><strong>coll_tuned_priority</strong></td>
<td>Priority of the tuned coll component</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td><strong>coll_tuned_pre_allocate_memory</strong></td>
<td>Size of communicator where you want to stop pre-allocating memory for the fixed internal buffer. The buffer is used for message requests attached to the communicator data segment. For example, if you have a 100,000-node cluster, you might not want to pre-allocate 200,000 request handle slots per communicator instance.</td>
<td>32768</td>
</tr>
<tr>
<td></td>
<td><strong>coll_tuned_use_dynamic_rules</strong></td>
<td>Switch used to decide whether to use static (compiled/if statements) or dynamic (built at runtime) decision function rules</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td><strong>coll_tuned_init_tree_fanout</strong></td>
<td>Initial fanout used in the tree topologies for each communicator. This is only an initial estimate value. If a tuned collective needs a different fanout for an operation, it builds it dynamically. This parameter is only for the first guess and might save a little time.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td><strong>coll_tuned_init_chain_fanout</strong></td>
<td>Initial fanout used in the chain (fanout followed by pipeline) topologies for each communicator. This is only an initial estimate value. If a tuned collective needs a different fanout for an operation, it builds it dynamically. This parameter is only for the first guess and might save a little time.</td>
<td>4</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>coll_tuned_use_dynamic_rules</td>
<td>Switch used to decide whether to use static (compiled/if statements) or dynamic (built at runtime) decision function rules (0 = use static rules)</td>
<td>0</td>
</tr>
<tr>
<td>io</td>
<td>io</td>
<td>Default selection set of components for the io framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>io_base_freelist_initial_size</td>
<td>Initial MPI-2 IO request freelist size</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>io_base_freelist_max_size</td>
<td>Max size of the MPI-2 IO request freelist</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>io_base_freelist_increment</td>
<td>Increment size of the MPI-2 IO request freelist</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>io_base_verbose</td>
<td>Verbosity level for the io framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>io_romio_priority</td>
<td>Priority of the io romio component</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>io_romio_delete_priority</td>
<td>Delete priority of the io romio component</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>io_romio_enable_parallel_optimizations</td>
<td>Enable set of Open MPI-added options to improve collective file I/O performance</td>
<td>0</td>
</tr>
<tr>
<td>mpool</td>
<td>mpool</td>
<td>Default selection set of components for the mpool framework (&lt;none&gt; means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>mpool_base_verbose</td>
<td>Verbosity level for the mpool framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpool_sm_allocator</td>
<td>Type of shared memory allocator used</td>
<td>bucket</td>
</tr>
<tr>
<td></td>
<td>mpool_sm_max_size</td>
<td>Maximum size of the mpool shared memory file</td>
<td>536870912</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>mpool</td>
<td>mpool_sm_min_size</td>
<td>Minimum size of the mpool shared memory file</td>
<td>134217728</td>
</tr>
<tr>
<td></td>
<td>mpool_sm_per_peer_size</td>
<td>Size (in bytes) to allocate per local peer in the mpool shared memory file, bounded by min_size and max_size</td>
<td>33554432</td>
</tr>
<tr>
<td></td>
<td>mpool_sm_priority</td>
<td>Priority of the shared memory allocator for mpool</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpool_udapl_priority</td>
<td>Priority of the uDAPL BTL</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpool_base_use_mem_hooks</td>
<td>Use memory hooks for deregistering freed memory</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mpool_use_mem_hooks</td>
<td>Deprecated; use mpool_base_use_mem_hook</td>
<td>0</td>
</tr>
<tr>
<td>pml</td>
<td>pml</td>
<td>Default selection set of components for the pml framework (none means &quot;use all components that can be found&quot;)</td>
<td>ob1</td>
</tr>
<tr>
<td></td>
<td>pml_base_verbose</td>
<td>Verbosity level for the pml framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>pml_cm_free_list_num</td>
<td>Initial size of request free lists</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>pml_cm_free_list_max</td>
<td>Maximum size of request free lists</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>pml_cm_free_list_inc</td>
<td>Number of elements to add when growing request free lists</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>pml_cm_priority</td>
<td>CM PML selection priority</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>pml_dr_free_list_num</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>pml_dr_free_list_max</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>pml_dr_free_list_inc</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>pml_dr_priority</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>pml_dr_eager_limit</td>
<td></td>
<td>131072</td>
</tr>
<tr>
<td></td>
<td>pml_dr_send_pipeline_depth</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pml</td>
<td>pml_dr_wdog_timer_sec</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_wdog_timer_usec</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_wdog_timer_multiplier</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_wdog_retry_max</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_ack_timer_sec</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_ack_timer_usec</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_ack_timer_multiplier</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_ack_retry_max</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_dr_enable_csum</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_free_list_num</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_free_list_max</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_free_list_inc</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_priority</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_eager_limit</td>
<td>131072</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_send_pipeline_depth</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pml_ob1_recv_pipeline_depth</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>bml</td>
<td>bml</td>
<td>Default selection set of components for the bml framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>bml_base_verbose</td>
<td>Verbosity level for the bml framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>bml_r2_show_unreach_errors</td>
<td>Show error message when procs are unreachable</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>bml_r2_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>rcache</td>
<td>rcache</td>
<td>Default selection set of components for the rcache framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>rcache_base_verbose</td>
<td>Verbosity level for the rcache framework (0 = no verbosity)</td>
<td>0</td>
</tr>
</tbody>
</table>
### Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rcache_rb_priority</td>
<td>The maximum size IN ENTRIES of the MRU (most recently used) rcache list</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rcache_vma_mru_len</td>
<td>The maximum size IN ENTRIES of the MRU (most recently used) rcache list</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>rcache_vma_mru_size</td>
<td>The maximum size IN BYTES of the MRU (most recently used) rcache list</td>
<td>1073741824</td>
</tr>
<tr>
<td></td>
<td>rcache_vma_priority</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>btl</td>
<td>btl</td>
<td>Default selection set of components for the btl framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>btl_base_debug</td>
<td>If btl_base_debug is 1 standard debug is output, if &gt; 1 verbose debug is output</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>btl_base_verbose</td>
<td>Verbosity level for the btl framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>btl_self_free_list_num</td>
<td>Number of fragments by default</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>btl_self_free_list_max</td>
<td>Maximum number of fragments</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>btl_self_free_list_inc</td>
<td>Increment by this number of fragments</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>btl_self_eager_limit</td>
<td>Eager size fragment (before the rendezvous protocol)</td>
<td>131072</td>
</tr>
<tr>
<td></td>
<td>btl_self_min_send_size</td>
<td>Minimum fragment size after the rendezvous</td>
<td>262144</td>
</tr>
<tr>
<td></td>
<td>btl_self_max_send_size</td>
<td>Maximum fragment size after the rendezvous</td>
<td>262144</td>
</tr>
<tr>
<td></td>
<td>btl_self_min_rdma_size</td>
<td>Minimum fragment size for the RDMA transfer</td>
<td>2147483647</td>
</tr>
<tr>
<td></td>
<td>btl_self_max_rdma_size</td>
<td>Maximum fragment size for the RDMA transfer</td>
<td>2147483647</td>
</tr>
<tr>
<td></td>
<td>btl_self_exclusivity</td>
<td>Device exclusivity</td>
<td>65536</td>
</tr>
<tr>
<td></td>
<td>btl_self_flags</td>
<td>Active behavior flags</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>btl_self_priority</td>
<td>Device priority</td>
<td>0</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>btl_sm_free_list_num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_free_list_max</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_free_list_inc</td>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_exclusivity</td>
<td>65535</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_latency</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_max_procs</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_extra_procs</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_mpool</td>
<td>sm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_eager_limit</td>
<td>4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_max_frag_size</td>
<td>32768</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_size_of_cb_queue</td>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_cb_lazy_free_freq</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_sm_priority</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_if_include</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_if_exclude</td>
<td>lo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_free_list_num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_free_list_max</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_free_list_inc</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_sndbuf</td>
<td>131072</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_rcvbuf</td>
<td>131072</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_endpoint_cache</td>
<td>30720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_exclusivity</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_eager_limit</td>
<td>65536</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_min_send_size</td>
<td>65536</td>
<td></td>
<td></td>
</tr>
<tr>
<td>btl_tcp_max_send_size</td>
<td>131072</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>btl_tcp_min_rdma_size</code></td>
<td>The user message size at which point the MPI library will begin registering portions of the user’s send and receive buffers, and begin using RDMA operations to write directly from user buffer to user buffer. The write operations occur in portions no greater than the value of <code>btl_tcp_max_rdma_size</code>.</td>
<td>131072</td>
</tr>
<tr>
<td></td>
<td><code>btl_tcp_max_rdma_size</code></td>
<td>The maximum message size that the MPI library will use to perform the actual RDMA operation.</td>
<td>2147483647</td>
</tr>
<tr>
<td></td>
<td><code>btl_tcp_flags</code></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td><code>btl_tcp_priority</code></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_free_list_num</code></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_free_list_max</code></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_free_list_inc</code></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_mpool</code></td>
<td></td>
<td><code>udapl</code></td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_max_modules</code></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_evd_qlen</code></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_max_request_dtos</code></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_max_recv_dtos</code></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_num_recv</code></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_num_sends</code></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_sr_win</code></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_timeout</code></td>
<td></td>
<td>10000000</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_eager_rdma_num</code></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_max_eager_rdma_peers</code></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_eager_rdma_win</code></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_timeout</code></td>
<td></td>
<td>10000000</td>
</tr>
<tr>
<td></td>
<td><code>btl_udapl_eager_rdma_guarantee</code></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
## Appendix B  List of Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>btl</td>
<td>btl_udapl_conn_priv_data</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_exclusivity</td>
<td>1014</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_eager_limit</td>
<td>32768</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_min_send_size</td>
<td>16384</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_max_send_size</td>
<td>65536</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_min_rdma_size</td>
<td>The user message size at which point the MPI library will begin registering portions of the user’s send and receive buffers, and begin using RDMA operations to write directly from user buffer to user buffer. The write operations occur in portions no greater than the value of btl_udapl_max_rdma_size.</td>
<td>524288</td>
</tr>
<tr>
<td></td>
<td>btl_udapl_max_rdma_size</td>
<td>The maximum message size that the MPI library will use to perform the actual RDMA operation.</td>
<td>131072</td>
</tr>
<tr>
<td></td>
<td>btl_udapl_flags</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_bandwidth</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_udapl_priority</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_base_include</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_base_exclude</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>btl_base_warn_component_unused</td>
<td>This parameter is used to turn on warning messages when certain NICs are not used.</td>
<td>0</td>
</tr>
<tr>
<td>mtl</td>
<td>mtl</td>
<td>Default selection set of components for the mtl framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>mtl_base_verbose</td>
<td>Verbosity level for the mtl framework (0 = no verbosity)</td>
<td>0</td>
</tr>
</tbody>
</table>
### Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>topo</td>
<td>topo</td>
<td>Default selection set of components for the topo framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>topo_base_verbose</td>
<td>Verbosity level for the topo framework (0 = no verbosity).</td>
<td>0</td>
</tr>
<tr>
<td>osc</td>
<td>osc</td>
<td>Default selection set of components for the osc framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>osc_base_verbose</td>
<td>Verbosity level for the osc framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>osc_pt2pt_no_locks</td>
<td>Enable optimizations available only if MPI_LOCK is not used.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>osc_pt2pt_eager_limit</td>
<td>Max size of eagerly sent data</td>
<td>16384</td>
</tr>
<tr>
<td></td>
<td>osc_pt2pt_priority</td>
<td>Max size of eagerly sent data</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>osc_rdma_fence_sync_method</td>
<td>Specifies how to synchronize fence: reduce_scatter, allreduce, alltoall</td>
<td>reduce_scatter</td>
</tr>
<tr>
<td></td>
<td>osc_rdma_eager_send</td>
<td>Attempt to start data movement during communication call, instead of at synchronization time. Info key of same name overrides this value, if info key given.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>osc_rdma_no_locks</td>
<td>Enable optimizations available only if MPI_LOCK is not used.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>osc_rdma_priority</td>
<td>Enable optimizations available only if MPI_LOCK is not used.</td>
<td>0</td>
</tr>
</tbody>
</table>
# TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>errmgr</td>
<td>errmgr</td>
<td>Default selection set of components for the errmgr framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>errmgr_hnp_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>errmgr_hnp_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>errmgr_orted_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>errmgr_orted_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>errmgr_proxy_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>errmgr_proxy_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>gpr</td>
<td>gpr</td>
<td>Default selection set of components for the gpr framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>gpr_base_maxsize</td>
<td></td>
<td>2147483647</td>
</tr>
<tr>
<td></td>
<td>gpr_base_blocksize</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>gpr_null_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>gpr_proxy_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>gpr_proxy_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>gpr_replica_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>gpr_replica_isolate</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>gpr_replica_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>iof</td>
<td>iof</td>
<td>Default selection set of components for the iof framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>iof_base_window_size</td>
<td></td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td>iof_base_service</td>
<td></td>
<td>0.0.0</td>
</tr>
<tr>
<td></td>
<td>iof_proxy_debug</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>iof_proxy_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>iof_svc_debug</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>iof_svc_priority</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
## Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns</td>
<td>ns</td>
<td>Default selection set of components for the ns framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>ns_proxy_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ns_proxy_maxsize</td>
<td></td>
<td>2147483647</td>
</tr>
<tr>
<td></td>
<td>ns_proxy_blocksize</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>ns_proxy_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ns_replica_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ns_replica_isolate</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ns_replica_maxsize</td>
<td></td>
<td>2147483647</td>
</tr>
<tr>
<td></td>
<td>ns_replica_blocksize</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>ns_replica_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>oob</td>
<td>oob</td>
<td>Default selection set of components for the oob framework (none means &quot;use all components that can be found&quot;).</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>oob_base_verbose</td>
<td>Verbosity level for the oob framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_peer_limit</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_peer_retries</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_debug</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_include</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_exclude</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_sndbuf</td>
<td></td>
<td>131072</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_rcvbuf</td>
<td></td>
<td>131072</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_connect_timeout</td>
<td>connect() timeout in seconds, before trying next interface</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>oob_tcp_connect_sleep</td>
<td>Enable (1) / Disable (0) random sleep for connection wireup</td>
<td>1</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ras</td>
<td>ras</td>
<td>Default selection set of components for the ras framework (none means “use all components that can be found”)</td>
<td>none</td>
</tr>
<tr>
<td>ras</td>
<td>dash_host_priority</td>
<td>Selection priority for the dash_host RAS component</td>
<td>5</td>
</tr>
<tr>
<td>ras</td>
<td>gridengine_debug</td>
<td>Enable debugging output for the gridengine RAS component</td>
<td>0</td>
</tr>
<tr>
<td>ras</td>
<td>gridengine_priority</td>
<td>Priority of the gridengine ras component</td>
<td>100</td>
</tr>
<tr>
<td>ras</td>
<td>gridengine_verbose</td>
<td>Enable verbose output for the gridengine ras component</td>
<td>0</td>
</tr>
<tr>
<td>ras</td>
<td>gridengine_show_jobid</td>
<td>Show the JOB_ID of the Grid Engine job</td>
<td>0</td>
</tr>
<tr>
<td>ras</td>
<td>hostfile_priority</td>
<td>Selection priority for the hostfile RAS component</td>
<td>10</td>
</tr>
<tr>
<td>ras</td>
<td>localhost_priority</td>
<td>Selection priority for the localhost RAS component</td>
<td>0</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ras</td>
<td>ras_tm_priority</td>
<td>Priority of the tm ras component</td>
<td>100</td>
</tr>
<tr>
<td>rds</td>
<td>rds</td>
<td>Default selection set of components for the rds framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>rds_hostfile_debug</td>
<td>Toggle debug output for hostfile RDS component. 1=enable; 0=disable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rds_hostfile_path</td>
<td>Path to the ORTE Host filename.</td>
<td>/opt/SUNW/hpc/HPC8.1/etc/openmpi-default-hostfile</td>
</tr>
<tr>
<td></td>
<td>rds_hostfile_priority</td>
<td>Path to the ORTE Host filename.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rds_proxy_priority</td>
<td>Path to the ORTE Host filename.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rds_resfile_debug</td>
<td>Toggle debug output for resfile RDS component. 1=enable; 0=disable.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rds_resfile_priority</td>
<td>ORTE Resource filename</td>
<td>none</td>
</tr>
<tr>
<td>rmaps</td>
<td>rmaps</td>
<td>Default selection set of components for the rmaps framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>rmaps_base_verbose</td>
<td>Verbosity level for the rmaps framework</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rmaps_base_schedule_policy</td>
<td>Scheduling policy for RMAPS. [slot</td>
<td>node]</td>
</tr>
<tr>
<td></td>
<td>rmaps_base_pernode</td>
<td>Launch one ppn as directed</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rmaps_base_n_pernode</td>
<td>Launch n procs/node</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>rmaps_base_schedule_local</td>
<td>If nonzero, allow scheduling MPI applications on the same node as mpirun (default). If zero, do not schedule any MPI applications on the same node as mpirun.</td>
<td>1</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>rmaps_base_no_oversubscribe</td>
<td>If nonzero, then do not allow oversubscription of nodes. mpirun will return an error if there are not enough nodes to launch all processes without oversubscribing.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rmaps_round_robin_debug</td>
<td>Toggle debug output for Round Robin RMAPS component</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>rmaps_round_robin_priority</td>
<td>Selection priority for Round Robin RMAPS component</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>rmgr</td>
<td>Default selection set of components for the rmgr framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>rmgr_proxy_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rmgr_urm_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rml</td>
<td>Default selection set of components for the rml framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>rml_base_verbose</td>
<td>Verbosity level for the rml framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>rml_oob_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm</td>
<td>Default selection set of components for the plm framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>plm_base_reuse_daemons</td>
<td>If nonzero, reuse daemons to launch dynamically spawned processes. If zero, do not reuse daemons (default)</td>
<td>0</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>plm_fork_reap</td>
<td>plm_fork_reap_timeout</td>
<td>Specifies whether or not to wait to reap all children before finalizing. 1=wait; 0=do not wait</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm_fork_reap</td>
<td>Wait to reap all children before finalizing. 1=wait; 0=do not wait</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>plm_gridengine_debug</td>
<td>When killing child processes, first send a SIGTERM, then wait at least this timeout (in seconds), then send a SIGKILL</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm_gridengine_verbose</td>
<td>Priority of this component</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>plm_gridengine_priority</td>
<td>Priority of the gridengine plm component</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>plm_gridengine_orted</td>
<td>The command name that the gridengine plm component will invoke for the ORTE daemon</td>
<td>orted</td>
</tr>
<tr>
<td></td>
<td>plm_proxy_priority</td>
<td>The command name that the gridengine plm component will invoke for the ORTE daemon</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_debug</td>
<td>Enable debugging output for the rsh plm component. 1=enable; 0=do not enable</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_num_concurrent</td>
<td>How many plm_rsh_agent instances to invoke concurrently (value must be greater than 0)</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_orted</td>
<td>The command name that the rsh plm component will invoke for the ORTE daemon</td>
<td>orted</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_force_rsh</td>
<td>Force the launcher to always use rsh, even for local daemons</td>
<td>0</td>
</tr>
<tr>
<td>Framework</td>
<td>Parameter Name</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_priority</td>
<td>Priority of the rsh plm component</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_delay</td>
<td>Delay (in seconds) between invocations of the remote agent, but only used when the debug MCA parameter is set to true, or when the top-level MCA debugging is enabled (otherwise this value is ignored)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_reap</td>
<td>If set to 1, wait for all the processes to complete before exiting. Otherwise, quit immediately without waiting for confirmation that all other processes in the job have completed.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_assume_same_shell</td>
<td>If set to 1, assume that the shell on the remote node is the same as the shell on the local node. Otherwise, probe for what the remote shell.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>plm_rsh_agent</td>
<td>The command used to launch executables on remote nodes (typically either &quot;ssh&quot; or &quot;rsh&quot;)</td>
<td>ssh : rsh</td>
</tr>
<tr>
<td></td>
<td>plm_tm_debug</td>
<td>Enable debugging of the TM plm. 1= enable; 0= do not enable</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm_tm_verbose</td>
<td>Enable verbose output of the TM plm. 1= enable; 0= do not enable</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>plm_tm_priority</td>
<td>Default selection priority</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>plm_tm_orted</td>
<td>Command to use to start proxy orted</td>
<td>orted</td>
</tr>
</tbody>
</table>
### TABLE B-1  Available MCA Parameters

<table>
<thead>
<tr>
<th>Framework</th>
<th>Parameter Name</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>plm_tm_want_path_check</td>
<td>Specify whether the launching process should check for the plm_tm_orted executable in the PATH before launching. Any nonzero value enables this check. Note: the TM API does not give an indication of failure.</td>
<td>1</td>
</tr>
<tr>
<td>sds</td>
<td>sds</td>
<td>Default selection set of components for the sds framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
<tr>
<td>sds</td>
<td>sds_base_verbose</td>
<td>Verbosity level for the sds framework (0 = no verbosity)</td>
<td>0</td>
</tr>
<tr>
<td>sds</td>
<td>sds_env_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>sds</td>
<td>sds_pipe_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>sds</td>
<td>sds_seed_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>sds</td>
<td>sds_singleton_priority</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>soh</td>
<td>soh</td>
<td>Default selection set of components for the soh framework (none means &quot;use all components that can be found&quot;)</td>
<td>none</td>
</tr>
</tbody>
</table>
Index

A
applications
  migrating, 15
  recompiling, 15

B
batch systems, 7

C
cluster
  about, 5
command line interface (CRE), 2
compilation, 16
compilers
  mpicc, 16
  mpicc compiler, 15
  mpiCC, mpicxx, mpic++, 16
  mpi77, 16
  mpi90, 16
  wrapper compilers, 16
compiling
  using the wrapper compilers, 16
configurations, supported, 1
CRE, 2

D
D language, 85
default settings
  how to run a program with, 23
documentation
  MPI Reference Manual, xi
  product notes, xi
  Sun documentation on the web, xiii
  DTrace, 85
    advantages over MPI profiling, 89
    attaching to an MPI process, 89
    running with MPI programs, 87
    tracing MPI programs, 89
    tracking resource leaks, 92
    using with MPI, 88
dtrace_proc, 86
dtrace_user, 86
Dynamic tracing
  using with MPI, 87

E
E privilege set
  dtrace privileges, 87
error classes, standard, 103
error classes, Sun MPI I/O, 105
error handling, MPI I/O, 105
errors
  tracing using DTrace, 89
  exceeding the file descriptor limit, 107

F
File descriptor
  exceeding the limit, 107

G
Grid Engine
  open source version, 8
help, how to display, 32

-attach DTrace to an MPI process, 89
-change the working directory, 31
determine which function is returning errors, 89
determine your mprun privileges, 86
display command help, 32
run a program as multiple processes, 23
run a program with default settings, 23
track down a resource leak, 93
use DTrace with an MPI program, 88
use DTrace with Sun MPI, 85

limit -h, 107
linking, 16

mapping MPI processes to nodes, 24
MCA parameters, 8
gridengine parameters, 55
messages, MPI, 103
MPI
attaching DTrace to a process, 89
running a program under DTrace, 88
Sun MPI, 3
tracing programs, 88
tracing programs using DTrace, 89
tracking resource leaks, 91
MPI messages, 103
MPI_COMM_WORLD
inheriting stdin, 30
mpicc compiler, 15
mpif77, 16
mpif90, 16
mpirun
-mca option, 8
mprun
-C, 31
default settings, 23
determining privileges on the cluster, 86
-h, 32
-mp, 23
privileges for use with DTrace, 86
syntax, 20

node
mapping MPI processes to, 24
nodes
about, 5

Open MPI
-and Sun N1 Grid Engine, 7

parallel environment (PE), 8
process
how to run a program as multiple, 23
mapping to nodes, 24

remote nodes
standard output, 31
Resource leaks
determining using DTrace, 93
tracking, 91

scalability, 1
Solaris Dynamic Tracing utility (DTrace), 85
standard error, 31
standard input, 30
standard output, 31
submitting jobs
under Sun N1 Grid Engine, 8
Sun Grid Engine
-and Open MPI, 7
Sun HPC ClusterTools 6
migrating applications, 15
Sun N1 Grid Engine
gridengine MCA parameters, 55

tmcc compiler, 15
troubleshooting, 103

ulimit -Hn, 108
W
wildcards
   using in tracing scripts, 91
working directory, how to change the, 31