

---

# Using Platform **LSF**<sup>®</sup> Parallel

Version 5.0

June 2002

Comments to: [doc@platform.com](mailto:doc@platform.com)

---

**Copyright** © 1994-2002 Platform Computing Corporation  
All rights reserved.

**We'd like to hear from you** You can help us make this manual better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this manual, please address your comments to [doc@platform.com](mailto:doc@platform.com). Your comments should pertain only to Platform LSF documentation. For product support, contact [support@platform.com](mailto:support@platform.com).

Although the information in this document has been carefully reviewed, Platform Computing Corporation ("Platform") does not warrant it to be free of errors or omissions. Platform reserves the right to make corrections, updates, revisions or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY PLATFORM, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM COMPUTING BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

**Trademarks** ® **LSF** is a registered trademark of Platform Computing Corporation in the United States and in other jurisdictions.

™ PLATFORM COMPUTING, and the PLATFORM and LSF logos are trademarks of Platform Computing Corporation in the United States and in other jurisdictions.

UNIX is a registered trademark of The Open Group in the United States and in other jurisdictions.

Other products or services mentioned in this document are identified by the trademarks or service marks of their respective owners.

**Last update** June 5 2002

# Contents

Welcome	5
About This Guide	6
Who should read this book	6
What you should already know	6
Typographical conventions	6
Command notation	6
Learning About Parallel Programming	7
Related publications	7
Learning About Platform LSF	8
World Wide Web and FTP	8
Platform LSF manuals	8
Technical support	8
We'd like to hear from you	8
<b>1 About LSF Parallel</b>	<b>9</b>
What Is LSF Parallel?	10
How Does LSF Parallel Work with Platform LSF?	11
MPI library	11
PAM	11
Platform LSF	12
LSF Parallel Architecture	13
LSF Parallel components	14
<b>2 Using LSF Parallel</b>	<b>15</b>
Writing a Distributed Application	16
Compiling and Linking the Application	17
Running the Application	18
Submitting to LSF	18
Executing interactively	18
<b>3 Building Parallel Applications</b>	<b>19</b>
Including the Header File	20
Include syntax	20
Compiling and Linking	21
C programs	21
Fortran 77 programs	21
Building a Heterogeneous Parallel Application	22
LSF host type naming convention	22
%a notation	23

<b>4</b>	<b>Submitting Parallel Applications</b>	<b>25</b>
	Job Submission Methods	26
	Batch execution	26
	Interactive execution	26
	Batch Execution	27
	Batch Job Status	28
	Job states	28
	Parallel batch job behavior	29
	Running and Controlling Batch Jobs	30
	Submitting jobs (bsub)	30
	The pam option	30
	Suspending Jobs (bstop)	30
	Resuming Jobs (bresume)	31
	Monitoring Job Status (bjobs)	31
	Terminating Jobs (bkill)	32
	Running Heterogeneous Parallel Applications	33
	Interactive Execution	34
	The pam Command	35
	Writing and using a PAM script	36
	Run time job resource usage collection	36
	Queue-level job control	37
	Runaway job cleanup	37
	Process Status Report	38
	Job states	38
	Getting Host Information	39
<b>A</b>	<b>Vendor MPI Implementations</b>	<b>41</b>
	HP MPI	42
	Automatic host allocation by LSF	42
	Running a job on a single host	42
	Running a job on multiple hosts	42
	More details on mpirun	42
	SGI MPI	43
	Compiling and linking your MPI program	43
	System requirements	43
	Configuring LSF to work with SGI MPI	44
	Using the -mpi option	44
	Signal propagation	45
	Limitations	45
	SUN HPC MPI	46
	IBM MPI	48
	Overview	48
	Submitting POE jobs in LSF with LSF Parallel	48
	Prerequisites	48
	Syntax	48
	Example	48
	OpenMP	49
	Overview	49
	Configuration	49
	Job submission	49
	<b>Index</b>	<b>51</b>

# Welcome

This document describes how to install, configure and use LSF Parallel. It also describes how to compile and link, execute, interact and monitor parallel applications submitted through Platform LSF.

For the most part, this guide does not repeat information that is available in detail elsewhere but focuses on what is specific to using the LSF Parallel system. References to more general sources are provided in “[Related publications](#)” on page 7 in this preface.

- Contents**
- ◆ “[About This Guide](#)” on page 6
  - ◆ “[Learning About Parallel Programming](#)” on page 7
  - ◆ “[Learning About Platform LSF](#)” on page 8

## About This Guide

### Who should read this book

This guide provides reference and tutorial material for:

- ◆ MPI programmers who want to compile and link MPI programs for use with LSF Parallel
- ◆ LSF Parallel users who want to submit (execute), monitor, and interact with parallel applications using Platform LSF

### What you should already know

The users of this guide are expected to be familiar with:

- ◆ Programming in the C or Fortran 77 language
- ◆ Message Passing Interface (MPI) concepts
- ◆ Platform LSF

### Typographical conventions

Typeface	Meaning	Example
Courier	The names of on-screen computer output, commands, files, and directories.	The <code>lsid</code> command
<b>Bold Courier</b>	What you type must be exactly as shown.	Type <code><b>cd /bin</b></code>
<i>Italics</i>	<ul style="list-style-type: none"> <li>◆ Book titles, new words or terms, or words to be emphasized.</li> <li>◆ Command-line place holders—replace with a real name or value.</li> </ul>	The queue specified by <i>queue_name</i>

### Command notation

Notation	Meaning	Example
Quotes " or '	Must be entered exactly as shown.	<code>"job_ID[index_list]"</code>
Commas ,	Must be entered exactly as shown.	<code>-C time0,time1</code>
Ellipsis ...	The argument before the ellipsis can be repeated. Do not enter the ellipsis.	<code>job_ID ...</code>
lower case italics	The argument must be replaced with a real value you provide.	<code>job_ID</code>
OR bar	You must enter one of the items separated by the bar. You cannot enter more than one item. Do not enter the bar.	<code>[-h   -V]</code>
Parenthesis ( )	Must be entered exactly as shown.	<code>-X "exception_cond([params]):action] ...</code>
Option or variable in square brackets [ ]	The argument within the brackets is optional. Do not enter the brackets.	<code>lsid [-h]</code>
Shell prompts	<ul style="list-style-type: none"> <li>◆ C shell: %</li> <li>◆ Bourne shell and Korn shell: \$</li> <li>◆ root account: #</li> </ul> Unless otherwise noted, the C shell prompt is used in all command examples.	<code>% cd /bin</code>

# Learning About Parallel Programming

## Related publications

This guide focuses on using parallel applications with the LSF Parallel product. It assumes familiarity with Platform LSF and the MPI standard. The following materials provide useful background about using Platform LSF and MPI.

### MPI and parallel programming

The following documents are available at your local bookstore:

- ◆ *MPI: The Complete Reference*, by Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra (MIT Press, 1995)
- ◆ *Using MPI*, by William Gropp, Ewing Lusk and Anthony Skjellum (MIT Press, 1994)
- ◆ *Parallel Programming with MPI*, by Peter Pacheco (Morgan Kaufmann Publishers, Inc., 1997)
- ◆ *Designing and Building Parallel Programs*, Ian Foster (Addison-Wesley, 1995)

### MPI standard

This document is available on the world wide web. *MPI: A Message-Passing Interface Standard*, Message Passing Interface Forum (University of Tennessee, 1995)

[www-unix.mcs.anl.gov/mpi/](http://www-unix.mcs.anl.gov/mpi/)

## Learning About Platform LSF

### World Wide Web and FTP

The latest information about all supported releases of Platform LSF is available on the Platform Web site at <http://www.platform.com>. Look in the Online Support area for current README files, Release Notes, Upgrade Notices, Frequently Asked Questions (FAQs), Troubleshooting, and other helpful information.

The Platform FTP site (<ftp.platform.com>) also provides current README files and Release Notes for all supported releases of Platform LSF.

If you have problems accessing the Platform web site or the Platform FTP site, send email to [info@platform.com](mailto:info@platform.com).

### Platform LSF manuals

All of the LSF manuals are available in HTML and PDF format on the Platform Web site at [www.platform.com/lsf\\_docs/](http://www.platform.com/lsf_docs/):

### Technical support

Contact Platform or your LSF Parallel vendor for technical support. Use one of the following to contact Platform technical support:

Email [support@platform.com](mailto:support@platform.com)

Toll-free phone ♦ 1-877-444-4LSF (+1 877 444 4573)

When contacting Platform, please include the full name of your company.

### We'd like to hear from you

If you find errors in any Platform documentation, or you have suggestions for improving it, please let us know. Contact [doc@platform.com](mailto:doc@platform.com).

# About LSF Parallel

- Contents
- ◆ “What Is LSF Parallel?” on page 10
  - ◆ “How Does LSF Parallel Work with Platform LSF?” on page 11
  - ◆ “LSF Parallel Architecture” on page 13

## What Is LSF Parallel?

LSF Parallel is a fully supported commercial software system that supports the programming, testing, and execution of applications in production environments.

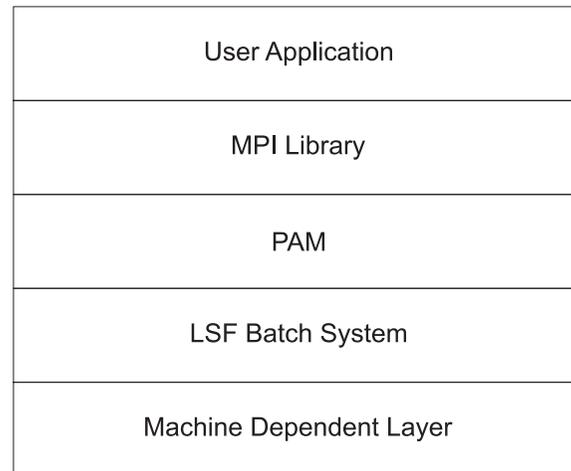
LSF Parallel is fully integrated with the Platform LSF, the de-facto industry standard resource management software product, to provide load sharing in a distributed system and batch scheduling for compute-intensive jobs. LSF Parallel provides support for:

- ◆ Dynamic resource discovery and allocation (resource reservation) for parallel batch job execution
- ◆ Transparent invocation of the distributed job processes across different platforms such as AIX, HP, Linux, SGI, and Solaris
- ◆ Full job-level control of the distributed processes to ensure no processes will become un-managed. This effectively reduces the possibility of one parallel job causing severe disruption to an organization's computer service
- ◆ The standard MPI interface
- ◆ All major UNIX operating systems
- ◆ Full integration with Platform LSF, providing heterogeneous resource-based batch job scheduling including job-level resource usage enforcement

## How Does LSF Parallel Work with Platform LSF?

LSF Parallel adopts a layered approach, shown below, that is fully integrated with Platform LSF. In addition to LSF system resources, the following components make up LSF Parallel:

- ◆ The MPI Library
- ◆ The Parallel Application Manager (PAM)
- ◆ Parallel Task Launcher



### MPI library

The Message Passing Interface (MPI) library is a message-passing library that must be linked to the parallel applications that are to be run in Platform LSF. The MPI library translates MPI message calls to messages for the machine-dependent layer and it interfaces the user application to PAM.

See Appendix A, “[Vendor MPI Implementations](#)” for details.

### PAM

The Parallel Application Manager (PAM) is the point of control for LSF Parallel. PAM is fully integrated with Platform LSF. PAM interfaces the user application with LSF. For all parallel application processes (tasks), PAM:

- ◆ Maintains the communication connection map
- ◆ Monitors and forwards control signals
- ◆ Receives requests to add, delete, start, and connect tasks
- ◆ Monitors resource usage while the user application is running
- ◆ Enforces job-level resource limits
- ◆ Collects resource usage information and exit status upon termination
- ◆ Handles standard I/O

## Platform LSF

Platform LSF is a sophisticated resource-based batch job scheduling system. It accepts user jobs and holds them in queues until suitable hosts are available and resource requirements are satisfied. Host selection is based on up-to-the-minute load information provided by the master Load Information Manager (LIM).

LSF runs user jobs on batch server hosts. It has sophisticated controls for sharing hosts with interactive users; there is no need to set aside dedicated hosts for processing batch jobs.

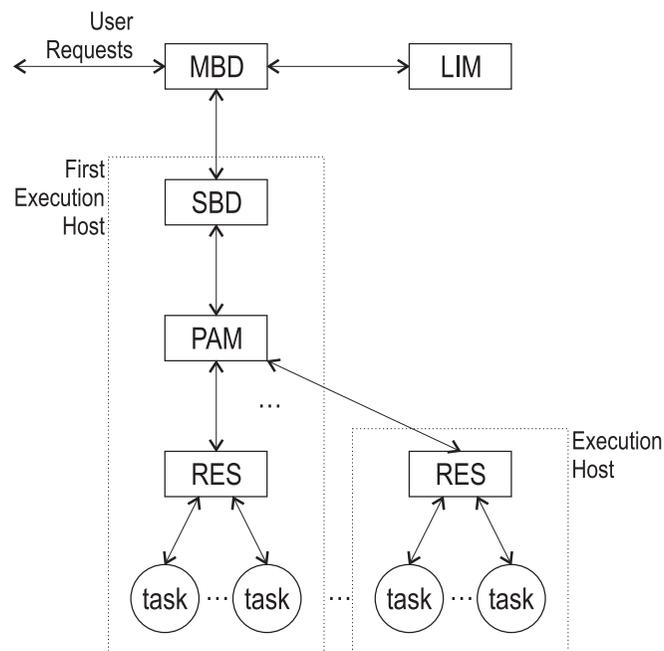
See *Administering Platform LSF* and the *Platform LSF Reference* for a detailed description of Platform LSF.

## LSF Parallel Architecture

LSF Parallel takes full advantage of the resources of LSF for resource selection and batch job process invocation and control:

- 1 User submits a parallel batch job to LSF
- 2 MBD retrieves a list of suitable execution hosts from the master LIM
- 3 MBD allocates (schedules, reserves) the execution hosts for the parallel batch job
- 4 MBD dispatches the parallel batch job to the SBD on the first execution host that was allocated to the batch job
- 5 SBD starts PAM on the same execution host
- 6 PAM starts RES on each execution host allocated to the batch job
- 7 RES starts the tasks on each execution host

This process is illustrated in the following diagram:



Platform LSF also supports interactive parallel job submission. The process is similar to that shown in the previous figure, except the user request is submitted directly to PAM which makes a simple placement query to LIM. Job queuing and resource reservations are not supported in interactive mode.

## LSF Parallel components

- User request** Batch job submission to LSF using the `bsub` command.
- mbatchd** Master Batch Daemon (MBD) is the policy center for LSF. It maintains information about batch jobs, hosts, users, and queues. All of this information is used in scheduling batch jobs to hosts.
- LIM** Load Information Manager is a daemon process running on each execution host. LIM monitors the load on its host and exchanges this information with the master LIM.
- The master LIM resides on one execution host and collects information from the LIMs on all other hosts in the LSF cluster. If the master LIM becomes unavailable, another host will automatically take over.
- For batch submission the master LIM provides this information to the MBD.
- For interactive execution the master LIM provides simple placement advice.
- sbatchd** Slave Batch Daemons (SBDs) are batch job execution agents residing on the execution hosts. SBD receives jobs from the MBD in the form of a job specification and starts RES to run the job according the specification. SBD reports the batch job status to the MBD whenever job state changes.
- PAM** The Parallel Application Manager is the point of control for LSF Parallel. PAM is fully integrated with Platform LSF. PAM interfaces the user application with the LSF system.
- If PAM or its host crashes, each RES will terminate all tasks under its management. This avoids the problem of orphaned processes.
- RES** The Remote Execution Servers reside on each execution host. RES manages all remote tasks and forwards signals, standard I/O, resources consumption data, and parallel job information between PAM and the tasks.
- Application task** The individual process of a parallel application
- Execution hosts** The most suitable hosts to execute the batch job as determined by LSF
- First execution host** The host name at the top of the execution host list as determined by LSF

## Using LSF Parallel

This chapter introduces the concepts needed to start using LSF Parallel. They are: compiling, linking, and submitting parallel applications. The example used in this chapter is a distributed version of the C program *Hello World* named `myjob`.

If the commands cannot be executed or the man pages cannot be viewed, the appropriate directories may need to be added to the systems path; ask your system administrator for help.

- Contents**
- ◆ “[Writing a Distributed Application](#)” on page 16
  - ◆ “[Compiling and Linking the Application](#)” on page 17
  - ◆ “[Running the Application](#)” on page 18

## Writing a Distributed Application

This example program, written in C, is a distributed version of the *Hello World* program named `myjob`. Use an editor to enter the code for this application. After the code is entered, save it in a file named `myjob.c`

Note that the command-line argument should be passed inside the program.

```
/*
 * File: myjob.c
 */
#include <stdio.h>
#include "mpi.h"                /* MPI header file */

int
main(int argc, char **argv)
{
    int myrank;                /* Rank of this process */
    int n_processes;          /* Number of Processes */
    int srcrank;              /* Rank of the Sender */
    int destrand;            /* Rank of the receiver */
    char mbuf[512];          /* Message buffer */
    MPI_Status mstat;        /* Return Status of an MPI operation */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &n_processes);

    if (myrank != 0) {
        sprintf(mbuf, "Hello, from process %d!", myrank);
        destrand = 0;
        MPI_Send(mbuf, strlen(mbuf)+1, MPI_CHAR,
                 destrand, 90, MPI_COMM_WORLD);
    } else {
        for (srcrank = 1; srcrank < n_processes; srcrank++) {
            MPI_Recv(mbuf, 512, MPI_CHAR,
                     srcrank, 90, MPI_COMM_WORLD, &mstat);
            printf("From process %d: %s\n", srcrank, mbuf);
        }
    }
    MPI_Finalize();
}
```

## Compiling and Linking the Application

After the example program is entered and saved as `myjob.c`, use the `mpicc` script to compile and link the application. The `mpicc` script is used in a similar manner to other UNIX-based C compilers. This script provides the options and special libraries needed to compile and link a parallel application for the Platform LSF environment.

To compile and link the source code in the `myjob.c` file in one step, enter the following command:

```
% mpicc myjob.c -o myjob
```

The binary created is called `myjob`.

## Running the Application

### Submitting to LSF

To submit the parallel application `myjob` to Platform LSF, requesting three processors, enter the following command:

```
% bsub -n 3 pam myjob
Job <1288> is submitted to default queue <normal>.
```

This command creates three processes and each runs an instance of `myjob`. The `bsub` command has a number of command line options, which are discussed in more detail in [“Running and Controlling Batch Jobs”](#) on page 30. To view the status of the parallel batch job, enter the following command:

```
% bjobs
JOBID USER   STAT  QUEUE      FROM_HOST  EXEC_HOST  JOB_NAME   SUBMIT_TIME
1288  user1  PEND  normal     hopper     host1      myjob      Apr 16 14:43
                               host2
                               host3
```

The `bjobs` command has a number of command line options, which are discussed in more detail in [“Monitoring Job Status \(bjobs\)”](#) on page 31.

### Executing interactively

To interactively execute the parallel application `myjob` on three processors, enter the following command

```
% pam -n 3 myjob
From process 1: Hello, from process 1!
From process 2: Hello, from process 2!

TID  HOST_NAME  COMMAND_LINE  STATUS  TERMINATION_TIME
===  =====  =====  =====  =====
0001 host1     myjob        Done    04/16/98 15:05:56
0002 host2     myjob        Done    04/16/98 15:05:56
0003 host3     myjob        Done    04/16/98 15:05:56
```

The `pam` command has a number of command line options, which are discussed in more detail in [“The pam Command”](#) on page 35.

## Building Parallel Applications

LSF Parallel provides tools to help build a parallel application to take full advantage of Platform LSF. Most parallel applications can be reused by simply re-linking with the PAM-aware MPI library, and in some instances there may not even be a need to re-compile.

This chapter discusses the basic steps in building a parallel application, the basic structure of the application and how it is compiled and linked.

This chapter focuses on building a parallel application to make optimal use of Platform LSF. It assumes familiarity with Platform LSF and standard MPI. Therefore it does not discuss writing MPI programs.

- Contents**
- ◆ “[Including the Header File](#)” on page 20
  - ◆ “[Compiling and Linking](#)” on page 21
  - ◆ “[Building a Heterogeneous Parallel Application](#)” on page 22

## Including the Header File

A set of PAM-aware header files are included with Platform LSF installation. They are typically located in the `LSF_INCLUDEDIR/lsf/mpi/` directory. The header files contain the MPI definitions, macros, and function prototypes necessary for using LSF Parallel.

### Include syntax

The include syntax must be placed at the top of any parallel application that calls MPI routines. The include statement looks like this in C applications:

```
#include <mpi.h>
```

In Fortran 77 applications:

```
INCLUDE "mpif.h"
```

If the header files are not located in the `LSF_INCLUDEDIR/lsf/mpi/` directory, check with your system administrator.

## Compiling and Linking

LSF Parallel provides a set of scripts that help with the creation of executable objects. They are:

- ◆ `mpicc` for C programs
- ◆ `mpif77` for Fortran 77 programs.

These scripts provide the options and special libraries needed to compile and link MPI programs for use with LSF Parallel. Applications are linked to system-dependent libraries and the appropriate MPI library.

### C programs

The LSF Parallel C compiler, `mpicc`, is used to compile MPI C source files. It is used in a similar manner to other UNIX-based C compilers. For example, to compile the sample program contained in a file `myjob.c` enter:

```
% mpicc -c myjob.c
```

This command produces the `myjob.o` that contains the object code for this LSF Parallel source file. To link the `myjob.o` object file with the LSF Parallel libraries to create an executable, enter:

```
% mpicc -o myjob myjob.o
```

As with most C compilers, the `-o` flag specifies that the name of the executable produced by the linker is to be `myjob`. The C source file can be compiled and linked in one step using the following command:

```
% mpicc myjob -o myjob
```

### Fortran 77 programs

The LSF Parallel Fortran 77 compiler, `mpif77`, is used to compile MPI Fortran 77 source files. It is used in a similar manner to other UNIX-based Fortran 77 compilers. For example, to compile the sample program contained in a file `myjob.f` enter:

```
% mpif77 -c myjob.f
```

This command produces the `myjob.o` that contains the object code for this LSF Parallel source file.

To link the `myjob.o` object file with the LSF Parallel libraries to create an executable, enter:

```
% mpif77 -o myjob myjob.o
```

As with most Fortran 77 compilers, the `-o` flag specifies that the name of the executable produced by the linker is to be `myjob`.

The Fortran 77 source file can be compiled and linked in one step using the following command:

```
% mpif77 myjob -o myjob
```

## Building a Heterogeneous Parallel Application

LSF Parallel provides a host type substitution facility to allow a heterogeneous multiple-architecture distributed application to be submitted to LSF. The following steps outline how to build and deploy a heterogeneous application:

- 1 Design the parallel application.
- 2 Compile the application on all LSF host-type architectures that will be used to support this application.  
The binaries must either be named with valid LSF host-type extensions or placed in directories named with valid LSF host-type path names.
- 3 Place binaries in the appropriate shared file system or distribute them accordingly.
- 4 Use the %a annotation to submit the parallel application to LSF.

### LSF host type naming convention

Binaries must be compiled on the target host type architectures. The binary must be named using a valid LSF host type string as the extension to its name or the name of a directory in its path (`lshosts` displays a list of valid LSF host types). When the %a notation is used to submit a parallel application to LSF the target host type string is substituted.

All binaries for a specific application must be named using the same host type substitution format (i.e., binary extension or path name).

For example, the following binaries are named with appropriate host type extensions to identify the target platform on which they are to run. These binaries are named to use Sun Solaris and RS6000 architecture machines:

- ◆ `myjob.SUNSOL`
- ◆ `myjob.RS6K`

For example, the following binaries are named with appropriate path names to identify the target platform on which they are to run. These binaries are named to use Sun Solaris and RS6000 architecture machines:

- ◆ `/user/batch/SUNSOL/myjob`
- ◆ `/user/batch/RS6K/myjob`

## %a notation

After a parallel application is submitted to LSF, the Parallel Application Manager (PAM) replaces the %a annotation with the appropriate LSF host type string. PAM then launches the individual tasks of the application on the remote hosts using the correct binaries. Use the `lshosts` command to determine which LSF hosts are available. For example:

```
% lshosts
HOST_NAME  type    model    cpuf  ncpus  maxmem  maxswp  server    RESOURCES
host1      SUNSOL  SunSparc 6.0   1      64M     112M    Yes (solaris cserver)
host2      RS6K    IBM350   7.0   1      64M     124M    Yes      (cserver aix)
```

For example, to submit the `myjob` application from the same directory using LSF host type extensions the following command is used:

```
% pam -n 2 myjob.%a
```

PAM will make the following substitutions for the %a notation:

- ◆ `myjob.SUNSOL`
- ◆ `myjob.RS6K`

For example, to submit the `myjob` application from different directories using host type path names the following command is used:

```
% pam -n 2 /user/batch/%a/myjob
```

PAM will make the following substitutions for the %a notation:

- ◆ `/user/batch/SUNSOL/myjob`
- ◆ `/user/batch/RS6K/myjob`



# Submitting Parallel Applications

- Contents
- ◆ “Job Submission Methods” on page 26
  - ◆ “Batch Execution” on page 27
  - ◆ “Interactive Execution” on page 34

## Job Submission Methods

LSF Parallel supports batch submission of parallel applications (batch jobs) using the facilities of Platform LSF. Interactive execution of parallel applications is also supported under control of the Parallel Application Manager (PAM).

An extensive and flexible set of tools is provided that allows parallel applications to be submitted through Platform LSF. Parallel applications can also be executed interactively under control of the Parallel Application Manager (PAM). These tools allow the specification of how, when, and where a parallel application is to be run.

### Batch execution

When submitting a parallel batch job, LSF Parallel uses the advanced features of LSF to select, submit, and interact with the individual tasks of the parallel batch job. The batch job is submitted to a queue using the `bsub` command and LSF attends to the details.

A parallel batch job is submitted to a queue, where it waits until it reaches the front of the queue and the appropriate resources become available. Then the batch job will be dispatched to the most suitable hosts for execution. This sophisticated queuing system allows batch jobs to run as soon as the suitable host resources becomes available.

To use the `bsub` command to submit a parallel batch job to LSF, see [“Running and Controlling Batch Jobs”](#) on page 30.

**Note** The batch job may not be run immediately, it may be queued until the appropriate resources become available.

### Interactive execution

When interactively executing a parallel batch job, the `pam` command is used to invoke PAM. When submitting batch jobs using the `pam` command, LSF is bypassed; the jobs are not queued. Batch jobs are run immediately upon entering the command if the specified resource requirements are met. If the resources are not available the job is not run.

Since the jobs do not wait, interactive job execution is beneficial for debugging parallel applications. Direct interaction is supported. All the input and output is handled transparently between the local and execution hosts.

To use the `pam` command to execute a parallel batch job interactively, see [“Interactive Execution”](#) on page 34.

## Batch Execution

LSF Parallel uses the features of Platform LSF to select the most suitable hosts, submit, and interact with parallel batch jobs. The batch job is submitted to a queue using the `bsub` command, as described in “[Running and Controlling Batch Jobs](#)” on page 30, and Platform LSF and LSF Parallel attend to the rest.

Like serial batch jobs, parallel batch jobs pass through many states. See “[Batch Job Status](#)” on page 28.

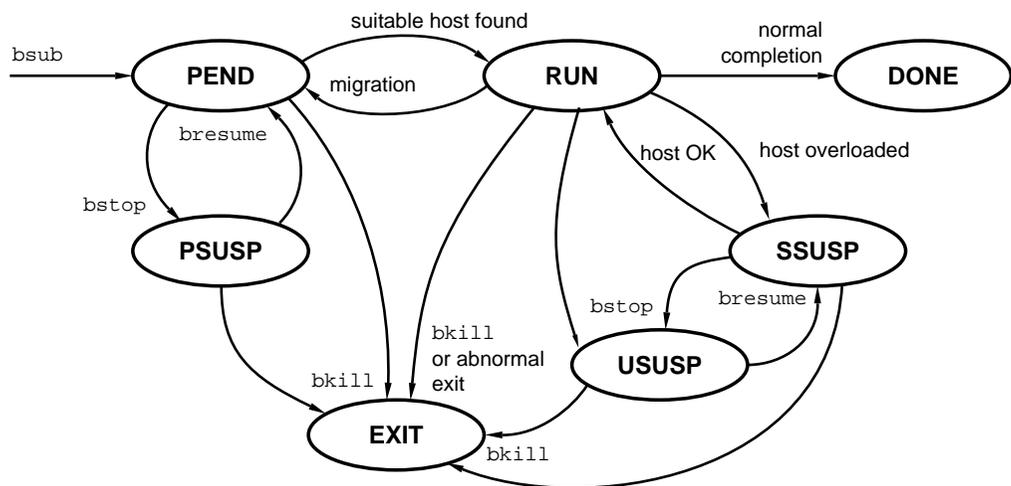
- In this section
- ◆ “[Batch Job Status](#)” on page 28
  - ◆ “[Running and Controlling Batch Jobs](#)” on page 30
  - ◆ “[Running Heterogeneous Parallel Applications](#)” on page 33

## Batch Job Status

Each batch job submitted to LSF passes through a series of states until the job completes normally (success) or abnormally (failure). The `bjobs` command allows the status of the batch jobs to be monitored; see “[Monitoring Job Status \(bjobs\)](#)” on page 31. The ability to monitor batch job status extends to the individual processes (tasks) of the parallel application.

### Job states

The following diagram shows the possible states a batch job can pass through when submitted to LSF. The diagram also shows the activities and commands that cause the state transitions. The batch job states are described below.



**PEND** A batch job is pending when it is submitted (using the `bsub` command) and waiting in a queue. It remains pending until it moves to the head of the queue and all conditions for its execution are met. The conditions may include:

- ◆ Start time specified by the user when the job is submitted
- ◆ Load conditions on qualified hosts
- ◆ Time windows during which:
  - ❖ The queue can dispatch jobs
  - ❖ Qualified hosts can accept jobs
- ◆ Relative priority to other users and jobs
- ◆ Availability of the specified resources

**RUN** A batch job is running when it has been dispatched to a host.

**DONE** A batch job is done when it has normally completed its execution.

- PSUSP** The job owner or the LSF administrator can suspend (using the `bstop` command) a batch job while it is pending.
- Also, the job owner or the LSF administrator can resume (using the `bresume` command) a batch that is in the PSUSP state, then the batch job state transitions to PEND.
- USUSP** The job owner or the LSF administrator can suspend (using the `bstop` command) a batch job after it has been dispatched.
- Also, the owner or the LSF administrator can resume (using the `bresume` command) a batch that is in the USUSP state, then the batch job state transitions to SSUSP.
- SSUSP** A batch job can be suspended by LSF after it has been dispatched. This is done if the load on the execution host or hosts becomes too high in order to maximize host performance or to guarantee interactive response time.
- LSF suspends batch jobs according to their priority unless the scheduling policy associated with the job dictates otherwise. A batch job may also be suspended if the job queue has a time window and the current time exceeds the window.
- LSF can later resume a system suspended (SSUSP) job if the load condition on the execution host decreases or the time window of the queue opens.
- EXIT** A batch job can terminate abnormally (fail) from any state for many reasons. Abnormal job termination can occur when:
- ◆ Cancelled (using the `bkill` command) by owner or LSF administrator while in PEND, RUN, or USUSP state
  - ◆ Aborted by LSF because job cannot be dispatched before a termination deadline
  - ◆ Fails to start successfully (e.g., the wrong executable was specified at time of job submission)
  - ◆ Crashes during execution

## Parallel batch job behavior

- ◆ When one task exits with a none-zero return value all the other tasks will run until they complete (DONE) or fail (EXIT)
- ◆ When one task is killed by a signal or core dumps, all the other tasks will be shut down

## Running and Controlling Batch Jobs

### Submitting jobs (bsub)

Use the `bsub` command to submit parallel batch jobs to LSF. The syntax for using `bsub` when submitting parallel applications is the same as LSF with the addition of the `pam` option:

```
bsub [options] pam [options] job
```

### The pam option

The `pam` options used with the `bsub` command are a subset of the `pam` command options, see “[The pam Command](#)” on page 35. Since LSF does all of the resource allocation and scheduling, the `pam` options `-m`, `-f`, and `-n` are not necessary and are ignored by the `bsub` command. The syntax for `bsub pam` is:

```
pam [-h][-V][-t][-v]
```

The `bsub pam` options are:

Option	Description
-h	Print command usage to <code>stderr</code> and exit.
-V	Print LSF version to <code>stderr</code> and exit.
-t	Suppress the printing of the process status summary on job completion.
-v	Specifies the job is to be run in verbose mode. The names of the selected hosts are displayed.

For example, the following command submits a parallel batch job named `myjob` to LSF and requests four processors of any type to run the job:

```
% bsub -n 4 pam myjob
```

When the parallel batch job named `myjob` is submitted to LSF and dispatched to `host1`, `host2`, `host3` and `host4`, the `bjobs` command will display:

```
% bjobs
JOBID USER      STAT  QUEUE          FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
713   user1      RUN   batch          host99     host1     myjob     Sep 12 16:30
                                     host2
                                     host3
                                     host4
```

### Suspending Jobs (bstop)

Use the `bstop` command to suspend parallel batch jobs running in LSF:

```
bstop jobId
```

For example, the following command suspends the parallel batch job named `myjob` running in LSF with job ID of 713:

```
% bstop 713
```

When the parallel batch job named `myjob` is suspended the `bjobs` command will display the batch job state of `USUSP`:

```
% bjobs
JOBID USER   STAT  QUEUE      FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
713  user1    USUSP batch     host99     host1     myjob     Sep 12 16:32
                               host2
                               host3
                               host4
```

## Resuming Jobs (`bresume`)

Use the `bresume` command to resume suspended parallel batch jobs running in LSF:

```
bresume jobID
```

For example, the following command resumes the suspended parallel batch job named `myjob` running in LSF with job ID of 713:

```
% bresume 713
```

When the parallel batch job named `myjob` is resumed the `bjobs` command will display the batch job state of `RUN` or `PEND`:

```
% bjobs
JOBID USER   STAT  QUEUE      FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
713  user1    RUN   batch     host99     host1     myjob     Sep 12 16:34
                               host2
                               host3
                               host4
```

## Monitoring Job Status (`bjobs`)

Use the `bjobs` command to view the running status and resource usage of parallel batch jobs running in LSF:

```
bjobs [options]
```

For example, the following command displays the running status and resource usage of the jobs running in LSF:

```
% bjobs
JOBID USER   STAT  QUEUE      FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
713  user1    RUN   batch     host99     host1     myjob     Sep 12 16:34
                               host2
                               host3
                               host4
```

For example, the following command uses the `-l` option to display run-time resource usage (CPU, memory, and swap) as well as the running status of the jobs running in LSF:

```
% bjobs -l
Job Id <713>, User, Project, Status, Queue, Interactive pseudo-terminal mode, Command
<myjob>

Thu Sep 12 16:39:17: Submitted from host <host99>, CWD <${HOME}/Work/utopia/pass/
pam>, 2-4 Processors Requested;
Thu Sep 12 16:39:18: Started on 4 Hosts/Processors host1 host2 host3 host4,
Execution Home < /pcc/s/user1, Execution CWD /pcc/s/user1/W
ork/utopia/pass/pam;
Thu Sep 12 16:40:41: Resource usage collected.
The CPU time used is 2 seconds.
MEM: 281 Kbytes; SWAP: 367 Kbytes
PGIDs: 4 PIDs: 4, 5, 6
PGIDs: 10 PIDs: 10, 11
PGIDs: 20 PIDs: 20, 21
PGIDs: 30 PIDs: 30, 31

SCHEDULING PARAMETERS:
      r15s  r1m  r15m  ut      pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -      -    -    -    -    -    -    -
loadStop  -    -    -    -      -    -    -    -    -    -    -
      nresj
loadSched -
loadStop  -
```

## Terminating Jobs (bkill)

Use the `bkill` command to terminate parallel batch jobs running in LSF:

```
bkill jobID [options]
```

For example, the following command terminates the parallel batch job named `myjob` running in LSF with a job ID of 713:

```
% bkill 713
```

When the parallel batch job named `myjob` is terminated the `bjobs` command will display the batch job state of `EXIT`:

```
% bkill 713
JOBID USER      STAT  QUEUE      FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
713  user1  EXIT  batch      host99    host1     myjob    Sep 12 16:30
                               host2
                               host3
                               host4
```

The time taken to terminate a parallel batch job varies and depends on the number of parallel processes.

## Running Heterogeneous Parallel Applications

LSF Parallel provides an LSF host type substitution facility to allow a heterogeneous multiple-architecture distributed application to be submitted to LSF.

- Assumptions**
- 1 The binary will run on each specified platform, or a binary exists for each platform.
  - 2 The binaries for the parallel application are specified using the %a notation format, see “[Building a Heterogeneous Parallel Application](#)” on page 22.

**Examples** For example, using the LSF host type extension format to specify the batch job named `myjob` to run on any two available processors having either Sun Solaris (SUNSOL) or RS6000 (RS6K) architectures, the following command can be used:

```
% bsub -n 2 pam myjob.%a
```

To specify SUNSOL and RS6K in an environment with other architectures, the following command is specified with the `-R` (resource) option:

```
% bsub -n 2 -R "type==SUNSOL || type==RS6K" pam myjob.%a
```

For both these examples, the Parallel Application Manager (PAM) substitutes the %a notation with the correct LSF host type extension. The binaries used are named:

- ◆ `myjob.SUNSOL`
- ◆ `myjob.RS6K`

For example, using the LSF host type path name format to specify the batch job named `myjob` to run on any two processors having either SUNSOL or RS6K architectures, the following command can be used:

```
% bsub -n 2 pam /user/batch/%a/myjob
```

To specify SUNSOL and RS6K in an environment with other architectures, the following command is specified with the `-R` (resource) option:

```
% bsub -n 2 -R "type==SUNSOL || type==RS6K" pam /user/batch/%a/myjob
```

For both these examples, the Parallel Application Manager (PAM) substitutes the %a notation with the correct LSF host type path name. The paths used to select the binaries are:

- ◆ `/user/batch/SUNSOL/myjob`
- ◆ `/user/batch/RS6K/myjob`

## Interactive Execution

LSF Parallel uses the Parallel Application Manager (PAM) to control the execution of parallel batch jobs interactively. Batch jobs are executed interactively using the `pam` command, see “[The pam Command](#)” on page 35. When submitting batch jobs using the `pam` command, Platform LSF is bypassed, the jobs are not queued. Batch jobs are run immediately upon entering the command if the resource requirements specified are met. If the resources are not available the job is not run. Since the jobs do not wait, interactive job execution is beneficial for debugging parallel applications.

To successfully execute an interactive parallel batch job, the `pam` command must be reissued at a time when the resources are available. If specific resources are not requested Platform LSF will run the batch job on the least loaded hosts that meet the batch jobs criteria.

Direct interaction is supported. All the input and output is handled transparently between local and execution hosts. All job control signals (e.g., `ctrl+x`, `ctrl+z`, and `ctrl+l`) are propagated to the execution hosts; this allows interaction with the job as if it were being executed locally.

- In this section
- ◆ “[The pam Command](#)” on page 35
  - ◆ “[Process Status Report](#)” on page 38
  - ◆ “[Getting Host Information](#)” on page 39

## The pam Command

The `pam` command is used to interactively execute parallel batch jobs in LSF Parallel. A subset of the `pam` command is used as a command option for the `bsub` command (see “Submitting jobs (`bsub`)” on page 30). The syntax for using the `pam` command is:

```
pam [-h][-V][-i][-t][-v]
      [-server_addr location ]
      [| -server_jobid location ]
      [| -server_jobname location ]
      {-m "host ..." }
      { | [-R req] -n num }
      job [arg ...]
```

Option	Description
-h	Print command usage to <code>stderr</code> and exit.
-V	Print LSF version to <code>stderr</code> and exit.
-i	Specifies interactive operation mode, the user will be asked if application is to be executed on all hosts. If yes (y) the task is started on all hosts specified in the list. If no (n) the user must interactively specify the hosts.
-t	Suppress the printing of the job task summary report to the standard output at job completion.
-v	Specifies the job is to be run in verbose mode. The names of the selected hosts are displayed.
-server_addr <i>location</i>	Specifies the location of the PAM server. The location is specified in the <code>hostname:port_no</code> format.
-server_jobid <i>location</i>	Specifies the location of the PAM server. The location is specified using the <code>jobid</code> for the server PAM job.
-server_jobname <i>location</i>	Specifies the location of the PAM server. The location is specified using the <code>jobname</code> for the server PAM job.
-m " <i>host ...</i> "	Specifies the list of hosts on which to run the parallel batch job tasks. The number of host names specified indicates the number of processors requested. This option cannot be used with options <code>-R</code> or <code>-n</code> , and is ignored when <code>pam</code> is used as a <code>bsub</code> option.
[-R <i>req</i> ] -n <i>num</i>	Specifies the number of processors required to run the parallel job. This option cannot be used with option <code>-m</code> , and is ignored when <code>pam</code> is used as a <code>bsub</code> option.
-R <i>req</i>	Default: <code>r15s:pg</code> This option is ignored when <code>pam</code> is used as a <code>bsub</code> option.
<i>job</i> [ <i>arg ...</i> ]	The name of the parallel job to be run. This must be the last argument on the <code>pam</code> command line.

For example, the following command executes the parallel batch job named myjob on Platform LSF requesting four processors of any type:

```
% pam -n 4 myjob
TID  HOST_NAME  COMMAND_LINE  STATUS  TERMINATION_TIME
====  =====  =====  =====  =====
1    host1     myjob        Done    03/31/98 10:31:58
2    host2     myjob        Done    03/31/98 10:31:59
3    host3     myjob        Done    03/31/98 10:31:59
4    host4     myjob        Done    03/31/98 10:31:58
```

For example, the following command uses the -m option to execute the parallel batch job named myjob on host1, host2, and host3:

```
% pam -m "host1 host2 host3" myjob
TID  HOST_NAME  COMMAND_LINE  STATUS  TERMINATION_TIME
====  =====  =====  =====  =====
1    host1     myjob        Done    03/31/98 10:31:58
2    host2     myjob        Done    03/31/98 10:31:59
3    host3     myjob        Done    03/31/98 10:31:59
```

## Writing and using a PAM script

A PAM script is a shell script that runs a series of PAM jobs and sequential jobs. The PAM script can be any UNIX shell script or perl script.

**Example** The following file, pamScript, runs three PAM jobs:

```
# File: pamScript
#!/bin/sh

/bin/echo "First run ... "
pam -mpi -np 128 a.out
/bin/echo "Second run ... "
pam -mpi -np 128 a.out
/bin/echo "Third run ... "
pam -mpi -np 128 a.out
```

The script pamScript then runs as one batch job:

```
bsub -n 128 -R "select[mem >= 512 && swap > 1024]" -o myFile
pamScript
```

Each parallel job started by pam runs sequentially and each can handle several MPI jobs.

## Run time job resource usage collection

Resource usage is collected and accumulated separately for each instance of PAM that is running for the following resources:

- ◆ CPU Time
- ◆ Memory and Swap Space
- ◆ Process IDs and Process Group IDs

## Queue-level job control

User-defined job controls normally run on sequential LSF jobs. LSF Parallel also supports job control actions on processes contained in parallel jobs.

**PID information file** SBD saves the process IDs for LSF Parallel jobs in a PID information file in the `/tmp` directory. The name of the file is in the form:

```
/tmp/.job_file_name.jobID.pid_info
```

The file name is put into the LSF system environment variable `LSB_PIDINFO_FILE`. Your job control program can use the function `getenv("LSB_PIDINFO_FILE")` to get the PID information file name.

**SGI IRIX array services** On SGI IRIX, you can use array services commands to control the jobs with queue-level job controls or signals according to their process IDs.

For example, you can use the `array ps` command to find a job's array handle and use other array commands to do job control.

See the IRIX system documentation and the `array_services(5)` man page for more information about array services, the array services daemon and array session handling.

**Example** For example, LSF creates the following PID information file for a PAM script job:

```
% ls -a /tmp
.963933993.2244.pidInfo
```

The contents of the process information file might look like the following:

```
% cat /tmp/.963933993.2244.pidInfo
726694 678 726694
732073 726694 726694
732286 726694 726694
```

The first ID is process ID, the second is the parent process ID, and the third is the process group ID.

The parent of the first process is array services daemon (`arrayd`, in this example 678):

```
% ps -ef | grep 678
root    678    1  0  Jul 13  ?    0:00 /usr/etc/arrayd
```

## Runaway job cleanup

If the PAM that controls the parallel jobs dies or is accidentally killed, LSF detects the broken connection to PAM and terminates the jobs that PAM started.

## Process Status Report

After a parallel batch job terminates in a successful (Done) or failed (EXIT) state LSF Parallel displays the status of all the processes. For example:

```
% pam -n 4 myjob
TID  HOST_NAME  COMMAND_LINE  STATUS  TERMINATION_TIME
====  =====  =====  =====  =====
1    host1     myjob         Done    03/31/98 10:31:58
2    host2     myjob         Done    03/31/98 10:31:59
3    host3     myjob         Done    03/31/98 10:31:59
3    host4     myjob         Done    03/31/98 10:31:59
```

## Job states

The possible job states for parallel jobs are described below:

Status	Description
Done	Process successfully completed with exit code of 0
Exit ( <i>code</i> )	Process unsuccessfully completed with an exit code of <i>code</i>
Exit (status unknown)	Connection broken; exit status unknown
Killed by PAM ( <i>signal</i> )	PAM shutdown process using <i>signal</i>
Local RES died	RES died before process exited
Run	Process running
Runaway	Process is still running; cannot be killed by PAM
Signaled ( <i>signal</i> )	Process was terminated by <i>signal</i>
Suspend	Process suspended
Undefined	PAM unable to read process exit state
Unreachable	PAM is unable to reach host after broken connection. No way to determine the state of the process

**Note** Use the `-t` option of `pam` to suppress the process status report.

## Getting Host Information

The `lshosts` command is used to display information about LSF host configurations including name, type, model, CPU normalization factor, number of CPUs, total memory, and available resources. For example:

```
% lshosts
HOST_NAME  type      model      cpuf ncpus maxmem maxswp server RESOURCES
host1      SGI64     SGI4D35    2.0  1    96M   153M   Yes (lsf_js irix gla)
host99     SUNSOL    SunSparc  12.0  4   1024M 1930M   Yes (solaris cs bigmem)
host2      LINUX     I486_33    14.0  1    30M   64M    Yes (linux)
host7      SUN41     SPARCSLC   3.0   1    15M   29M    Yes (sparc bsd sun41)
host3      ALPHA~1   DEC5000    5.0   1    88M   384M   Yes (cs bigmem alpha gla)
host6      ALPHA~1   DEC5000    5.0   1    84M   350M   Yes (gla)
host4      SUNSOL    SunSparc  12.0  2   256M   733M   Yes (solaris cs bigmem)
host5      SGI       SGIINDIG   15.0  1    96M   300M   Yes (irix)
host8      SUNSOL    SunSparc  12.0  1    56M   90M    Yes (solaris cs bigmem)
```





## Vendor MPI Implementations

- Contents
- ◆ “HP MPI” on page 42
  - ◆ “SGI MPI” on page 43
  - ◆ “SUN HPC MPI” on page 46
  - ◆ “IBM MPI” on page 48
  - ◆ “OpenMP” on page 49

## HP MPI

When you use `mpirun` in stand-alone mode, you specify host names to be used by the MPI job.

### Automatic host allocation by LSF

To achieve better resource utilization, you can have LSF manage the allocation of hosts, coordinating the start-up phase with `mpirun`.

This is done by preceding the regular HP MPI `mpirun` command with:

```
% bsub pam -mpi
```

### Running a job on a single host

For example, to run a single-host job and have LSF select the host, the command:

```
% mpirun -np 14 a.out
```

is entered as:

```
% bsub pam -mpi mpirun -np 14 a.out
```

### Running a job on multiple hosts

For example, to run a multi-host job and have LSF select the hosts, the command:

```
% mpirun -f appfile
```

is entered as:

```
% bsub pam -mpi mpirun -f appfile
```

where `appfile` contains the following entries:

```
-h host1 -np 8 a.out  
-h host2 -np 4 b.out  
-h host1 -np 2 c.out
```

In this example, the hosts `host1` and `host2` are treated as symbolic names and refer to the actual hosts that LSF allocates to the job.

The `a.out` and `c.out` processes are guaranteed to run on the same host. The `b.out` processes may run on a different host, depending on the resources available and LSF scheduling algorithms.

### More details on `mpirun`

For a complete list of `mpirun` options and environment variable controls, refer to the `mpirun` man page and the *HP MPI User's Guide* version 1.4.

## SGI MPI

### Compiling and linking your MPI program

You must use the SGI IRIX C compiler (`cc` by default). You cannot use `mpicc` to build your programs.

For example, use the following compilation command to build the program `mpi_sgi`:

```
cc -g -n32 -mips3 -o mpi_sgi mpi_sgi.c -lmpi
```

### System requirements

SGI MPI has the following system requirements:

- ◆ Your SGI IRIX systems must be running IRIX 6.5 or higher with the latest operating system patches applied. Use the `uname` command to determine your system configuration. For example:

```
% uname -aR
IRIX64 hostA 6.5 6.5.7m 01200532 IP19
```

- ◆ SGI MPI 3.2.04 (MPT 1.3.0.3) released December 7 1999 or later with the latest patches applied. Use the `versions mpi` and `versions sma` commands to determine your installation. For example:

```
% versions mpi
```

```
I = Installed, R = Removed
```

Name	Date	Description
I mpi	06/20/2000	MPI 3.2.0.7 (MPT 1.4)
I mpi.books	06/20/2000	IRIS InSight MPI Documentation (3.2.0.7)
I mpi.books.mpi_manual	06/20/2000	MPT: MPI Programmer's Manual (3.2.0.7)
I mpi.hdr	06/20/2000	MPI 3.2.0.7 Headers
I mpi.hdr.lib	06/20/2000	MPI 3.2.0.7 Library Headers
I mpi.man	06/20/2000	MPI 3.2.0.7 Man Pages
I mpi.man.base	06/20/2000	MPI 3.2.0.7 Man Pages
I mpi.relnotes	06/20/2000	MPT 1.4 Release Notes
I mpi.relnotes.base	06/20/2000	MPT 1.4 Release Notes
I mpi.sw	06/20/2000	MPI 3.2.0.7 Software
I mpi.sw.mpirun	06/20/2000	MPI 3.2.0.7 Program Launcher
I mpi.sw32	06/20/2000	MPI 3.2.0.7 N32 Libraries
I mpi.sw32.lib	06/20/2000	MPI 3.2.0.7 N32 DSO Libraries
I mpi.sw64	06/20/2000	MPI 3.2.0.7 N64 Libraries
I mpi.sw64.lib	06/20/2000	MPI 3.2.0.7 N64 DSO Libraries

```
% versions sma
```

```
I = Installed, R = Removed
```

Name	Date	Description
I sma	06/20/2000	SMA 3.1.2.5 (MPT 1.4)
I sma.hdr	06/20/2000	SMA 3.1.2.5 Headers
I sma.hdr.lib	06/20/2000	SMA 3.1.2.5 Library Headers

I	sma.man	06/20/2000	SMA 3.1.2.5	Man Pages
I	sma.man.base	06/20/2000	SMA 3.1.2.5	Man Pages
I	sma.sw32	06/20/2000	SMA 3.1.2.5	N32 Libraries
I	sma.sw32.lib	06/20/2000	SMA 3.1.2.5	N32 DSO Libraries
I	sma.sw64	06/20/2000	SMA 3.1.2.5	N64 Libraries
I	sma.sw64.lib	06/20/2000	SMA 3.1.2.5	N64 DSO Libraries

## Configuring LSF to work with SGI MPI

To use 32-bit or 64-bit SGI MPI with LSF Parallel, set the following parameters in `lsf.conf`:

- ◆ Set `LSF_VPLUGIN` to the full path to the SGI MPI library `libxmpi.so`.  
For example:  
`LSF_VPLUGIN=/usr/lib32/libxmpi.so`
- ◆ `LSF_PAM_USE_ASH=Y` enables LSF to use the SGI IRIX Array Session Handler (ASH) to propagate signals to the parallel jobs.  
See the IRIX system documentation and the `array_session(5)` man page for more information about array sessions.

**libxmpi.so file permission** For PAM to access the `libxmpi.so` library, the file permission mode must be 755 (`-rwxr-xr-x`).

## Using the `-mpi` option

The `-mpi` option on the `bsub` and `pam` command line is equivalent to `mpirun` in the SGI environment. Arguments following `pam -mpi` are treated exactly the same as if `mpirun` were used.

**Running a job on a single host** To run a single-host job and have LSF select the host, the command:

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

is entered as:

```
% bsub -n 4 -m "hostA hostB" pam -mpi -auto_place a.out
```

**Running a Job on Multiple Hosts** To run a multihost job and have LSF select the hosts, the following command:

```
% mpirun -f appfile
```

is entered as:

```
% bsub -n 4 pam -mpi -f appfile
```

where `appfile` contains the following entries:

```
host1 -np 4 a.out
host2 -np 4 b.out
host1 -np 2 c.out
```

For a complete list of `mpirun` options and environment variable controls refer to the SGI `mpirun` man page.

## Signal propagation

Typically, signals received by SBD are sent directly to PAM, not to the individual parallel jobs started by PAM. LSF also allows you to signal currently running PAM jobs or all the jobs started by a PAM script.

Set `LSF_PAM_USE_ASH=Y` in `lsf.conf` to enable LSF to use the SGI IRIX Array Session Handler (ASH) to propagate signals to the parallel jobs.

See “[Writing and using a PAM script](#)” on page 36 for more information about PAM scripts.

See the IRIX system documentation and the `array_session(5)` man page for more information about array sessions.

## Limitations

- ◆ SBD and MBD take a few seconds to get the process IDs and process group IDs of the PAM jobs from the SGI MPI components. If you use `bstop`, `brresume`, or `bkill` before this happens, uncontrolled MPI child processes may be left running.
- ◆ If `SIGKILL` or `SIGTERM` is issued within the first 60 seconds of MPI job run time, some orphan processes may be left on the system.
- ◆ If a PAM script does not exit when it receives an abnormal exit code from `pam`, the script itself is not stopped by the run time limit job control action. For example, if a PAM script job is submitted with a run time limit, and the script exceeds this limit, the job control action only applies to the existing processes from the PAM script component that exceeded the limit. Unless the PAM script is designed to exit on the failure of the previous `pam`, the next `pam` that runs will start.
- ◆ The following signals are supported:
  - ❖ `SIGKILL`
  - ❖ `SIGINT`
  - ❖ `SIGQUIT`
  - ❖ `SIGTERM`
  - ❖ `SIGSTOP`
  - ❖ `SIGCONT`
  - ❖ `SIGURG`

The following signals are not fully supported:

  - ❖ `SIGTSTP`
  - ❖ `SIGUSER1`
  - ❖ `SIGUSER2`

## SUN HPC MPI

When running LSF jobs on Sun platforms, you can include the Sun-specific argument `-sunhpc` on the `bsub` command line, after any other `bsub` arguments. The following arguments to `-sunhpc` provide additional control over `bsub` behavior in a Sun HPC environment.

**-n processes** Specify the number of processes to run. Note that the `bsub -n` argument specifies the number of CPUs to be used for the job. For example, to start a 48-process interactive job on PAM-enabled queue `hpc` that will wrap over at least 4, and as many as 16, CPUs:

```
% bsub -I -n 4,16 -q hpc -sunhpc -n 48 jobname
```

Setting the minimum number of CPUs to a number greater than 1 raises the possibility that, if there are fewer CPUs available than the minimum number you specify, the job may fail to start. In this example, if fewer than 4 CPUs are available, the job will not start. You can avoid this potential problem by setting the minimum number of CPUs to 1. However, this introduces the potential cost to performance of having the processes wrapped over a smaller number of CPUs.

**-P host:port** Specify the PAM address of another job with which the new job should colocate. The PAM address is the TCP socket used for communications between the job and PAM. For example, to start a 4-CPU interactive job on PAM-enabled queue `hpc`:

```
% bsub -I -n 4 -q hpc -sunhpc -P Athos:123 jobname
```

The new job is colocated with the job whose PAM is running on host `Athos`, using port 123.

**-j job\_ID** Specify the job ID of another job with which the new job should colocate.

**-J job\_name** Specify the job name of another job with which the new job should colocate.

**-s** Specify that the job is to be spawned in the STOPPED state.

To identify processes in the STOPPED state, issue the `ps` command with the `-e1` argument:

```
orpheus 215 => ps -e1
F S  UID  PID  PPID  C PRI NI     ADDR          SZ    WCHAN  TTY    TIME CMD
19 T   0    0    0    0  0  0  SY  f0274e38      0             ?      0:00 sched
```

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

Note that, when spawning a process in the STOPPED state under LSF, the name of your program will not appear in the `ps` output. Instead, the stopped process will be identified as a `RES` daemon.

For example, to start a 1-CPU interactive job on PAM-enabled queue `hpc`, in the STOPPED state:

```
% bsub -I -n 1 -q hpc -sunhpc -s jobname
```

# IBM MPI

## Overview

The IBM MPI Integration for LSF Parallel enhances the LSF 4.0-IBM SP2 Integration to launch and control jobs through LSF Parallel. The POE is still the job launcher while PAM handles resource collection and job control.

LSF will automatically collect the resource use of tasks in the parallel job. You can also automatically manage parallel jobs and perform operations on them with commands such as `bstop`, `bresume`, etc.

## Submitting POE jobs in LSF with LSF Parallel

Use `pbsub` along with `pam -g 1` to submit a POE job in LSF and have LSF Parallel manage it. In this way, PAM launches the POE and collects resource usage for all running tasks in the parallel job. In addition, you can automatically control the job through LSF commands (`bstop`, `bresume`, etc).

## Prerequisites

If you use `pam -g 1`, the program to launch must be compiled using IBM's `mpicc` for MPI support.

You can still use PAM to launch your jobs by not including the `-g` option, but you will need to compile the job using using LSF `mpicc` for MPI support.

## Syntax

```
pbsub [bsub_options] [pam [-g 1]] [-poe program_name [program_options] [poe_options]]
```

where:

- g** Indicates to PAM that it is not the job launcher (the process responsible for starting tasks on hosts).
- 1** This value must always be 1. Number of arguments after `-poe` where PAM can find the parallel program name.

For additional details, see the `pbsub` man page.

## Example

To submit an LSF-managed POE job that uses the switch in user-space (`us`) mode, and runs on six processors:

```
% pbsub pam -g 1 -poe my_prog my_prog_arg -eulib us -euidvice css0 -procs 6
```

Previously parallel jobs launched by the IBM POE could not have their resources collected and could not be managed by LSF. With this integration, we wrap PAM around the POE to overcome this limitation.

# OpenMP

## Overview

LSF Parallel provides the ability to start parallel jobs that use OpenMP to communicate between process on shared-memory machines and MPI to communicate across networked and non-shared memory machines.

This implementation allows you to specify the number of machines and to reserve an equal number of processors per machine. When the job is dispatched, PAM will only start 1 process per machine.

**OpenMP specification** The OpenMP specifications are owned and managed by the OpenMP Architecture Review Board, see <http://www.openmp.org>.

## Configuration

Set `LSF_PAM_HOSTLIST_USE=unique` in `lsf.conf` or the job's environment.

## Job submission

Specify the number of processors and the number of processes per machine. For example, to reserve 32 processors and run 4 processes per machine:

```
% bsub -n 32 -R "span[ptile=4]" pam yourOpenMPJob
```

`yourOpenMPJob` will run across 8 machines ( $4/32=8$ ) and PAM will start 1 MPI process per machine.



# Index

## Symbols

%a notation 23

## B

batch job  
    interactive execution 35  
    monitor 31  
    resource usage 32  
    resume 31  
    submit 30  
    suspend 30  
    terminate 32  
batch job state 28  
batch job status 28  
bjobs 31  
bkill 32  
bresume 31  
bstop 30  
bsub, pam option 30

## C

C program, compile 21  
command syntax 6  
compile  
    C program 21  
    Fortran 77 program 21

## D

DONE 28

## E

execution host substitution 23  
EXIT 29

## F

Fortran 77 program, compile 21

## H

host type substitution 23  
HP MPI 42

## I

interactive execution, batch job 35

## J

job  
    interactive execution 35  
    monitor 31  
    resource usage 32  
    resume 31

submit 30  
suspend 30  
terminate 32

job state 28  
    DONE 28  
    EXIT 29  
    PEND 28  
    PSUSP 29  
    RUN 28  
    SSUSP 29  
    USUSP 29  
job status 28

## L

link  
    C program 21  
    Fortran 77 program 21  
lshosts command 39

## M

monitor, batch job 31  
MPI  
    HP 42  
    OpenMP 49  
    SGI 43  
    SUN HPC 46  
mpicc 21  
mpif77 21

## N

notation, %a 23

## O

OpenMP MPI 49

## P

pam  
    %a option 23  
    bsub option 30  
    command 35  
PEND 28  
PSUSP 29

## R

resource usage, batch job 32  
resume, batch job 31  
RUN 28

## S

SGI MPI 43  
SSUSP 29

submit, batch job 30  
substitution, host type 23  
SUN HPC MPI 46  
suspend, batch job 30  
syntax 6

T  
terminate, batch job 32

U  
USUSP 29