LAM/MPI Installation Guide
Version 7.1

The LAM/MPI Team
Open Systems Lab
http://www.lam-mpi.org/

September 14, 2004

Contents

Chapter 1

Who Should Read This Document?

This document is intended for people who need to configure, compile, and install LAM/MPI. Although the document is quite lengthy, most users will be able to get LAM up and running with minimal effort (and minimal reading). A quick-install chapter (Chapter 3, page 9) is available for the impatient, although we recommend reading at least the Release Notes (Chapter 4, page 11) and browsing the available configure options (Chapter 6, page 27).

If you are a user of LAM with no intention of ever building LAM yourself, this isn't the document for you. You should be reading the LAM/MPI User's Guide [10] for detailed information on how to use LAM/MPI.

Chapter 2

Introduction to LAM/MPI

This chapter provides a summary of the MPI standard and the LAM/MPI implementation of that standard.

2.1 About MPI

The Message Passing Interface (MPI) [1, 2], is a set of API functions enabling programmers to write high-performance parallel programs that pass messages between processes to make up an overall parallel job. MPI is the culmination of decades of research in parallel computing, and was created by the MPI Forum – an open group representing a wide cross-section of industry and academic interests. More information, including the both volumes of the official MPI standard, can be found at the MPI Forum web site.[1]

MPI is suitable for "big iron" parallel machines such as the IBM SP, SGI Origin, etc., but it also works in smaller environments such as a group of workstations. Since clusters of workstations are readily available at many institutions, it has become common to use them as a single parallel computing resource running MPI programs. The MPI standard was designed to support portability and platform independence. As a result, users can enjoy cross-platform development capability as well as transparent heterogenous communication. For example, MPI codes which have been written on the RS-6000 architecture running AIX can be ported to a SPARC architecture running Solaris with little or no modifications.

2.2 About LAM/MPI

LAM/MPI is a high-performance, freely available, open source implementation of the MPI standard that is researched, developed, and maintained at the Open Systems Lab at Indiana University. LAM/MPI supports all of the MPI-1 Standard and much of the MPI-2 standard. More information about LAM/MPI, including all the source code and documentation, is available from the main LAM/MPI web site.[2]

LAM/MPI is not only a library that implements the mandated MPI API, but also the LAM run-time environment: a user-level, daemon-based run-time environment that provides many of the services required by MPI programs. Both major components of the LAM/MPI package are designed as component frameworks – extensible with small modules that are selectable (and configurable) at run-time. This component framework is known as the System Services Interface (SSI). The SSI component architectures are fully documented in [3, 5, 6, 7, 8, 9, 10].

[1] http://www.mpi-forum.org/
[2] http://www.lam-mpi.org/

Chapter 3

For the Impatient

If you don't want to read the rest of the instructions, the following should build a minimal installation of LAM/MPI in most situations (note that some options will be disabled, particularly if supporting libraries are installed in non-default locations, such as Myrinet, BProc, Globus, PBS, etc.).

```
shell$ gunzip -c lam-7.1.tar.gz | tar xf -
shell$ cd lam-7.1

# Set the desired C, C++, and Fortran compilers, unless using the GNU compilers is sufficient
# (this example assumes a Bourne-like shell)
shell$ CC=cc
shell$ CXX=CC
shell$ FC=f77
shell$ export CC CXX FC

shell$ ./configure --prefix=/directory/to/install/in
# ...lots of output...

shell$ make
# ...lots of output...
shell$ make install
# ...lots of output...

# The following step is optional. Ensure that $prefix/bin is in in your $path so that
# LAM's newly-created ''mpicc'' can be found before running this step.
shell$ make examples
# ...lots of output...
```

If you do not specify a prefix, LAM will first look for `lamclean` in your path. If `lamclean` is found, it will use the parent of the directory where `lamclean` is located as the prefix. Otherwise, `/usr/local` is used (like most GNU software).

Now read Chapter 4 (page 11); it contains information about the new features of this release of LAM/MPI.

Chapter 4

Release Notes

This chapter contains release notes as they pertain to the configuration, building, and installing of LAM/MPI. The User's Guide [10] contains additional release notes on the run-time operation of LAM/MPI.

4.1 New Feature Overview

A full, high-level overview of all changes in the 7 series (and previous versions) can be found in the `HISTORY` file that is included in the LAM/MPI distribution.

This docuemntation was originally written for LAM/MPI v7.0. Changebars are used extensively throughout the document to indicate changes, updates, and new features in the versions since 7.0. The change bars indicate a version number in which the change was introduced.

Major new features specific to the 7 series include the following:

- LAM/MPI 7.0 is the first version to feature the System Services Interface (SSI). SSI is a "pluggable" framework that allows for a variety of run-time selectable modules to be used in MPI applications. For example, the selection of which network to use for MPI point-to-point message passing is now a run-time decision, not a compile-time decision. ⊤ (7.1)

  SSI modules can be built as part of the MPI libraries that are linked into user applications or as standalone dynamic shared objects (DSOs). When compiled as DSOs, all SSI modules are installed in `$prefix/lib/lam`; new modules can be added to or removed from an existing LAM installation simply by putting new DSOs in that directory (there is no need to recompile or relink user applications). ⊥ (7.1)

- When used with supported back-end checkpoint/restart systems, LAM/MPI can checkpoint parallel MPI jobs (see Section 6.4.2, page 36, for more details). ⊤ (7.1)

  LAM/MPI now also supports checkpointing and restarting the `gm` RPI module, but only when used with the GM 2.x function `gm_get()`, which is disabled by default. See Section 6.4.2, page 36, for more details. ⊥ (7.1)

- LAM/MPI supports the following underlying networks for MPI communication, including several run-time tunable-parameters for each (see Section 6.4.2, page 35) for configuration options and the User's Guide for tuning parameters):

  – TCP/IP, using direct peer-to-peer sockets

⊤ (7.1)
  - Myrinet, using the native gm message passing library
⊥ (7.1)
  - Infiniband, using the Mellanox Verbs API (mVAPI) library
  - Shared memory, using either spin locks or semaphores
  - "LAM Daemon" mode, using LAM's native run-time environment message passing

- LAM's run-time environment can now be "natively" executed in the following environments (see Section 6.4.2, page 35 for more details):
  - BProc clusters
  - Globus grid environments (beta level support)
  - Traditional `rsh` / `ssh`-based clusters
⊤ (7.1)
  - OpenPBS/PBS Pro/Torque batch queue jobs (See Section 4.2.2, page 13)
⊥ (7.1)
  - SLURM batch queue systems
⊤ (7.1)

- LAM can be configured to use one TCP network for "out-of-band" communications and another TCP network for MPI communications. This is typically useful in environments where multiple TCP networks with different operational purposes and characteristics are available. For example, it may be suitable to allow LAM's out-of-band traffic to travel on a "slow" network and reserve the "fast" network for MPI traffic.

  Although this configuration is performed at run-time; a network administrator can setup default be-
⊥ (7.1) havior. See Section 8.2 (page 44 for more details.

4.2 Usage Notes

4.2.1 Filesystem Issues

Case-insensitive filesystems. On systems with case-insensitive filesystems (such as Mac OS X with HFS+, Linux with NTFS, or Microsoft Windows(TM) (Cygwin)), the `mpicc` and `mpiCC` commands will both refer to the same executable. This obviously makes distinguishing between the `mpicc` and `mpiCC` wrapper compilers impossible. LAM will attempt to determine if you are building on a case-insensitive filesystem. If you are, the C++ wrapper compiler will be called `mpic++`. Otherwise, the C++ compiler will be called `mpiCC` (although `mpic++` will also be available).

The `--with-cs-fs` and `--without-cs-fs` flags to LAM's `configure` script can be used to force LAM to install as if it was on either a case-sensitive filesystem or a case-insensitive filesystem. See
⊤ (7.1) Section 6.4.1 for more details.

In Microsoft Windows(TM) (Cygwin), file and directory names are allowed to have spaces. As a consequence, environment variables such as `HOME` may contain spaces. Since occurrence of spaces in such
⊥ (7.1) variables cause problems with the build, it is advised to escape these variables.

NFS-shared `/tmp`. The LAM per-session directory may not work properly when hosted in an NFS directory, and may cause problems when running MPI programs and/or supplementary LAM run-time environment commands. If using a local filesystem is not possible (e.g., on diskless workstations), the use of `tmpfs` or `tinyfs` is recommended. LAM's session directory will not grow large; it contains a small amount of meta data as well as known endpoints for Unix sockets to allow LAM/MPI programs to contact the local LAM run-time environment daemon.

AFS and tokens/permissions. AFS has some peculiarities, especially with file permissions when using `rsh`/`ssh`.

Many sites tend to install the AFS `rsh` replacement that passes tokens to the remote machine as the default `rsh`. Similarly, most modern versions of `ssh` have the ability to pass AFS tokens. Hence, if you are using the rsh boot module with `recon` or `lamboot`, your AFS token will be passed to the remote LAM daemon automatically. If your site does not install the AFS replacement `rsh` as the default, consult the documentation on `--with-rsh` to see how to set the path to the `rsh` that LAM will use.

Once you use the replacement `rsh` or an AFS-capable `ssh`, you should get a token on the target node when using the rsh boot module.[1] This means that your LAM daemons are running with your AFS token, and you should be able to run any program that you wish, including those that are not `system:anyuser` accessible. You will even be able to write into AFS directories where you have write permission (as you would expect).

Keep in mind, however, that AFS tokens have limited lives, and will eventually expire. This means that your LAM daemons (and user MPI programs) will lose their AFS permissions after some specified time unless you renew your token (with the `klog` command, for example) on the originating machine before the token runs out. This can play havoc with long-running MPI programs that periodically write out file results; if you lose your AFS token in the middle of a run, and your program tries to write out to a file, it will not have permission to, which may cause Bad Things to happen.

If you need to run long MPI jobs with LAM on AFS, it is usually advisable to ask your AFS administrator to increase your default token life time to a large value, such as 2 weeks.

4.2.2 PBS Job Cleanup

OpenPBS and PBS Pro are batch queuing products from Altair Grid Technologies, LLC. Torque is an open source derrivative of Open PBS. LAM/MPI's tm boot SSI module provides interaction with both versions of PBS when launching the LAM run-time environment.

There are a number of limitations in OpenPBS, PBS Pro, and Torque with regards to proper job shutdown. There is a race condition in sending both a SIGKILL and SIGTERM to spawned tasks such that the LAM RTE may not be properly shutdown. A patch [2] for OpenPBS to force OpenPBS to send a SIGTERM was developed by the WGR/PDL Lab at Oxford University. Although the patch is for Linux only, it should be straight-forward to modify for other operating systems.

4.2.3 PBS Pro RPM Notes

The Altair-provided client RPMs for PBS Pro do not include the `pbs_demux` command, which is necessary for proper execution of TM jobs. The solution is to copy the executable from the server RPMs to the client nodes.

4.2.4 Root Execution Disallowed

It is a Very Bad Idea to run the LAM executables as `root`.

LAM was designed to be run by individual users; it was *not* designed to be run as a `root`-level service where multiple users use the same LAM daemons in a client-server fashion. The LAM run-time environment

[1] If you are using a different boot module, you may experience problems with obtaining AFS tokens on remote nodes.

[2] `http://bellatrix.pcl.ox.ac.uk/~ben/pbs/`

should be started by each individual user who wishes to run MPI programs. There are a wide array of security issues when `root` runs a service-level daemon; LAM does not even attempt to address any of these issues.

Especially with today's propensity for hackers to scan for `root`-owned network daemons, it could be tragic to run this program as `root`. While LAM is known to be quite stable, and LAM does not leave network sockets open for random connections after the initial setup, several factors should strike fear into system administrator's hearts if LAM were to be constantly running for all users to utilize:

1. LAM leaves a Unix domain socket open on each machine (usually under the `/tmp` directory). Hence, if `root` is compromised on one machine, `root` is effectively compromised on all machines that are connected via LAM.

2. There must be a `.rhosts` (or some other trust mechanism) for `root` to allow running LAM on remote nodes. Depending on your local setup, this may not be safe.

3. LAM has never been checked for buffer overflows and other malicious input types of errors. LAM is tested heavily before release, but never from a `root`-level security perspective.

4. LAM programs are not audited or tracked in any way. This could present a sneaky way to execute binaries without log trails (especially as `root`).

Hence, it's a Very Bad Idea to run LAM as `root`. LAM binaries will quit immediately if `root` runs them. Login as a different user to run LAM.

The only exception to this rule is the `recon` command. Since `recon` can be used to verify basic LAM functionality, it is useful to run this as `root`, and is therefore the only LAM executable that allows itself to be run as `root`.

⊤ (7.1)

4.2.5 Operating System Bypass Communication: Myrinet and Infiniband

Supported Versions

The gm rpi has been tested with GM version 2.0.13 and 1.6.4. Versions prior to this in the 2.0.x and 1.6.x series had bugs and are not advised. The gm rpi has not been tested in the GM 2.1.x series.

The ib rpi has been tested with Mellanox VAPI thca-linux-3.2-build-024. Other versions of VAPI, to include OpenIB and versions from other vendors, have not been well tested.

Memory Managers

The gm and ib RPI modules require an additional memory manager in order to run properly. On most systems, LAM will automatically select the proper memory manager and the system administrator / end user doesn't need to know anything about this. However, on some systems and/or in some applications, extra work is required.

The issue is that OS-bypass networks such as Myrinet and Infiniband require virtual pages to be "pinned" down to specific hardware addresses before they can be used by the Myrinet/Infiniband NIC hardware. This allows the NIC communication processor to operate on memory buffers independent of the main CPU because it knows that the buffers will never be swapped out (or otherwise be relocated in memory) before the operation is complete.[3]

[3] Surprisingly, this memory management is unnecessary on Solaris. The details are too lengthy for this document.

LAM performs the "pinning" operation behind the scenes; for example, if application MPI_SENDs a buffer using the gm or ib RPI modules, LAM will automatically pin the buffer before it is sent. However, since pinning is a relatively expensive operation, LAM usually leaves buffers pinned when the function completes (e.g., MPI_SEND). This typically speeds up future sends and receives because the buffer does not need to be [re-]pinned. However, if the user frees this memory, the buffer *must* be unpinned before it is given back to the operating system. This is where the additional memory manager comes in.

LAM will, by default, intercept calls to `malloc()`, `calloc()`, and `free()` by use of the ptmalloc, ptmalloc2, or Mac OS X dynlib functionality (note that C++ `new` and `delete` are *not* intercepted). However, this is actually only an unfortunate side effect: LAM really only needs to intercept the `sbrk()` function in order to catch memory before it is returned to the operating system. Specifically, an internal LAM routine runs during `sbrk()` to ensure that all memory is properly unpinned before it is given back to the operating system.

There is, sadly, no easy, portable way to intercept `sbrk()` without also intercepting `malloc()` et al. In most cases, however, this is not a problem: the user's application invokes `malloc()` and obtains heap memory, just as expected (and the other memory functions also function as expected). However, there are some applications do their own intercepting of `malloc()` (et al.). These applications will not work properly with a default installation of LAM/MPI.

To fix this problem, LAM allows you to disable all memory management, but only if the top-level application promises to invoke an internal LAM handler function when `sbrk()` is invoked (*before* the memory is returned to the operating system). This is accomplished by configuring LAM with the following switch:

```
shell$ configure --with-memory-manager=external ...
```

"`external`" specifically indicates that if the gm or ib RPI modules are used, the application promises to invoke the internal LAM function for unpinning memory as required. Note that this function is irrelevant (but harmless) when any other RPI module is used. The function that must be invoked is prototyped in `<mpi.h>`:

```
void lam_handle_free(void *buf, size_t length);
```

For applications that must use this functionality, it is probably safest to wrap the call to `lam_handle_free()` in the following preprocessor conditional:

```
#include <mpi.h>

int my_sbrk(...) {
  /* ...sbrk() functionality... */
#if defined(LAM_MPI)
  lam_handle_free(bufer, length);
#endif
  /* ...rest of sbrk() functionality... */
}
```

Note that when LAM is configured this way, *all* MPI applications that use the gm or ib RPI modules must invoke this function as required. Failure to do so will result in undefined behavior. ⊥ (7.1)

4.3 Build-Related Notes

4.3.1 Compiler Warnings

The LAM Team is aware of the following compiler warnings:

- Several warnings about undefined preprocessor macros may be emitted when compiling most of the GM RPI source files. This is a function of the GM library's main include file (`gm.h`), and is safe to ignore.

4.3.2 64-bit Compilation

LAM is 64-bit clean and regularly used in a variety of 64-bit environments. To compile LAM in 64-bit mode, you will likely need to specify additional compiler and linker flags. Each compiler/architecture has its own flags to enable 64 bit compilation; consult the documentation for your compiler. See Section 6.3 (page 28) for information on specifying compiler flags.

Flags used by the LAM development team during 64-bit testing are specified in the Platform-Specific Release Notes section.

4.3.3 C++ and Exceptions

A default build of LAM/MPI will not include support for C++ exceptions. Enabling C++ exceptions typically entails a slight run-time performance degradation because of extra bootstrapping required for every function call (particularly with the GNU compilers). As such, C++ exceptions are disabled by default, and using the MPI::ERRORS_THROW_EXCEPTIONS error handler will print out error messages rather than throw an exception. If full exception handling capabilities are desired, LAM must be configured with the `--with-exceptions` flag. It should be noted that some C++ (and C and Fortran) compilers need additional command line flags to properly enable exception handling.

For example, with `gcc`/`g++` 2.95.2 and later, `gcc`, `g77`, and `g++` all require the command line flag `-fexceptions`. `gcc` and `gf77` require `-fexceptions` so that they can pass C++ exceptions through C and Fortran functions properly. As such, *all* of LAM/MPI must be compiled with the appropriate compiler options, not just the C++ bindings. Using MPI::ERRORS_THROW_EXCEPTIONS without having compiled with proper exception support will cause undefined behavior.

If building with IMPI or the C++ bindings, LAM's configure script will automatically guess the necessary compiler exception support command line flags for the `gcc`/`g++` and `KCC` compilers. That is, if a user selects to build the MPI 2 C++ bindings and/or the IMPI extensions, and also selects to build exception support, and `g++` or `KCC` is selected as the C++ compiler, the appropriate exceptions flags will automatically be used.

The configure option `--with-exflags=FLAGS` is provided for other compilers that require command line flags for exception support, or if LAM's configuration script guesses the wrong compiler flags. See Section 6.4.1 for more information.

Note that this also applies even if you do not build the C++ bindings. If LAM is to call C++ functions that may throw exceptions (e.g,. from an MPI error handler or other callback function), you need to build LAM with the appropriate exceptions compiler flags.

4.3.4 Internal Unix Signal

LAM uses SIGUSR2 for internal control. The signal used is configurable; see Section 6.4.1 (page 29) for specific instructions how to change this signal.

4.3.5 Shared Libraries

LAM/MPI uses the GNU Libtool package for building libraries. While Libtool generally does a good job at building libraries across a wide variety of platforms, there are limitations with Libtool and shared libraries. The known issues with building LAM and shared libraries are documented in Section 4.4. If building shared libraries, please read the release notes for your platform.

The TotalView queue debugging support requires the ability to build shared libraries. If it is not possible to build shared libraries with your combination of platform and compiler, building the TotalView library will fail. Using the `--disable-tv-queue` configure flag will disable building the TotalView support shared library. ⊤ (7.1)

4.3.6 Dynamic SSI Modules

The term "Dynamic SSI modules" refers to SSI modules compiled as stand-alone shared libraries that are loaded by the LAM/MPI SSI framework at run-time. This is quite convenient from a system-administrator perspective; users won't need to recompile their MPI applications when a new network is added (and therefore a new module is added to an existing LAM/MPI installation), for example – the system administrator simply places the SSI shared library module in the correct directory and all LAM/MPI applications "see" it at run-time.

Be sure to also read Section 4.3.5 – all of the issues in that section also apply to dynamic SSI modules.

Also note that the GNU compilers prior to version 3.0 *cannot* build self-contained shared libraries on Solaris systems. The `configure` script will emit large warnings in such circumstances. Hence, attempting to build dynamic SSI modules on Solaris systems with old versions of GNU compilers will not work. ⊥ (7.1)

4.3.7 TotalView Support

LAM/MPI supports the TotalView parallel debugger, both for attaching parallel programs (including partial attach) and for message passing queue debugging. See the User's Guide [10] for more details.

In order to support the message queue debugging functionality, a LAM-specific shared library (`liblam_totalview`) must be built that the TotalView debugger can load at run-time. Hence, the queue-debugging functionality is only supported on platforms that support shared libraries.

The TotalView debugger itself is a 32-bit application. As such, `liblam_totalview` must also be built in 32-bit mode, regardless of what mode the LAM/MPI libraries and applications are built. To avoid complicated build scenarios, if LAM/MPI is being built in a non-32 bit mode, `liblam_totalview` will not be built or installed.

This behavior can be overridden with the `--enable-tv-dll-force`, which will force `liblam_totalview` to be built and installed, regardless of LAM's build mode. Using a non-32 bit TotalView shared library is not expected to work; this option is mainly for future expandability (for if TotalView debugger becomes able to load non-32 bit shared libraries).

4.3.8 VPATH Builds

LAM supports the "VPATH" building mechanism. If LAM/MPI is to be installed in multiple environments that require different options to configure, or require different compilers (such as compiling for multiple architectures/operating systems), the following form can be used to configure LAM:

```
shell$ cd /some/temp/directory
shell$ /directory/containing/lam-7.1/configure <option> ...
```

where `/directory/containing/lam-7.1` is the directory where the LAM/MPI distribution tarball was expanded. This form will build the LAM executables and libraries under `/some/temp/directory` and will not produce any files in the `/directory/containing/lam-7.1` tree. It allows multiple, concurrent builds of LAM/MPI from the same source tree.

Note that you must have a VPATH-enabled `make` in order to use this form. The GNU `make`[4] supports VPATH builds, for example, but the Solaris Workshop/Forte `make` does not.

4.3.9 Large Job Size Support

⊤ (7.0.3)

On many platforms, it is possible to compile an application to use a large number of file descriptors. This allows LAM/MPI applications using TCP to be much larger than was previously possible. The configure flag `--with-fd-setsize` allows users to raise the soft per-process file descriptor limit. The system administrator may be required to raise the hard per-process file descriptor limit. Be sure to see the platform-specific release notes for information on your particular platform. ⊥ (7.0.3)

4.3.10 Configure Cache Support (`config.cache`)

⊤ (7.1)

The use of `configure`'s "`-C`" flag to specify using a cachefile for the results of various configuration tests is unsupported. There are some complex interactions between configure and the environment as a result of LAM's SSI system that makes using the configure cache impossible. In short, each module has its own `configure` script, and LAM usually must override various flags that are passed to these sub-configure scripts. This unfortunately conflicts which the `config.cache` usage policies.

As a result, the `-C` option cannot be used with LAM's `configure` script. This will never create incorrect results, but it does force LAM's `configure` script (and all of its sub-`configure` scripts) to take a long time on some platforms. ⊥ (7.1)

4.4 Platform-Specific Release Notes

LAM 7.1 has been officially tested on the following systems:

AIX 5.1	Mac OS X 10.3
IRIX 6.5	OpenBSD 3.5
Linux 2.4.x	Solaris 8, 9
Windows(TM) (Cygwin)	

Even if your operating system is not listed here, LAM will likely compile and run properly. The source code is fairly portable; it should work properly under most POSIX-like systems.

[4] `ftp://ftp.gnu.org/gnu/make/`

The LAM Team also requests that you also download the "`lamtest`" package from the LAM web site and run it on your system. See Chapter 8 for more details. We would greatly appreciate your time and effort in helping to verify LAM/MPI on a wide variety of systems.

4.4.1 AIX

Building LAM/MPI as shared libraries is not well tested with the `xlc` compiler suite, and is not supported. With more recent releases of AIX and the compiler suite, it may be possible to build correct shared libraries on AIX. There have been reports of linker errors similar to those of Mac OS X when building with `--disable-static --enable-shared`. LAM will build properly but fail with strange errors from the LAM daemon involving `nsend`.

To build LAM/MPI in 64-bit mode with the `xlc`/`xlC`/`xlf` compilers, it is recommended that the environment variable `OBJECT_MODE` be set to `64` before running LAM's `configure` script. Building 64-bit mode with the GNU compiler is not tested, but should be possible.

The memory management code that is required for GM and InfiniBand support is disabled by default on AIX. On at least one version of AIX, it caused random failures during `free`. The functionality can be enabled with the `--with-memory-manager` configure option.

Finally, there have been repeatable problems with AIX's `make` when building ROMIO [11, 12]. This does not appear to be ROMIO's fault – it appears to be a bug in AIX's `make`. The LAM Team suggests that you use GNU `make` when building if you see failures when using AIX's `make` to avoid these problems.

4.4.2 HP-UX

It appears that the default C++ compiler on HP-UX (CC) is a pre-ANSI standard C++ compiler. As such, it will not build the C++ bindings package. The C++ compiler `aCC` should be used instead. See Section 6.3 for information on changing the default C++ compiler.

4.4.3 IRIX

Parts of LAM/MPI use the C++ Standard Template Library (STL). The IRIX Workshop compilers require an additional flag to compile STL code properly. The following flag must be added to the `CXXFLAGS` environment variable before running LAM's `configure` script: `-LANG:std`.

Note that setting `CXXFLAGS` will override any automatic selection of optimization flags. Hence, if you want the C++ code in LAM to be compiled with optimization, you will need to set both `-LANG:std` *and* any optimization flags in `CXXFLAGS`. For example:

```
shell$ CXXFLAGS=''-LANG:std -O3''
shell$ export CXXFLAGS
shell$ ./configure ...
```

4.4.4 Linux

LAM/MPI is frequently used on Linux-based machines (IA-32 and otherwise). Although LAM/MPI is generally tested on Red Hat and Mandrake Linux systems using recent kernel versions, it should work on other Linux distributions as well.

Red Hat 7.2 `awk`. There appears to be an error in the `awk` shipped with Red Hat 7.2 for IA-64, which will cause random failures in the LAM test suite. Rather than using make check, we recommend specifying the available RPIs manually: `make check MODES="usysv sysv tcp"`, replacing the list with whatever RPIs are available in your installation.

Portland Group Compilers. The Portland Group compiler suite requires special flags to build shared libraries which Libtool does not properly provide. It does not appear possible to build shared libraries (see Section 4.3.5). This has implications for building the TotalView queue debugging support, which must be built as a shared library. LAM will attempt to detect the Portland Group C compiler and disable the queue support automatically. This can be overridden using the `--enable-tv-queue` flag at configure time.

Intel Compilers Using the Intel compiler suite with unsupported versions of glibc may result in strange errors while building LAM. This is a problem with the compiler-supplied header files, not LAM. For more
⊤ (7.0.3) information, see either the LAM mail archives[5] or the Intel support pages.

As an example, on Red Hat 9, some versions of the Intel compiler will complain about the `<stropts.h>` and/or complain that System V semaphores and shared memory cannot be used. The problem is actually with replacement header files that ship with the Intel compiler, not with LAM (or `<stropts.h>` or System V semaphores or shared memory). Any solution included in this documentation would likely be outdated quickly. Contacting Intel technical support for workarounds and/or upgrading your Intel compiler may solve
⊥ (7.0.3) these problems.

Older Linux Kernels Note that kernel versions 2.2.0 through 2.2.9 had some TCP/IP performance problems. It seems that version 2.2.10 fixed these problems; if you are using a Linux version between 2.2.0 and 2.2.9, LAM may exhibit poor TCP performance due to the Linux TCP/IP kernel bugs. We recommend that you upgrade to 2.2.10 (or the latest version). See `http://www.lam-mpi.org/linux/` for a full
⊤ (7.0.3) discussion of the problem.

It is not possible to use `--with-fd-setsize` to increase the per-process file descriptor limit. The
⊥ (7.0.3) glibc header files hard code file descriptor related sizes to 1024 file descriptors.

4.4.5 Mac OS X

The Apple developer's CD does not include a Fortran compiler. If you do not wish to use Fortran, you need to tell LAM not to build the Fortran bindings. The `--without-fc` flag to LAM's `configure` script will tell LAM to skip building the Fortran MPI bindings.

The Apple dynamic linker works slightly differently than most Unix linkers. Due to a difference in symbol resolution when using the shared libraries, it is not possible to build *only* shared libraries. It is possible to build LAM/MPI with both shared and static libraries; the build system will automatically statically link the binaries that can not be dynamically linked.

In previous versions of LAM, it was necessary to pass an argument [6] to the ROMIO build system in order to properly compile. This is no longer needed; LAM will pass the correct flags to ROMIO automatically.

Libtool does not support building shared libraries with the IBM XL compilers on OS X. Therefore, LAM/MPI does not support building shared libraries using this configuration. Building LAM/MPI with
⊤ (7.1) shared libraries with the GCC suite is supported.

[5] `http://www.lam-mpi.org/MailArchives/`
[6] `--with-romio-flags=-DNO_AIO`

By default, LAM adds a hook into the dynamic memory management interface to catch page deallocations (used for OS-bypass devices like Myrinet) – except on Solaris, AIX, and Cygwin systems. This hook requires all MPI applications to be built with flat namespaces. If this causes problems for your application, you can disable the allocation hook with the configure option `--with-memory-manager=none`. ⊥ (7.1)

There are no published limits to the value specified to `--with-fd-setsize`. However, there are memory considerations so do not specify a limit much higher than needed. ⊤ (7.0.3) ⊥ (7.0.3)

4.4.6 OpenBSD

The `make` on some versions of OpenBSD requires the `-i` option for the clean target: `make -i clean`.

On some versions of OpenBSD, there appears to be a problem with the POSIX Threads library which causes a segfault in the `lamd`. If the `lamd` is dying, leaving a core file[7], it is recommended LAM be rebuilt without threads support (using the `--with-threads` option – See Section 6.4).

4.4.7 Solaris

To build LAM/MPI in 64-bit mode with the Sun Forte compilers, it is recommended that the compiler option `-xarch=v9` be used. See Section 6.3 for information on setting compiler flags.

Building shared libraries in 64-bit mode with the Sun Forte compilers is not possible due to a bug in the GNU Libtool package. Specifically, Libtool does not pass `CFLAGS` to the linker, which causes the linker to attempt to build a 32-bit shared object. One possible workaround (which has not been tested by the LAM development team) is to write a "wrapper" `ld` script that inserts the proper flags and then calls the Solaris `ld`.

TotalView support on Solaris may require special compiler flags – in particular, for some versions of the Forte compilers, you may have to use the configure option `--with-tv-debug-flags="-g -W0,-y-s"` to work around a bug in the Forte compiler suite. ⊤ (7.0.3)

For 32-bit builds of LAM, `--with-fd-size` can be used to set a value up to 65,536. For 64-bit builds of LAM, the FD_SETSIZE is already 65,536 and can not be increased. ⊥ (7.0.3)

There is an important note about dynamic SSI modules and older versions of the GNU compilers (i.e., before 3.0) on Solaris systems in Section 4.3.6 (page 17). ⊤ (7.1) ⊥ (7.1)

4.4.8 Microsoft Windows(TM) (Cygwin)

⊤ (7.1)

LAM/MPI is supported on Microsoft Windows(TM) (Cygwin 1.5.5). Currently, tcp, sysv, usysv and lamd RPIs are supported. To ensure that sysv and usysv RPIs are built, it is necessary that the IPC module be both installed *and running* (even while configuring and building LAM/MPI). See Section 5.2.7 for more information on building and using sysv and usysv RPIs.

ROMIO is not supported. It is necessary to use the `--without-romio` option while configuring LAM/MPI for proper installation.

Since there are some issues with the use of the native Cygwin terminal for standard IO redirection, it is advised to run MPI applications on xterm. For more information about getting X services for Cygwin, please see the XFree86 web site.[8] ⊥ (7.1)

[7] `lamd.core` in the directory from which `lamboot` was run or the user's home directory

[8] `http://www.cygwin.com/`

Chapter 5

Requirements

LAM/MPI requires a minimal set of features common on most Unix-like operating systems in order to configure, compile, install, and operate. However, optional plug-in modules may require additional libraries and operating system features. The requirements for modules included with the LAM/MPI distribution are documented in Section 5.2

5.1 General LAM Requirements

In order to configure and build LAM, the following are required:

- A POSIX-like operating system (see Section 4.4 for notes on commonly used operating systems)
- A modern compiler suite:
  - An ANSI C compiler
  - A C++ compiler with STL and namespace support[1]
- Common Unix shell utilities such as `sed`, `awk`, and `grep`.
- A "modern" `make`. Unless noted in the Release Notes (Chapter 4), the `make` shipped with any modern version of the supported operating systems should work.

Once built, LAM does not require a compiler on all nodes used to run parallel applications. However, the wrapper compilers provided with LAM/MPI will only function properly if they are able to invoke the compiler originally used to build LAM.

5.2 SSI Module Requirements

LAM/MPI is built around a core component architecture known as the System Services Interface (SSI). One of its central principles is the ability to have run-time "plug-in" modules that can extend LAM's native capabilities. As such, many of LAM's intrinsic functions have been moved into SSI modules, and new modules have been added that provide capabilities that were not previously available in LAM/MPI. Component

[1] If using the GNU compiler suite, it is recommended that version 2.95 or later be used.

modules are selected from each type at run-time and used to effect the LAM run-time environment and MPI library.

There are currently four types of components used by LAM/MPI:

- boot: Starting the LAM run-time environment, used mainly with the `lamboot` command.
- coll: MPI collective communications, only used within MPI processes.
- cr: Checkpoint/restart functionality, used both within LAM commands and MPI processes.
- rpi: MPI point-to-point communications, only used within MPI processes.

The LAM/MPI distribution includes instances of each component type referred to as modules. Each module is an implementation of the component type which can be selected and used at run-time to provide services to the LAM run-time environment and MPI communications layer.

Each SSI module has its own configuration and build sub-system. The top-level LAM `configure` script will act as a coordinator and attempt to configure and build every SSI module that it finds in the LAM/MPI source tree. Any module that fails to configure properly will simply be skipped – it will not cause the entire build to fail. For example, if a system has no Myrinet hardware (and therefore no corresponding gm message passing software), the gm RPI module will simply be skipped.

SSI modules may have requirements in addition to the general LAM requirements. Failure to meet the requirements listed below does not prohibit using LAM/MPI, but does prohibit using the given SSI module. A module may require additional libraries standard to using a particular interface (for example, the bproc boot module requires `libbproc.(a|so)` in order to build properly). Such dependencies are not explicitly listed here.

5.2.1 BProc Boot Module

⊤ (7.1)

The LAM/MPI bproc boot module has only been tested with the 3.2.x series of BProc starting with v3.2.5. Prior versions of the 3.2.x series are known to have bugs that may cause LAM/MPI to fail. The bproc ⊥ (7.1) module has also been tested with BProc v4.0.

5.2.2 SLURM Boot Module

⊤ (7.1)

The LAM/MPI slurm boot module supports running in SLURM run-time environments. No additional configuration or build parameters are needed to include support for SLURM. The `laminfo` command can ⊥ (7.1) be used to verify that the slurm boot module is available in a LAM/MPI installation.

5.2.3 TM Boot Module

As noted in the Release Notes, the Altair-provided client RPMs for PBS Pro do not include the `pbs_demux` command, which is necessary for proper execution of TM jobs. The solution is to copy the executable from the server RPMs to the client nodes.

5.2.4 BLCR Checkpoint/Restart Module

The BLCR checkpoint/restart SSI module for LAM/MPI requires the Berkeley Laboratory Checkpoint/Restart package for operation. BLCR has its own limitations (e.g., BLCR does not yet support saving and restoring file descriptors); see the documentation included in BLCR for further information. Check the project's main web site[2] to find out more about BLCR. ⊤ (7.1)

If the BLCR module(s) are compiled dynamically, the `LD_PRELOAD` environment variable must include the location of the `libcr.so` library. This is to ensure that `libcr.so` is loaded before the PThreads library. ⊥ (7.1)

5.2.5 Myrinet (gm) RPI Module

⊤ (7.0.1)

LAM/MPI supports both the 1.6.x and 2.0.x series of the Myrinet native communication library.

Note that LAM/MPI does not need Myrinet hardware installed on the node where LAM is configured and built to include Myrinet support; LAM/MPI only needs the appropriate gm libraries and header files installed. Running MPI applications with the gm RPI module requires Myrinet hardware, of course. ⊥ (7.0.1)

The Myrinet native communication library (gm) can only communicate through "registered" or "pinned" memory. In most operating systems, LAM/MPI handles this automatically by pinning user-provided buffers when required. This allows for good message passing performance, especially when re-using buffers to send/receive multiple messages. However, user applications may choose to free memory at any time. Hence, LAM/MPI must intercept calls to functions such as `sbrk()` and `munmap()` in order to guarantee that pinned memory is unpinned before it is released back to the operating system.

To this end, the ptmalloc v2 memory allocation package[3] is included in LAM/MPI. Use of ptmalloc will effectively overload all memory allocation functions (e.g., `malloc()`, `calloc()`, `free()`, etc.) for all applications that are linked against the LAM/MPI libraries (potentially regardless of whether they are using the gm RPI module or not). ⊤ (7.1)

Please see Section 4.2.5 for important notes about this functionality. ⊥ (7.1)

Note that on Solaris, the gm library does not have the ability to pin arbitrary memory and auxiliary buffers must be used. Although LAM/MPI controls all pinned memory (since the user application cannot free pinned memory, the ptmalloc package is not necessary, and therefore is not used), this has a detrimental effect on performance of large messages: LAM/MPI must copy all messages from the application-provided buffer to an auxiliary buffer before it can be sent (and vice versa for receiving messages). As such, users are strongly encouraged to use the MPI_ALLOC_MEM and MPI_FREE_MEM functions instead of `malloc()` and `free()`. Using these functions will allocate "pinned" memory such that LAM/MPI will not have to use auxiliary buffers and an extra memory copy.

5.2.6 Infiniband (ib) RPI Module

⊤ (7.1)

LAM/MPI supports the Infiniband RPI using the Mellanox Verbs API (VAPI).

Note that LAM/MPI does not need Infiniband hardware installed on the node where LAM is configured and built to include Infiniband support; LAM/MPI only needs the appropriate VAPI libraries and header files installed. Running MPI applications with the ib RPI module requires Infiniband hardware, of course.

The Infiniband communication library can only communicate through "registered" or "pinned" memory. In most operating systems, LAM/MPI handles this automatically by pinning user-provided buffers when re-

[2] http://ftg.lbl.gov/
[3] http://www.malloc.de/

quired. This allows for good message passing performance, especially when re-using buffers to send/receive multiple messages. However, user applications may choose to free memory at any time. Hence, LAM/MPI must intercept calls to functions such as `sbrk()` and `munmap()` in order to guarantee that pinned memory is unpinned before it is released back to the operating system.

To this end, the ptmalloc v2 memory allocation package is included in LAM/MPI. Use of ptmalloc will effectively overload all memory allocation functions (e.g., `malloc()`, `calloc()`, `free()`, etc.) for all applications that are linked against the LAM/MPI libraries (potentially regardless of whether they are using the ib RPI module or not).

⊥ (7.1) Please see Section 4.2.5 for important notes about this functionality.

5.2.7 Shared Memory RPI Modules

The sysv and usysv shared memory RPI modules require both System V shared memory and System V semaphore support.

The sysv module allocates a semaphore set (of size 6) for each process pair communicating via shared memory. On some systems, it may be necessary to reconfigure the system to allow for more semaphore sets if running tasks with many processes communicating via shared memory.

The operating system may run out of shared memory and/or semaphores when using the shared memory RPI modules. This is typically indicated by failing to run an MPI program, or failing to run more than a certain number of MPI processes on a single node. To fix this problem, operating system settings need to be modified to increase the allowable shared semaphores/memory.

For Linux, reconfiguration can only be done by building a new kernel. First modify the appropriate constants in `include/asm-<arch>/shmparam.h`. Increasing SHMMAX will allow larger shared segments and increasing _SHM_ID_BITS allows for more shared memory identifiers.[4]

For Solaris, reconfiguration can be done by modifying `/etc/system` and then rebooting. See the Solaris `system(4)` manual page for more information. For example to set the maximum shared memory segment size to 32 MB put the following in `/etc/system`:

```
set shmsys:shminfo_shmmax=0x2000000
```

If you are using the sysv module and are running out of semaphores, the following parameters can be set:

```
set semsys:seminfo_semmap=32
set semsys:seminfo_semmni=128
set semsys:seminfo_semmns=1024
```

⊤ (7.1) For Microsoft Windows(TM) (Cygwin), the IPC services are provided by the CygIPC module. It is necessary to have this module installed to build the sysv and usysv RPIs. Specifically, these RPIs are built if and only if `libcygipc.a` is found and `ipc-daemon2.exe` is running when configuring LAM/MPI.

⊥ (7.1) Furthermore, `ipc-daemon2.exe` should be running on all the nodes if the RPI modules are to be used.

Consult your system documentation for help in determining the correct values for your system(s).

[4] This information is likely from 2.0 Linux kernels; it may or may not have changed in more recent versions.

Chapter 6

Configuring LAM/MPI

The first, and most complex, step in installing LAM/MPI is the configuration process. Many LAM/MPI options can be set at either configure or run time; the configure settings simply provide default values. Detailed instructions on setting run-time values is provided in the LAM/MPI User's Guide [10]. If you have problems configuring LAM/MPI, see the Chapter 10, "Troubleshooting."

6.1 Unpacking the Distribution

The LAM distribution is packaged as a compressed tape archive, in either `gzip` or `bzip2` format. It is available from the main LAM web site.[1]

Uncompress the archive and extract the sources.

```
shell$ gunzip -c lam-7.1.tar.gz | tar xf -
# Or, if using GNU tar:
shell$ tar zxf lam-7.1.tar.gz
```

or

```
shell$ bunzip2 -c lam-7.1.tar.bz2 | tar xf -
# Or, if using a recent version of GNU tar:
shell$ tar jxf lam-7.1.tar.bz2
```

6.2 Configuration Basics

LAM uses a GNU `configure` script to perform site and architecture specific configuration.

Change directory to the top-level LAM directory (`lam-7.1`), set any environment variables necessary, and run the `configure` script.

```
shell$ cd lam-7.1
shell$ ./configure <option> ...
```

or

[1] http://www.lam-mpi.org/

```
shell$ cd lam−7.1
shell$ sh ./configure <option> ...
```

By default, the `configure` script sets the LAM install directory to the parent of where the LAM command `lamclean` is found (if it is in your path – this will effectively build LAM/MPI to overlay a previous installation), or `/usr/local` if `lamclean` is not in your path. This behavior can be overridden with the `--prefix` option (see below).

6.3 Compiler Options

LAM's `configure` script makes an attempt to find the correct compilers to use to build LAM/MPI.

6.3.1 Specifying Compilers and Compiler Options

Environment variables are used to override the default compiler behavior. The same environment variables should be set during both configuration and build phases of the process.

Compilers The `CC`, `CXX`, and `FC` environment variables specify the C, C++, and Fortran 77 compilers to use to build LAM. The same compilers are later used by the `mpicc`, `mpiCC`, and `mpif77` wrapper compilers.

Compiler Flags The `CFLAGS`, `CXXFLAGS`, and `FFLAGS` environment variables specify the compiler flags to pass to the appropriate compiler. These flags are only used when building LAM and are not added to the wrapper compiler argument list.

Linker Flags The `LDFLAGS` and `CXXLDFLAGS` environment variables are used to pass argument to the linker (or C++ compiler when used as a linker), as appropriate. They are not added to the argument list of the wrapper compilers.

6.3.2 Mixing Vendor Compilers

A single vendor product line should typically be used to compile all of LAM/MPI. For example, if `gcc` is used to compile LAM, `g++` should be used to compile the C++ bindings, and `gcc`/`g++`/`g77` should be used to compile any user programs. Mixing multiple vendors' compilers between different components of LAM/MPI and/or to compile user MPI programs, particularly when using the C++ MPI bindings, is almost guaranteed not to work.

C++ compilers are not link-compatible; compiling the C++ bindings with one C++ compiler and compiling a user program that uses the MPI C++ bindings will almost certainly produce linker errors. Indeed, if exception support is enabled in the C++ bindings, it will only work if the C and/or Fortran code knows how to pass C++ exceptions through their code. This will only happen properly when the same compiler (or a single vendor's compiler product line, such as `gcc`, `g77`, and `g++`) is used to compile *all* components – LAM/MPI, the C++ bindings, and the user program. Using multiple vendor compilers with C++ exceptions will almost certainly not work.

The one possible exception to this rule (pardon the pun) is the `KCC` compiler. Since `KCC` turns C++ code to C code and then gives it to the back end "native" C compiler, `KCC` may work properly with the native C and Fortran compilers.

6.4 Configure Options

The `configure` script will create several configuration files used during the build phase, including the header file `share/include/lam_config.h`. Although `configure` usually guesses correctly, you may wish to inspect this file for a sanity check before building LAM/MPI.

There are many options available from the `configure` script. Although the command `./configure --help` lists many of the command line options, it will *not* list all options that are available to SSI modules. This document is the only comprehensive list of all available options.

6.4.1 General LAM Options

The following options are relevant to the general LAM/MPI infrastructure.

- `--disable-static`

  Do not build static libraries. This flag is only meaningful when `--enable-shared` is specified; if this flag is specified without `--enable-shared`, it is ignored and static libraries are created.

- `--enable-shared`

  Build shared libraries. ROMIO can not be built as a shared library, so it will always be built as a static library, even if `--enable-shared` is specified.

  Also note that enabling building shared libraries does *not* disable building the static libraries. Specifying `--enable-shared` without `--disable-static` will result in a build taking twice as long, and installing both the static and shared libraries. ⊤ (7.1)

  This flag *must* be specified if `--with-modules` is used. Specifically, modules must be built as shared libraries. ⊥ (7.1)

- `--disable-tv-queue`

  Disable building the TotalView shared library, which provides queue debugging support for LAM/MPI. This should only be necessary if your architecture and compiler combination do not support building shared libraries.

- `--prefix=PATH`

  Sets the installation location for the LAM binaries, libraries, etc., to the directory `PATH`. `PATH` must be specified as an absolute directory name.

- `--with-boot=MODULE_NAME`

  Set `MODULE_NAME` to be the default SSI boot module. `MODULE_NAME` must be one of the supported boot mechanisms. Currently, these include: rsh, tm, slurm, and bproc, and globus.

  The default value for this option is rsh. Note that LAM's configuration/build system will attempt to build all available RPI modules (regardless of what the default module is). All boot modules that are successfully built will be available for selection at run-time.

- `--with-boot-promisc`

  Sets the default behavior of the base boot SSI startup protocols. On many kinds of networks, LAM can know exactly which nodes should be making connections while booting the LAM run-time environment, and promiscuous connections (i.e., allowing any node to connect) are discouraged. However,

this is not possible in some complex network configurations and promiscuous connections *must* be enabled.

By default, LAM's base boot SSI startup protocols disable promiscuous connections. Using the `--with-boot-promisc` flag changes the default behavior to allow promiscuous connections.

Note that this setting can also be overridden at run time when lamboot is invoked. See the `lamssi_boot(7)` manual page for more details.

- `--without-dash-pthread`

  Disable use of `-pthread` with GCC compilers, using the traditional `-D_REENTRANT` and `-lpthread` flags instead. This is useful for using the GCC C compiler with non-GCC Fortran or C++ compilers. For example, when using the KCC C++ compiler on Linux, `configure` will incorrectly guess that it can pass `-pthread` to the C++ compiler. Specifying `--without-dash-pthread` and setting `CXXFLAGS` and `CXXLDFLAGS` to the appropriate values for the C++ compiler will allow KCC to properly compile threaded code.

- `--with-debug`

  This option is typically only used by the LAM Team; it should generally not be invoked by users. This option both causes additional compiler debugging and warning flags to be used while compiling LAM/MPI, and also activates some debugging-specific code within LAM/MPI (some of which may cause performance degredation). This option is disabled by default. ⊤ (7.1)

- `--disable-deprecated-executable-names`

  Several LAM command executable names are deprecated (e.g., `hcc`, `hcp`, `hf77`, `wipe`). While the executables all still exist, they are now named differently. By default, symbolic links are created to the old/deprecated names in the installation tree. This option will disable that behavior such that the deprecated names will not be created in the installation tree. ⊥ (7.1)

- `--with-exceptions`

  Used to enable exception handling support in the C++ bindings for MPI. Exception handling support (i.e., the MPI::ERRORS_THROW_EXCEPTIONS error handler) is disabled by default. See the Section 4.3.3 (page 16).

- `--with-exflags=FLAGS`
  `--without-exflags`

  Used to specify any command line arguments that are necessary for the C, C++, and Fortran compilers to enable C++ exception support. This switch is ignored unless `--with-exceptions` is also specified.

  This switch is unnecessary for `gcc`/`g77`/`g++` version 2.95 and above; `-fexceptions` will automatically be used (when building `--with-exceptions`). Additionally, this switch is unnecessary if the KCC compiler is used; `-x` is automatically used. ⊤ (7.1)

  At lease some compilers lie about who they are. The Intel C++ compiler, for example, will set the symbol `__GNUC__` to be 1, even though it is not the GNU C compiler. This makes LAM's `configure` script guess the wrong flags (because it thinks the compiler is `g++`, not `icc`). In this case, you may wish to disable all compiler exception flags. Use `--without-exflags` (as opposed to `--with-exflags=""`). ⊥ (7.1)

See the Section 4.3.3 (page 16).

- `--without-fc`

  Do not build the Fortran MPI bindings. Although `--with-fc=FC` used to be able to be used to specify the Fortran compiler, its use is deprecated (see Section 6.3 and Section 6.4.3).

- `--with-impi`

  Use this switch to enable the IMPI extensions. The IMPI extensions are still considered experimental, and are disabled by default.

  IMPI, Interoperable MPI, is a standard that allows compliant MPI implementations to interact, forming an MPI_COMM_WORLD that spans multiple implementations. Currently, there are a limited number of implementations with IMPI support. Unless necessary, it is not recommended you use the IMPI extensions.

- `--with-lamd-ack=MIRCOSECONDS`

  Number of microseconds until an ACK is resent between LAM daemons. You probably should not need to change this; the default is half a second.

- `--with-lamd-hb=SECONDS`

  Number of seconds between heartbeat messages in the LAM daemon (only applicable when running in fault tolerant mode). You probably should not need to change this; the default is 120 seconds.

- `--with-lamd-boot=SECONDS`

  Set the default number of seconds to wait before a process started on a remote node is considered to have failed (e.g., during `lamboot`). You probably should not need to change this; the default is 60 seconds. ⊤ (7.1)

- `--with-memory-manager=ptmalloc|ptmalloc2|darwin|external|none`

  The gm and ib RPI modules require an additional memory manager in order to ensure that freed memory is released from the kernel modules properly.

  If you are planning on using the gm or ib RPI modules, please read Section 4.2.5 (page 14).

  On most systems, the correct memory manager will be selected automatically. The ptmalloc package (discussed in Section 5.2.5, page 25) is used for this purpose. ptmalloc v1 ("ptmalloc") is included for historical reasons, and is currently never used unless explicitly asked for. darwin is the default on OS X platforms, none is the default on Solaris, AIX, and Cygwin systems, and ptmalloc v2 ("ptmalloc2") is the default everywhere else.

  The value "external" is a special case for a small number of cases where it is known that all applications that will use LAM/MPI have their own memory manager and will invoke LAM's internal handle release function upon invocation of `sbrk()`. See Section 4.2.5. If you have no idea what this means, you probably don't need it. ⊥ (7.1)

- `--without-mpi2cpp`

  Build LAM without the MPI-2 C++ bindings (see chapter 10 of the MPI-2 standard); the default is to build them. Unlike previous versions of LAM/MPI, a working C++ compiler is required even if the C++ bindings are disabled.

- `--with-prefix-memcpy`
  `--without-prefix-memcpy`

  The `memcpy()` function in the GNU C library (glibc) performs poorly in many cases. On glibc systems, LAM will default to using a "prefix" `memcpy()` workaround that significantly increases `memcpy()` performance (this has drastic effects on the shared memory RPI modules as well as unexpected message buffering). These two flags can be used to override the default behavior on glibc and non-glibc systems, respectively.

- `--without-profiling`

  Build LAM/MPI without the MPI profiling layer (Chapter 8 of the MPI-1 standard). The default is to build this layer since ROMIO requires it. See the `--without-romio` option for more details.

- `--with-purify`

  Causes LAM to zero out all data structures before using them. This option is not necessary to make LAM function correctly (LAM/MPI already zeros out relevant structure members when necessary), but it is very helpful when running MPI programs through memory checking debuggers, such as Purify, Solaris Workshop/Forte's `bcheck`, and Linux's `valgrind`.

  By default, LAM/MPI only initializes relevant struct members before using a structure. As such, a partially-initialized structure may be sent to a remote host. This is not a problem because the remote host will ignore the irrelevant struct members (depending on the specific function being invoked). LAM/MPI was designed this way to avoid setting variables that will not be used; this is a slight optimization in run-time performance.

  Memory-checking debuggers are both popular useful to find memory leaks, indexing past the end of arrays, and other types of memory problems. Since LAM/MPI "uses" uninitialized memory, it tends to generate many warnings with these types of debuggers. Setting the `--with-purify` option will cause LAM to always initialize the entire structure, thereby eliminating "false" warnings when running MPI applications through memory checking debuggers. Using this option, however, incurs a slight performance penalty since every structure member will be initialized before the structure is used.

- `--without-romio`

  Build LAM without ROMIO [11, 12] support (ROMIO provides the MPI-2 I/O support, see Chapter 9 of the MPI-2 standard); the default is to build *with* ROMIO support. ROMIO is known to work on a large subset of the platforms on which LAM/MPI properly operates. Consult the `romio/README` file for more information. Note that ROMIO is always built as a static library, even if `--enable-shared` and `--disable-static` are specified.

  Note also that building ROMIO implies building the profiling layer. ROMIO makes extensive use of the MPI profiling layer; if `--without-profiling` is specified, `--without-romio` must be specified.

⊤ (7.0.1)

- `--with-romio-flags=FLAGS`

  Pass `FLAGS` to ROMIO's configure script when it is invoked during the build process. This switch is to effect specific behavior in ROMIO, such as building for a non-default file system (e.g., PVFS). Note that LAM already sends the following switches to ROMIO's configure script – the `--with-romio-flags` switch should not be used to override them:

```
--prefix                -cflags
-mpi                    -fflags
-mpiincdir              -nof77
-cc                     -make
-fc                     -mpilib
-debug
```

Also note that this switch is intended for ROMIO configure flags. It is *not* intended as a general mechanism to pass prepropcessor flags, compiler flags, linker flags, or libraries to ROMIO. See `--with-romio-libs` for more details.

- `--with-romio-libs=LIBS`

  Pass `LIBS` to ROMIO's configure script and add it to the list of flags that are automatically passed by the wrapper compilers. This option is named "libs" instead of "ldflags" (and there is no corresponding `--with-romio-ldflags` option) as a simplistic short-cut; these flags are added in the traditional `LIBS` location on the wrapper compiler line (i.e., to the right of the `LDFLAGS` variable).

  This should *only* be used for flags that must be used to link MPI programs (e.g., additional "`-L`" and "`-l`" switches required to find/link libraries required by ROMIO). A typical example might be:

```
shell$ ./configure ... --with-romio-flags=-file_system=pvfs+nfs \
  --with-romio-libs=''-L/location/of/pvfs/installation -lpvfs''
```

⊥ (7.0.1)

- `--with-rpi=MODULE_NAME`

  Set `MODULE_NAME` to be the default RPI module for MPI jobs. `MODULE_NAME` should be one of the supported RPI modules. Currently, these include:

  crtcp Same as the tcp module, except that it can be checkpointed and restarted (see Section 6.4.2). A small performance overhead exists compared to tcp.

  gm Myrinet support, using the native gm message passing interface. ⊤ (7.1)

  ib Infiniband support, using the Mellanox Verbs API (VAPI) interface. ⊥ (7.1)

  lamd Native LAM message passing. Although slower than other RPI modules, the lamd RPI module provides true asynchronous message passing which can overall higher performance for MPI applications that utilize latency-hiding techniques.

  tcp Communication over fully-connected TCP sockets.

  sysv Shared memory communication for processes on the same node, TCP sockets for communication between processes on different nodes. A System V semaphore is used for synchronization between processes.

  usysv Like sysv, except spin locks with back-off are used for synchronization. When a process backs off, it attempts to yield the processor using `yield()` or `sched_yield()`.

  The default value for this option is tcp. Note that LAM's configuration/build system will attempt to build all available RPI modules (regardless of what the default module is). All RPI modules that are successfully built will be available for selection at run-time.

- `--without-shortcircuit`

  LAM's MPI message passing engines typically use queueing systems to ensure that messages are sent and received in order. In some cases, however, the queue system is unnecessary and simply adds overhead (such as when the queues are empty and a blocking send is invoked). The short-circuit optimization bypasses the queues and directly sends or receives messages when possible.

  Although not recommended, this option can be used to disable the short-circuit optimization. This option exists mainly for historical reasons and may be deprecated in future versions of LAM/MPI.

- `--with-signal=SIGNAL`

  Use SIGNAL as the signal used internally by LAM. The default value is SIGUSR2. To set the signal to SIGUSR1 for example, specify `--with-signal=SIGUSR1`. To set the signal to 50, specify `--with-signal=50`. Be *very* careful to ensure that the signal that you choose will not be used by anything else!

- `--with-threads=PACKAGE`

  Use PACKAGE for threading support. Currently supported options are pthreads, solaris, and no. no will disable threading support in LAM, with the side effects that MPI_THREAD_FUNNELED and MPI_THREAD_SERIALIZED will not be available, and checkpoint/restart functionality is disabled. Specifying `--without-threads` is equivalent to `--with-threads=no`.

- `--with-trillium`

  Build and install the Trillium support executables, header files, and man pages. Trillium support is *not* necessary for normal MPI operation; it is intended for the LAM Team and certain third party products that interact with the lower layers of LAM/MPI. Building XMPI [2], for example, requires that all the Trillium header files were previously installed. Hence, if you intend to compile XMPI after installing LAM/MPI, you should use this option.

- `--with-wrapper-extra-ldflags`

  When compiling LAM with shared library support, user MPI programs will need to be able to find the shared libraries at run time. This typically happens in one of three ways:

  1. The `LD_LIBRARY_PATH` environment variable specifies `$prefix/lib` where the LAM and MPI libraries are located.
  2. The linker is configured to search `$prefix/lib` for libraries at run-time.
  3. The path `$prefix/lib` is added to the run-time search path when the user's MPI program is linked.

  Although either option #1 or #2 is used as the typical solution to this problem, they are outside the scope of LAM. Option #3 can be effected when `--with-wrapper-extra-ldflags` is used. The appropriate compiler options to find the LAM and MPI shared libraries are added to the wrapper's underlying compiler command line to effect option #3. Note, however, that this does *not* necessarily mean that supporting shared libraries (such as those required by SSI modules) will automatically be found, particularly if they are installed in non-standard locations.

[2] http://www.lam-mpi.org/software/xmpi/

If `--with-wrapper-extra-ldflags` is *not* specified, these options are not added to the wrapper's underlying compiler command line.

6.4.2 SSI Module Options

LAM/MPI's configure/build procedure will attempt to configure and build *all* SSI modules that can be found in the source tree. Any SSI module that fails to configure properly will simply be skipped; it will not abort the entire build. Closely examining the output of LAM's `configure` script will show which modules LAM decided to build and which ones it decided to skip. Once LAM/MPI has been built and installed, the `laminfo` command can be used to see which modules are included in the installation.

The options below list several SSI module-specific configuration options. Modules that have no specific configuration options are not listed. ⊤ (7.1)

- `--with-modules[=list]`

  Enable LAM SSI modules to be built as dynamic shared objects (DSO). This option *must* be used in conjunction with `--enable-shared`; `configure` will abort with an error if this option is used without `--enable-shared`.

  If no list is specified, then all SSI modules that are built will be DSOs. If a list is specified, it can be a comma-separated list of SSI types and/or names specifying which modules to build as DSOs. Module names must be prefixed with their type name to prevent ambiguity. For example:

```
# Build all modules as DSOs
shell$ ./configure --enable-shared --with-modules
# Build all boot modules as DSOs
shell$ ./configure --enable-shared --with-modules=boot
# Build all boot and RPI modules as DSOs
shell$ ./configure --enable-shared --with-modules=boot,rpi
# Build all boot modules and the gm RPI module as DSOs
shell$ ./configure --enable-shared --with-modules=boot,rpi:gm
# Build the rsh boot module and gm RPI module as DSOs
shell$ ./configure --enable-shared --with-modules=boot:rsh,rpi:gm
```

⊥ (7.1)

BProc boot Module

⊤ (7.1)

- `--with-boot-bproc=PATH`

  If the BProc header and library files are installed in a non-standard location, this option must be used to indicate where they can be found. If the bproc module's configuration fails to find the BProc header and library files, it will fail (and therefore be skipped).

⊥ (7.1)

RSH boot Module

- `--with-rsh=AGENT`

Use AGENT as the remote shell command. For example, to use ssh then specify --with-rsh="ssh -x".[3] This shell command will be used to launch commands on remote nodes from binaries such as lamboot, recon, lamwipe, etc. The command can be one or more shell words, such as a command and multiple command line switches. This value is adjustable at run-time.

TM (OpenPBS/PBS Pro/Torque) boot Module

⊤ (7.1)

- --with-boot-tm=PATH

  If the TM (PBS) header and library files are installed in a non-standard location, this option must be used to indicate where they can be found. If the tm module's configuration fails to find the PBS header and library files, it will fail (and therefore be skipped).

⊥ (7.1)

BLCR Checkpoint/Restart (cr) Module

The blcr checkpoint/restart module may only be used when the running RPI is crtcp. You may want to use
⊤ (7.1) the configure option --with-rpi=crtcp to make crtcp the default RPI.

- --with-cr-blcr=PATH

  If the BLCR header and library files are installed in a non-standard location, this option must be used to indicate where they can be found. If the blcr module's configuration fails to find the BLCR header and library files, it will fail (and therefore be skipped).

- --with-cr-base-file-dir=PATH

  Use PATH as the default location for writing checkpoint files. This parameter is adjustable at run-time.

⊥ (7.1)

CRTCP rpi Module

⊤ (7.1)

- --with-rpi-crtcp-short=BYTES

  Use BYTES as the maximum size of a short message when communicating over TCP using the checkpoint/restart-enabled TCP module. The default is 64 KB. See Section 9.1 (page 49) for information on communication protocols. This parameter is adjustable at run-time.

⊥ (7.1)

Myrinet/GM rpi Module

Note that LAM/MPI does not need Myrinet hardware installed on the node where LAM is configured and built to include Myrinet support; LAM/MPI only needs the appropriate gm libraries and header files in-
⊤ (7.1) stalled. Running MPI applications with the gm RPI module requires Myrinet hardware, of course.

- --with-rpi-gm=PATH

  If the GM header and library files are installed in a non-standard location, this option must be used to indicate where they can be found. If the gm module's configuration fails to find the GM header and library files, it will fail (and therefore be skipped).

[3] Note that -x is necessary to prevent the ssh 1.x series of clients from sending its standard banner information to the standard error, which will cause recon/lamboot/etc. to fail.

- `--with-rpi-gm-get`

  Enable the experimental use of the GM 2.x library call `gm_get()`. Although this can slightly reduce latency for long messages, its use is not yet advised. As of this writing, Myricom still does not recommend the use of the GM 2.x series in production clusters.

  More specifically, by the time LAM/MPI 7.1 shipped, there still appeared to be a problem with using `gm_get()` – it wasn't clear if it was LAM's problem or GM's problem. If it turns out to be a GM problem, and GM gets fixed, this switch is provided to enable the use of `gm_get()` in the gm module.

  Note that checkpointing the gm module is only supported when using this option. Specifically, LAM can only checkpoint gm jobs that use `gm_get()`. ⊥ (7.1) ⊤ (7.0.5)

- `--with-rpi-gm-lib=PATH`

  By default, LAM looks for the GM library in `$GMDIR/lib` and `$GMDIR/binary/lib`. If your system's GM library is in neither of these locations, this option can be used to specify the directory where the GM library can be found. ⊥ (7.0.5) ⊤ (7.1)

- `--with-rpi-gm-tiny=MAX_BYTES`

  Set the boundary size between the tiny and long protocols. See Section 9.1 (page 49) for information on communication protocols. This parameter is adjustable at run-time.

- `--with-rpi-gm-max-port=N`

  Use N as the maximum GM port to try when searching for an open port number. The default value is 8, and is also modifiable at run-time.

- `--with-rpi-gm-port=N`

  Use N as a fixed port to use for GM communication. The default is -1, meaning "search available ports for an open port". This value is adjustable at run-time. ⊥ (7.1)

Infiniband rpi Module ⊤ (7.1)

Note that LAM/MPI does not need Infiniband hardware installed on the node where LAM is configured and built to include Infiniband support; LAM/MPI only needs the appropriate VAPI libraries and header files installed. Running MPI applications with the ib RPI module requires Infiniband hardware, of course.

- `--with-rpi-ib=PATH`

  If the IB/VAPI header and library files are installed in a non-standard location, this option must be used to indicate where they can be found. If the ib module's configuration fails to find the IB/VAPI header and library files, it will fail (and therefore be skipped).

- `--with-rpi-ib-tiny=bytes`

  This is the tiny message cut-off for message passing using IB. The default is 1024 bytes, but can be overridden with this option. This option is adjustable at runtime.

- `--with-rpi-ib-port=N`

  Use port N as the fixed port for IB communication. The default is -1, meaning "search available ports for an open port". This value is adjustable at run-time.

- `--with-rpi-ib-num-envelopes=N`

  Pre-post N envelopes per peer process for the tiny messages. In the case of Infiniband, the receive buffers need to be preposted on the receiver side of the message. These preposts are on a per peer process basis and each post will be of the sum of size of message headers internally generated by LAM/MPI and tiny message size (specified by the configure time `--with-rpi-ib-tiny=N` option or the runtime `rpi_ib_tinymsglen` parameter or the default size of 1024 bytes).

⊥ (7.1)

Shared Memory rpi Modules

The usysv and sysv modules differ only in the mechanism used to synchronize the transfer of messages via shared memory. The usysv module uses spin locks with back-off while the sysv module uses System V semaphores.

Both transports use a few System V semaphores for synchronizing the deallocation of shared structures or for synchronizing access to the shared pool. Both modules also use TCP for off-node communication, so options for the tcp RPI module are also applicable.

⊤ (7.1) See Section 9.2 (page 49) for information on shared memory tuning parameters.

- `--with-rpi-sysv-maxalloc=BYTES`
  `--with-rpi-usysv-maxalloc=BYTES`

  Use `BYTES` as the maximum size of a single allocation from the shared memory pool. If no value is specified, configure will set the size according to the value of `--shm-poolsize` (below). This parameter is adjustable at run-time.

- `--with-rpi-sysv-poolsize=BYTES`
  `--with-rpi-usysv-poolsize=BYTES`

  Use `BYTES` as the size of the shared memory pool. If no size is specified, `configure` will determine a suitably large size to use. This parameter is adjustable at run-time.

- `--with-rpi-sysv-short=BYTES`
  `--with-rpi-usysv-short=BYTES`

  Use `BYTES` as the maximum size of a short message when communicating via shared memory. The default is 8 KB. This parameter is adjustable at run-time.

- `--with-rpi-sysv-pthread-lock`
  `--with-rpi-usysv-pthread-lock`

  Use a process-shared pthread mutex to lock access to the shared memory pool rather than the default System V semaphore. This option is only valid on systems which support process-shared pthread mutexes.

⊥ (7.1)

- `--with-select-yield`

  The usysv transport uses spin locks with back-off. When a process backs off, it attempts to yield the processor. If the `configure` script finds a system-provided yield function such as `yield()` or `sched_yield()`, this is used. If no such function is found, then `select()` on NULL file descriptor sets with a timeout of 10us is used.

  The `--with-select-yield` option forces the use of `select()` to yield the processor. This option only has meaning for the usysv RPI.

TCP rpi Module

⊤ (7.1)

- `--with-rpi-tcp-short=BYTES`

  Use `BYTES` as the maximum size of a short message when communicating over TCP. The default is 64 KB. This is relevant to the `sysv` and `usysv` RPI modules, since the shared memory RPIs are multi-protocol – they will use TCP when communicating with MPI ranks that are not in the same node. See Section 9.1 (page 49) for information on communication protocols. This parameter is adjustable at run-time.

⊥ (7.1)

6.4.3 Deprecated Flags

The flags listed below are deprecated. While they are still usable in LAM/MPI 7.1, it is possible they will disappear in future versions of LAM/MPI. ⊤ (7.1)

- `--with-blcr=PATH`

  Use the `--with-cr-blcr=PATH` flag instead.

- `--with-bproc=PATH`

  Use the `--with-boot-bproc=PATH` flag instead. ⊥ (7.1)

- `--with-cc=CC`

  Use the `CC` environment variable instead.

- `--with-cflags=CFLAGS`

  Use the `CFLAGS` environment variable instead. ⊤ (7.1)

- `--with-cr-file-dir=PATH`

  Use the `--with-cr-base-file-dir=PATH` flag instead. ⊥ (7.1)

- `--with-cxx=CXX`

  Use the `CXX` environment variable instead.

- `--with-cxxflags=CXXFLAGS`

  Use the `CXXFLAGS` environment variable instead.

- `--with-cxxldflags=CXXLDFLAGS`

  Use the `CXXLDFLAGS` environment variable instead.

- `--with-fc=FC`

  Use the `FC` environment variable instead.

- `--with-fflags=FFLAGS`

  Use the `FFLAGS` environment variable instead. ⊤ (7.1)

- `--with-gm=PATH`

  Use the `--with-rpi-gm=PATH` flag instead.

- `--with-pthread-lock`

  Use the `--with-rpi-sysv-pthread-lock` and `--with-rpi-usysv-pthread-lock` flags instead.

- `--with-shm-maxalloc=BYTES`

  Use the `--with-rpi-sysv-maxalloc=BYTES` and `--with-rpi-usysv-maxalloc=BYTES` flags instead.

- `--with-shm-poolsize=BYTES`

  Use the `--with-rpi-sysv-poolsize=BYTES` and `--with-rpi-usysv-poolsize=BYTES` flags instead.

- `--with-shm-short=BYTES`

  Use the `--with-rpi-sysv-short=BYTES` and `--with-rpi-usysv-short=BYTES` flags instead.

- `--with-tcp-short=BYTES`

  Use the `--with-rpi-tcp-short=BYTES` flag instead.

- `--with-tm=PATH`

⊥ (7.1) Use the `--with-boot-tm=PATH` flag instead.

Chapter 7

Building LAM/MPI

This chapter provides instructions on building both LAM/MPI and the examples that are part of the LAM distribution. In general, this step is both the easiest and longest phase of the installation process. This step is best accompanied by a trip to the coffee maker – even on a modern machine it can take well over 15 minutes.

7.1 Building LAM

Once the configuration step has completed, build LAM by doing:

```
shell$ make
```

in the top level LAM directory. This will build the LAM binaries and libraries within the distribution source tree. Once they have compiled properly, you can install them with:

```
shell$ make install
```

`make` is capable of "chaining" commands, performing the build and install in one command. The following example chains the two above commands ("`make all`" and "`make install`") into one, but if there is a failure during the build phase, `make` will not start the install phase.

```
shell$ make all install
```

NOTE : Previous version of LAM included `make install` in the default `make`. *This is no longer true.* You *must* execute `make install` to install the LAM executables, libraries, and header files to the location specified by the `--prefix` option to configure.

7.2 Advanced Building: Alternate Install Directory

The LAM build system is capable of installing into a directory other than the one specified by the `--prefix` configure option. LAM will not operate properly when installed in the alternate directory. However, the feature is useful for building binary packages (such as RPMs or Mac OS X Installer packages) or staging for read-only filesystems.

The `DESTDIR` flag to `make` allows LAM to be installed in a "prefixed" location. If LAM is configured to install in `/usr/local` and the `DESTDIR=/tmp/lam` flag is used to `make`, LAM will be installed in `/tmp/lam/usr/local`:

```
shell$ ./configure --prefix=/usr/local
# ...lots of output...
shell$ make all
# ...lots of output...
shell$ make DESTDIR=/tmp/lam install
# ...lots of output - LAM installed in /tmp/lam/usr/local ...
shell$ export LAMHOME=/tmp/lam/usr/local
# allow testing of LAM while installed in alternate directory
shell$ lamboot boot_schema
# ... usual LAM testing ...
```

LAM will not operate correctly when installed in an alternate directory to the one specified with the `--prefix` configure option. LAM relies on the installation prefix information to search for various files. It is possible to "fix" LAM's search path by using the `LAMHOME` environment variable. It is not recommended that `LAMHOME` not be used beyond initial testing (See Chapter 8, page 43 for testing information).

7.3 Building the Included Examples

⊤ (7.1) LAM includes two example packages: general LAM examples and ROMIO examples. Both packages can
⊥ (7.1) be build from a single top-level "`make lamexamples`". Note that the examples can *only* be built after a
⊤ (7.1) successful `make install`, and `$prefix/bin` has been placed in your `$PATH`.

```
shell$ make lamexamples
```

⊥ (7.1)

This will do the following (where `TOPDIR` is the top-level directory of the LAM source tree):

1. Build the LAM examples. They are located in:

   `TOPDIR/examples`

   The `examples` directory includes C, C++, and Fortran examples. The C++ and Fortan examples if support for the language is built in LAM.

2. If you configured LAM with ROMIO support (i.e., if you did not configure with `--without-romio`), the ROMIO examples will be built. See the notes about ROMIO in the RELEASE_NOTES file. They are located in:

   `TOPDIR/romio/test`

Additionally, the following commands can be used to build each of the packages' examples separately
⊤ (7.1) (provided that support for each was compiled in to LAM) from `TOPDIR`:

```
shell$ (cd examples; make examples)
shell$ make romio-examples
```

⊥ (7.1)

Chapter 8

After the Build

After the build process, LAM/MPI should be built and installed. Building the examples shows that the wrapper compilers provided by LAM/MPI work properly. However, there is still much of LAM that should be tested before it is put into production use. This chapter details the basic testing we recommend be performed on each installation.

8.1 Sanity Check

With LAM/MPI 7.1, it is possible to reconfigure many portions of LAM that were previously relatively static. Therefore, the `laminfo` command has been added to allow users to determine how a particular version of LAM is configured. System administrators may wish to use the `laminfo` as a sanity check when installing LAM. An example of the output from `laminfo` is shown below.

```
shell$ laminfo
              LAM/MPI: 7.1
               Prefix: /usr/local
         Architecture: powerpc-apple-darwin7.4.0
        Configured by: jsquyres
        Configured on: Fri Jul  9 08:58:12 EST 2004
       Configure host: macosx.example.org
           C bindings: yes
         C++ bindings: no
     Fortran bindings: no
           C compiler: gcc
         C++ compiler: g++
     Fortran compiler: false
      Fortran symbols: none
          C profiling: no
        C++ profiling: no
    Fortran profiling: no
       C++ exceptions: no
       Thread support: yes
        ROMIO support: no
         IMPI support: no
```

```
       Debug support: yes
        Purify clean: yes
            SSI boot: globus (API v1.1, Module v0.5)
            SSI boot: rsh (API v1.1, Module v1.0)
            SSI boot: slurm (API v1.1, Module v1.0)
            SSI coll: lam_basic (API v1.1, Module v7.1)
            SSI coll: smp (API v1.1, Module v1.1)
             SSI rpi: crtcp (API v1.1, Module v1.0.1)
             SSI rpi: lamd (API v1.0, Module v7.0)
             SSI rpi: tcp (API v1.0, Module v7.0)
             SSI rpi: sysv (API v1.0, Module v7.0)
             SSI rpi: usysv (API v1.0, Module v7.0)
```

The `laminfo` command includes the option `-parsable`, which will format the output of `laminfo` in a script friendly way. This has proven useful when automating sanity checks after a build.

8.2 Separating LAM and MPI TCP Traffic

⊤ (7.1)

LAM uses two distinct types of communication: its out-of-band communications (e.g., process control, file transfer, and I/O redirection) and MPI "in-band" communication. When using the tcp rpi module, it is possible to separate these two forms of communications onto separate networks.

Such functionality can be useful in environments where multiple TCP networks exist. For example, a common cluster configuration is to have two networks:

1. A Fast Ethernet network for commodity file transfers, remote login, and other interactive services.
2. A Gigabit Ethernet network exclusively reserved for high-speed data transfers (e.g., MPI communications).

In such environments, it is desirable to use the Fast Ethernet for LAM's out-of-band traffic, and the Gigabit Ethernet network for MPI traffic. By default, LAM uses one network for all of its traffic. LAM's out-of-band communication is actually fairly sparse and typically does not have a noticable impact on MPI communications. However, a subtle issue arises when using LAM's boot SSI modules in various batch environments such as PBS (Torque) and SLURM.

These batch schedulers are typically configured to operate on the "slow" network in order to keep their traffic from interfering with the high-speed traffic on the "fast" network. The pbs and slurm modules automatically obtain their boot schemas from their respective batch systems, which, in these cases, will result in a list of "slow" network interfaces. Although the automatic determination of the boot schema is a nice feature, having LAM's MPI traffic flow across the "slow" network is clearly not desirable.

LAM provides a feature to map from one TCP network to another. Specifically, LAM can map from the set of IP names/addresses specified in the boot schema to a second set of IP names/addresses that will be used for MPI communication. Moreover, a default mapping can be provided by the system administrator so that users are unaware of this functionality.

The file `$sysconf/lam-hostmap.txt` (which is usually `$prefix/etc/lam-hostmap.txt`) is the location of the default map. It is a simple text file that lists origin host names/addresses, one per line, and their correspoding remapped host name/address for use with MPI communications. For example:

```
node1.slow.example.com mpi=node1.fast.example.come
node2.slow.example.com mpi=node2.fast.example.come
```

The remapped host names/addresses are listed in the "`mpi`" key.

Unless otherwise specified, this `lam-hostmap.txt` file is used at run-time. An alternate hostmap file can also be specified on the `mpirun` command line using the `mpi_hostmap` SSI parameter:

```
shell$ mpirun C -ssi mpi_hostmap my_hostmap.txt my_mpi_application
```

This tells LAM to use the hostmap `my_hostmap.txt` instead of `$sysconf/lam-hostmap.txt`. The special filename "`none`" can also be used to indicate that no address remapping should be performed. ⊥ (7.1)

8.3 The LAM Test Suite

The LAM test suite provides basic coverage of the LAM/MPI installation. If the test suite succeeds, users can be confident LAM has been configured and installed properly. The test suite is best run with 4 or more nodes. However, it can easily be run on two machines.

The details of booting the LAM run-time environment are found in the LAM/MPI User's Guide. If you have never used LAM before, you may wish to look at the quick overview chapter before running the LAM test suite. `boot_schema`, below, is the name of a file containing a list of hosts on which to run the test suite.

WARNING: It is *strongly* discouraged to run the test suite on the lamd RPI over a large number of nodes. The lamd RPI uses UDP for communication and does not scale well. Running on a large number of nodes can cause "packet storm" issues, and generally degrade network performance for the duration of the test. The default configuration of the lamtests suite is to run on all available nodes using all available RPI modules. To run on a subset of available RPI modules, specify the `MODES` parameter with a list of RPI modules to use (not all versions of `make` allow this). See the example below.

```
shell$ gunzip -c lamtests-7.1.tar.gz | tar xf -
shell$ cd lamtests-7.1
shell$ lamboot boot_schema
# ...lots of output...

shell$ ./configure
# ...lots of output...

shell$ make -k check MODES=''sysv usysv gm tcp''
# ...lots of output...

shell$ lamhalt
```

Common errors in the test suite include:

- Some Fortran compilers will confuse the GNU Autoconf fortran libraries test in the test suite's `configure` script (e.g., some versions of the Intel Fortran compiler, and some versions of `g77` on OS X), causing the entire process to fail. If the "checking for dummy main to link with Fortran 77 libraries" test fails, set the `FLIBS` environment variable to the output of the "checking for Fortran

77 libraries" test, but without the syntax errors that it incorrectly contains. For example (artificially word-wrapped to fit on the page):

```
checking for Fortran 77 libraries... −L/u/jsquyres/local/lib
−llamf77mpi −lmpi −llam −lutil −lpthread'' −L\ −lpthread
−L/usr/local/intel/compiler70/ia32/lib −L/usr/lib −lintrins −lIEPCF90
−lF90 −limf −lm −lirc −lcxa −lunwind
checking for dummy main to link with Fortran 77 libraries... unknown
configure: error: linking to Fortran libraries from C fails
See 'config.log' for more details.
```

Note the extra quotes and empty "`-L`" in the output. Remove these syntax errors when setting the `FLIBS` variable:

```
shell$ FLIBS=''−L/u/jsquyres/local/lib −llamf77mpi −lmpi −llam −lutil \
−lpthread −L/usr/local/intel/compiler70/ia32/lib −L/usr/lib −lintrins \
−lIEPCF90 −lF90 −limf −lm −lirc −lcxa −lunwind''
shell$ export FLIBS
shell$ ./configure ...
```

⊤ (7.1)

In other cases, the test simply gets the wrong answer. With `g77` on OS X, for example, the test may include "`-lcrt2.o`" in the argument list, which may cause linking problems such as duplicate symbols (which can be seen by looking in the `config.log` file). In this case, setting `FLIBS` environment variable without the "`-lcrt2.o`" argument may fix the problem.

⊥ (7.1)

After setting `FLIBS` to this value, run `configure` again; the bad test will be skipped and `configure` will finish successfully.

- If the test suite was not on a filesystem that is available on all nodes used in the testing, you may see a small number of errors during testing of the dynamic process features. These errors are normal and can safely be ignored.
- BProc systems will see errors in the `procname` test, complaining that the name returned by MPI was not what was expected. This is normal and can safely be ignored.
- If using the `gm` RPI on a Solaris system,[1] there may be limited amounts of DMA-able memory that LAM can access. Some tests require $O(N)$ (or even $O(N^2)$) memory (N is the number of processes in the test), and can exhaust the available DMA-able memory. This is not a problem with LAM, but rather a problem with the DMA memory system. Either tweak your operating system's settings to allow more DMA-able memory, or set the environment variable `LAM_MPI_test_small_mem` (to any value) before running the tests. Setting this variable will cause some of the LAM tests to artificially decrease their message sizes such that most tests (if not all) should complete successfully.
- Some tests require a fixed number of MPI processes (e.g., 6 or 8 processes for `cart`, `graph`, `range`, `sub`) rather than using the default LAM universe with "C". This may cause problems with some RPI modules (such as `sysv`) that try to allocate resources on a per-process basis if the total number of

[1] Solaris systems in particular are suseciptable to this problem because the native Myrinet message passing library (gm) is unable to "pin" arbitrary memory under Solaris, and instead must rely on special DMA-malloc functions.

processes causes oversubscription on the testing nodes (e.g., running all 8 processes on a single node). These errors are safe to ignore, or can be fixed by running on a larger number of nodes.

If there are any other errors, please see the troubleshooting section (Section 10, page 51).

Chapter 9

Advanced Resources

This section is intended to provide background information on some of the more advanced features of LAM/MPI necessary to obtain the best performance for each network. More detailed information is available in the LAM/MPI User's Guide, but this section should provide enough detail to set reasonable configure-time defaults.

9.1 Short / Long Protocols

LAM MPI may use one of two different protocols for its underlying message passing, depending on the module selected and the size of the message being transferred.

tcp (and therefore sysv, and usysv), gm, and ib use a short/long message protocol. If a message is "short," it is sent together with a header in one transfer to the destination process. If the message is "long," then a header (possibly with some data) is sent to the destination. The sending process then waits for an acknowledgment from the receiver before sending the rest of the message data. The receiving process sends the acknowledgment when a matching receive is posted.

For tcp, the crossover point should be at most the maximum buffer size for a TCP socket. The default value, 64 KB, is supported on every platform that LAM runs on. On some platforms, it may be possible to allocate a larger buffer, and doing so may result in improved performance. On gigabit Ethernet systems, for example, a larger buffer will almost always improve performance. Using a utility such as NetPIPE [4] can assist in finding good values for a particular platform.

The gm and ib modules both call the short protocol "tiny," but the general idea is the same.

The crossover message size between these protocols is configurable at both build and run-time for each module. See Section 6.4.2 (page 35) to change the crossover sizes.

9.2 Shared Memory RPI Modules

When using the sysv and usysv RPI modules, processes located on the same node communicate via shared memory. The module allocates one System V shared segment per node shared by all processes of an MPI application that are on the same node. This segment is logically divided into three areas. The total size of the shared segment (in bytes) allocated on each node is:

$$(2 \times C) + (N \times (N - 1) \times (S + C)) + P$$

where C is the cache line size, N is the number of processes on the node, S is the maximum size of short messages, and P is the size of the pool for large messages.

The makeup and usage of this shared memory pool is discussed in detail in the User's Guide. Skipping all the details, it may be necessary to increase the number of System V semaphores and/or shared memory on a machine, and/or change the shared memory parameters available via LAM's `configure` script. See Section 6.4.2 for more details.

Chapter 10

Troubleshooting

In general, LAM builds correctly on a wide variety of platforms with no problems. Due to the wide variety of platforms using LAM, problems do occur from time to time. The primary resource for assistance is the LAM web page and the LAM mailing list. If LAM built and installed without error, please see the LAM/MPI User's Guide for additional troubleshooting information.

10.1 What To Do When Something Goes Wrong

It is highly recommended that you execute the following steps *in order*. Many people have similar problems with configuration and initial setup of LAM, and most common problems have already been answered in one way or another.

1. Check the LAM FAQ:

   `http://www.lam-mpi.org/faq/`

2. Check the mailing list archives. Use the "search" features to check old posts and see if others have asked the same question and had it answered:

   `http://www.lam-mpi.org/MailArchives/lam/`

3. If you do not find a solution to your problem in the above resources, and your problem specifically has to do with *building* LAM, send the following information to the LAM mailing list (See Section 10.2 – please compress the files before sending them!):

   - All output (both compilation output and run time output, including all error messages)
   - Output from when you ran `./configure` to configure LAM (**please compress!**)
   - The `config.log` file from the top-level LAM directory (**please compress!**)
   - Output from when you ran `make` to build LAM (**please compress!**)
   - Output from when you ran `make install` to install LAM (**please compress!**)

   To capture the output of the configure and make steps you can use the script command or the following technique if using a csh style shell:

```
shell% ./configure {options} |& tee config.LOG
shell% make all |& tee make.LOG
shell% make install |& tee make-install.LOG
```

or if using a Bourne style shell:

```
shell$ ./configure {options} 2>&1 | tee config.LOG
shell$ make all 2>&1 | tee make.LOG
shell$ make install 2>&1 | tee make-install.LOG
```

To compress all the files listed above, we recommend using the `tar` and `gzip` commands. For example (using a `csh`-style shell):

```
shell% mkdir $HOME/lam-output
shell% cd /directory/for/lam-7.0
shell% ./configure |& tee $HOME/lam-output/configure.LOG
# ...lots of output...
shell% cp config.log share/include/lam_config.h $HOME/lam-output
shell% make all |& tee $HOME/lam-output/make.LOG
# ...lots of output...
shell% make install |& tee $HOME/lam-output/make-install.LOG
# ...lots of output...
shell% cd $HOME
shell% tar cvf lam-output.tar lam-output
# ...lots of output...
shell% gzip lam-output.tar
```

Then send the resulting `lam-output.tar.gz` file to the LAM mailing list.

If you are using an `sh`-style shell:

```
shell$ mkdir $HOME/lam-output
shell$ cd /directory/for/lam-7.0
shell$ ./configure 2>&1 | tee $HOME/lam-output/configure.LOG
# ...lots of output...
shell$ cp config.log share/include/lam_config.h $HOME/lam-output
shell$ make all 2>&1 | tee $HOME/lam-output/make.LOG
# ...lots of output...
shell$ make install 2>&1 | tee $HOME/lam-output/make-install.LOG
# ...lots of output...
shell$ cd $HOME
shell$ tar cvf lam-output.tar lam-output
# ...lots of output...
shell$ gzip lam-output.tar
```

Then send the resulting `lam-output.tar.gz` file to the LAM mailing list.

10.2 The LAM/MPI Mailing Lists

There are two mailing lists: one for LAM/MPI announcements, and another for questions and user discussion of LAM/MPI.

10.2.1 Announcements

This is a low-volume list that is used to announce new version of LAM/MPI, important patches, etc. To subscribe to the LAM announcement list, visit its list information page (you can also use that page to unsubscribe or change your subscription options):

`http://www.lam-mpi.org/mailman/listinfo.cgi/lam-announce`

NOTE: Users cannot post to this list; all such posts are automatically rejected – only the LAM Team can post to this list.

10.2.2 General Discussion / User Questions

BEFORE YOU POST TO THIS LIST: *Please* check all the other resources listed in this chapter first. Search the mailing list to see if anyone else had a similar problem before you did. Re-read the error message that LAM displayed to you (LAM can sometimes give *incredibly* detailed error messages that tell you *exactly* how to fix the problem). This, unfortunately, does not stop some users from cut-n-pasting the entire error message, verbatim (including the solution to their problem) into a mail message, sending it to the list, and asking "How do I fix this problem?" So please: think (and read) before you post.[1]

This list is used for general questions and discussion of LAM/MPI. User can post questions, comments, etc. to this list. **Due to recent increases in spam, only subscribers are allowed to post to the list**. If you are not subscribed to the list, your posts will be discarded.

To subscribe or unsubscribe from the list, visit the list information page:

`http://www.lam-mpi.org/mailman/listinfo.cgi/lam`

After you have subscribed (and received a confirmation e-mail), you can send mail to the list at the following address:

You must be subscribed in order to post to the list
`lam@lam-mpi.org`
You must be subscribed in order to post to the list

Be sure to include the following information in your e-mail:

- The `config.log` file from the top-level LAM directory, if available (**please compress!**).
- The output of "`laminfo -all`".

[1] Our deep appologies if some of the information in this section appears to be repetitive and condescending. Believe us when we say that we have tried all other approaches – some users simply either do not read the information provided, or only read the e-mail address to send "help!" e-mails to. It is our hope that big, bold print will catch some people's eyes and enable them to help themselves rather than having to wait for their post to distribute around the world and then further wait for someone to reply telling them that the solution to their problem was already printed on their screen. Thanks for your time in reading all of this!

- A *detailed* description of what is failing. The more details that you provide, the better. E-mails saying "My application doesn't work!" will inevitably be answered with requests for more information about *exactly what doesn't work*; so please include as much detailed information in your initial e-mail as possible.

NOTE: People tend to only reply to the list; if you subscribe, post, and then unsubscribe from the list, you will likely miss replies.

Also please be aware that `lam@lam-mpi.org` is a list that goes to several hundred people around the world – it is not uncommon to move a high-volume exchange off the list, and only post the final resolution of the problem/bug fix to the list. This prevents exchanges like "Did you try X?", "Yes, I tried X, and it did not work.", "Did you try Y?", etc. from cluttering up peoples' inboxes.

Bibliography

[1] Al Geist, William Gropp, Steve Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, William Saphir, Tony Skjellum, and Marc Snir. MPI-2: Extending the Message-Passing Interface. In Luc Bouge, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *Euro-Par '96 Parallel Processing*, number 1123 in Lecture Notes in Computer Science, pages 128–135. Springer Verlag, 1996.

[2] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.

[3] Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine. Checkpoint-restart support system services interface (SSI) modules for LAM/MPI. Technical Report TR578, Indiana University, Computer Science Department, 2003.

[4] Q. Snell, A. Mikler, and J. Gustafson. NetPIPE: A Network Protocol Independent Performace Evaluator, 1996.

[5] Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine. Boot system services interface (SSI) modules for LAM/MPI. Technical Report TR576, Indiana University, Computer Science Department, 2003.

[6] Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine. MPI collective operations system services interface (SSI) modules for LAM/MPI. Technical Report TR577, Indiana University, Computer Science Department, 2003.

[7] Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine. Request progression interface (RPI) system services interface (SSI) modules for LAM/MPI. Technical Report TR579, Indiana University, Computer Science Department, 2003.

[8] Jeffrey M. Squyres, Brian Barrett, and Andrew Lumsdaine. The system services interface (SSI) to LAM/MPI. Technical Report TR575, Indiana University, Computer Science Department, 2003.

[9] The LAM/MPI Team. *LAM/MPI Installation Guide*. Open Systems Laborator, Pervasive Technology Labs, Indiana University, Bloomington, IN, 7.0 edition, May 2003.

[10] The LAM/MPI Team. *LAM/MPI User's Guide*. Open Systems Laborator, Pervasive Technology Labs, Indiana University, Bloomington, IN, 7.0 edition, May 2003.

[11] Rajeev Thakur, William Gropp, and Ewing Lusk. Data sieving and collective I/O in ROMIO. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189. IEEE Computer Society Press, February 1999.

[12] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, pages 23–32. ACM Press, May 1999.

Index