ANL/MCS-TM-ANL-01/x Rev x

Installation and User's Guide to MPICH, a Portable Implementation of MPI Version 1.2.6 The ch_shmem device for Shared Memory Processors

by

William Gropp and Ewing Lusk

ARGONNE NATIONAL LABORATORY • UNIVERSITY OF CHICAGO •

MATHEMATICS AND
COMPUTER SCIENCE
DIVISION

Contents

Abstract

MPI (Message-Passing Interface) is a standard specification for message-passing libraries. MPICH is a portable implementation of the full MPI-1 specification for a wide variety of parallel and distributed computing environments. MPICH contains, along with the MPI library itself, a programming environment for working with MPI programs. The programming environment includes a portable startup mechanism, several profiling libraries for studying the performance of MPI programs, and an X interface to all of the tools. This document describes how to install and use the MPICH implementation of MPI.

This document describes how to obtain, install, and use MPICH [10], the portable implementation of the MPI Message-Passing Standard. This document describes version 1.2.6.

1 Introduction

MPICH is a freely available implementation of the MPI standard that runs on a wide variety of systems. The details of the MPICH implementation are described in [10]; related papers include [7] and [8].

MPICH is no longer being developed; the current release may be the last release of MPICH. A new version of MPI, MPICH2, is available from `www.mcs.anl.gov/mpi/mpich2`. This all-new implementation of MPI includes all of both MPI-1 and MPI-2

Major Features of MPICH:

- Full MPI 1.2 compliance, *including* cancel of sends.
- MPMD programs (see Section 3.4) and heterogeneous clusters are supported.
- The MPI-2 standard C++ bindings are available for the MPI-1 functions.
- Both Fortran 77 and Fortran 90 bindings, including both '`mpif.h`' and an MPI module.
- A Windows NT version is available as open source. The installation and use for this version is different; this manual covers only the Unix version of MPICH.
- Supports a wide variety of environments, including clusters of SMPs and massively parallel computers.
- Follows many (but not yet all) of GNU-recommended build and install targets, including VPATH.
- Parts of MPI-2 are also supported:
  - Most of MPI-IO is supported through the ROMIO implementation (See '`romio/README`' for details).
  - Support for `MPI_INIT_THREAD` (but only for `MPI_THREAD_SINGLE` and `MPI_THREAD_FUNNELLED`).
  - Miscellaneous new `MPI_Info` and `MPI_Datatype` routines.

- `MPICH` also includes components of a parallel programming environment, including
  - Tracing and logfile tools based on the MPI profiling interface, including a scalable logfile format (SLOG).
  - Parallel performance visualization tools (`upshot` and `jumpshot`).
  - Extensive correctness and performance tests.
  - Both large and small application examples.

1.1 Features of recent releases

Version 1.2.6 of `MPICH` is primarily a bug fix and increased portability release, particularly for Linux-based clusters.

New and improved in 1.2.6:

- Improved build for Mac OSX, including use of the IBM compilers
- Many small fixes
- Additional improvements to the collective algorithms

New and improved in 1.2.5:

- Updated Globus2 device for Globus 2.2 or later
- Improved SLOG and Jumpshot
- New collective algorithms
- Better handling of startup in the `ch_p4` device.

New and improved in 1.2.4:

- Added support for the POSIX `sched_yield` call. This significantly increases the performance of the `ch_shmem` device and `ch_p4` device when `-comm=shared` is chosen, when there are more total processes (including operating system and other user processes) than processors on a node. This is now the default where available. However, when there are fewer processes than processors, lower latencies can be achieved by configuring with `--disable-yield`.
- The option `--with-common-prefix=dir` can be used to set the directory path for installing tools such as `upshot` and `jumpshot` that are independent of the MPICH device. For example,

```
configure --with-device=ch_p4 --prefix=/usr/local/mpich-1.2.4/ch_p4 \
    --with-common-prefix=/usr/local/mpich-1.2.4
make
make install
configure --with-device=ch_p4mpd  --prefix=/usr/local/mpich-1.2.4/ch-p4mpd \
    --with-common-prefix=/usr/local/mpich-1.2.4
```

- Added CYGWIN_NT to known architectures.
- Added more support for the MPI-2 feature MPI_STATUS_NULL.
- Fixed several bugs in the flow control code that could cause programs to hang.
- Many improvements to the MPICH-NT version.
- A bug in the autoconf macro AC_CHECK_HEADER has been fixed; this caused configure to conclude that header files were not available even though they were.
- Fixes for NETBSD (usc code in ch_p4)
-

New and improved in 1.2.3:

- Reorganized manuals, arranged by device and including a frequently asked questions section.
- Further improvements to the ch_p4mpd device.
- Improvements to the SMP support in the ch_p4 device, thanks to Pete Wykcoff.
- Support for IA64 systems running Linux and for Mac OS X.
- Support for version 2 of Globus in the globus2 device.
- Many bug fixes and code improvements. See www.mcs.anl.gov/mpi/mpich/r1_2_3changes.html for a complete list of changes.

New and improved in 1.2.2:

- A greatly improved ch_p4mpd device.
- Improved support for assorted Fortran 77 and Fortran 90 compilers, including compile-time evaluation of Fortran constants used in the MPICH implementation.
- An improved globus2 device, providing better performance.
- A new bproc mode for the ch_p4 device supports Scyld Beowulfs.
- Many TCP performance improvements for the ch_p4 and ch_p4mpd devices, as well as
- Many bug fixes and code improvements. See www.mcs.anl.gov/mpi/mpich/r1_2_2changes.html for a complete list of changes.

1.2 Choosing the correct device and release

MPICH is designed to be ported and optimized for a variety of systems though implementations of an *abstract device interface* (ADI). Many different implementations of the ADI exist; each is called a *device*.

Workstation Networks, Beowulf Clusters, and Individual Workstations. The most important devices are ch_p4 and ch_p4mpd. The ch_p4 device is the most general and supports SMP nodes, MPMD programs, and heterogeneous collections of systems. The ch_p4mpd device (so far) supports only homogenous clusters of uniprocessors, but provides for far faster and more scalable startup.

Grids. The globus2 uses Globus (www.globus.org) to provide a grid-enabled implementation of MPI. This device is appropriate for collections of systems on which Globus is installed.

Symmetric Multiprocessors. The ch_shmem device is appropriate for a single shared-memory system, such as a SGI Origin or Sun E10000. This device uses shared memory to pass messages between processes, and makes use of facilities provided by the operating system such as System V shared memory or anonymous mmap regions for data and System V semaphores or OS-specific mutex routines for synchronization. The ch_lfshmem device is a version of ch_shmem that uses a lock-free approach and that we developed for the NEC SX-4 [7]. The ch_shmem is appropriate for most systems; the ch_lfshmem requires special assembly language coding but can be ported to most systems. Currently, however, the ch_lfshmem supports only the NEC SX-4.

Massively Parallel Processors (MPPs). MPICH was originally developed to provide an implementation of MPI for many of the massively parallel processors, each of which had its own, propritary, message-passing system. We still include the devices ch_meiko, ch_nx and ch_mpl with MPICH releases, though most of these systems have now disappeared.

Others. MPICH was designed to enable other groups to use it when developing their own MPI implementations. Both vendors and research groups have used MPICH as the basis for their implementation. One important one in the cluster environment is MPICH-GM, for Myrinet switch-connected clusters; it is available directly from Myricom.

To make it easier to build and use MPICH, customized versions of this manual are constructured for each major device.

1.3 Citations and References

The name MPICH is derived from MPI and Chameleon; Chameleon both because MPICH can run (adapt its color) on a wide range of environments and because the initial implementation of MPICH used the Chameleon [14] message-passing portability system. MPICH is pronounced "Em Pee Eye See Aych," not "Emm Pitch." The proper references for MPICH are the paper in the journal *Parallel Computing* [10] and this manual [9]. BibTeX versions of these citations are available at www.mcs.anl.gov/mpi/mpich/mpich-prefer-ref.html.

2 Quick Start

Here is a set of steps for setting up and minimally testing MPICH. Details and instructions for a more thorough tour of MPICH's features, including installing, validating, benchmarking, and using the performance evaluation tools, are given in the following sections.

2.1 Downloading MPICH

The first step is to download MPICH and install any necessary patches.

1. The easiest way to get MPICH is to use the web page `www.mcs.anl.gov/mpi/mpich/download.html`; you can also use anonymous ftp from `ftp.mcs.anl.gov` in directory '`pub/mpi`'. Get the file '`mpich.tar.gz`'. (If that file is too big, try getting the pieces from `pub/mpi/mpisplit` and `cat`ting them together.)

2. Unpack the '`mpich.tar.gz`' file into a build directory. We recommend using a locally mounted partition rather than an NFS (network file system) partition. For example, on many systems, '`/tmp`' or '`/sandbox`' are locally mounted. Make sure that there is enough space available (100MB should be more than enough). To unpack, assuming that '`mpich.tar.gz`' has been downloaded into '`/tmp`', use

   ```
   % cd /tmp
   % tar zxovf mpich.tar.gz
   ```

   If your `tar` does not accept the `z` option, use

   ```
   % cd /tmp
   % gunzip -c mpich.tar.gz | tar zxovf -
   ```

3. Apply any patches. Check the web page `www.mcs.anl.gov/mpi/mpich/buglist-tbl.html` for any patches that need to be applied. Normally, versions of MPICH that already have these important patches applied are available; they are indicated by a fourth number in the release name (e.g., 1.2.2.3). But in some cases, a patch is made available early. The patch page has instructions on applying the patches.

Now you are ready to build MPICH.

2.2 Configuring, Making, and Installing

Before you can use MPICH, you must configure and make it. The configuration process analyzes your system and determines the correct options and settings; it also creates the '`Makefile`'s that are used to make MPICH.

1. Decide where you want to install MPICH. This step is not strictly necessary (except in the case of the `ch_p4mpd` device); however, installing MPICH (which can be done

without any privilages in a user directory) both makes it easier to manage updates to MPICH and allows you to reduce the amount disk space that MPICH takes up, since the installed version contains only the libraries, header files, documentation, and supporting programs. We recommend an install path that contains the version number of MPICH. For example, if you are installing MPICH for others to use, and you have the required access rights, you could choose '/usr/local/mpich-1.2.6/'. If you are installing it just for your own use, you could use something like '/home/me/software/mpich-1.2.6'.

2. Invoke configure with the appropriate prefix:

   ```
   % ./configure --prefix=/usr/local/mpich-1.2.6 |& tee c.log
   ```

   This will configure MPICH for the default device; this is usually the appropriate choice. Section 4.1 discusses the options that can be given to configure to customize MPICH.

   The output of configure is piped to tee; this program both writes the output to the file specified by its argument (here 'c.log') and to standard output. If you have trouble with the configure or make step, the file 'c.log' will help identify any problems.

3. Make MPICH:

   ```
   % make |& tee make.log
   ```

   This may take a while, depending on the load on your system and on your file server, it may take anywhere from a few minutes to an hour or more.

2.3 Running examples

1. (Optional) Build and run a simple test program:

   ```
   % cd examples/basic
   % make cpi
   % ../../bin/mpirun -np 4 cpi
   ```

   At this point you have run an MPI program on your system. If you have trouble, see Section 6.

2. (Optional) Put the distribution through its complete acceptance test (See Section 4.10 for how to do this).

3. (Optional) If you wish to install MPICH in a public place so that others may use it, use

   ```
   % make install
   ```

   to install MPICH into the directory specified by the --prefix option to configure. Installation will consist of an 'include', 'lib', 'bin', 'sbin', 'www', and 'man' directories and a small 'examples' directory. Should you wish to remove the installation, you can run the script sbin/mpiuninstall.

4. (Optional) At this point you can announce to your users how to compile and run MPI programs, using the installation you have just built in '`/usr/local/mpich-1.2.6/`' (or wherever you have installed it). See Section 3 for commands they can use. They can also copy the '`Makefile`' in '`/usr/local/mpich-1.2.6/examples`' and adapt it for their own use.

2.4 Sample MPI programs

The `MPICH` distribution contains a variety of sample programs, which are located in the `MPICH` source tree. Most of these will work with any MPI implementation, not just `MPICH`.

examples/basic contains a few short programs in Fortran, C, and C++ for testing the simplest features of MPI.

examples/test contains multiple test directories for the various parts of MPI. Enter "make testing" in this directory to run our suite of function tests.

examples/perftest Performance benchmarking programs. See the script `runmpptest` for information on how to run the benchmarks. These are relatively sophisticated.

mpe/contrib/mandel A Mandelbrot program that uses the MPE graphics package that comes with mpich. It should work with any other MPI implementation as well, but we have not tested it. This is a good demo program if you have a fast X server and not too many processes.

mpe/contrib/mastermind A program for solving the Mastermind puzzle in parallel. It can use graphics (`gmm`) or not (`mm`).

Additional examples from the book *Using MPI* [11] are available at `www.mcs.anl.gov/mpi/using`. Tutorial material on MPI can also be found at `www.mcs.anl.gov/mpi`.

3 Programming Tools

The `MPICH` implementation comes with a variety of tools for building, running, debugging, and analyzing MPI programs. This section details these tools.

3.1 Compiling, linking, and running programs

The `MPICH` implementation provides four commands for compiling and linking C (`mpicc`), C++ (`mpicxx`), Fortran 77 (`mpif77`), and Fortran 90 (`mpif90`) programs.

Use these commands just like the usual C, Fortran 77, C++, or Fortran compilers. For example,

```
mpicc -c foo.c
mpif77 -c foo.f
```

```
mpicxx -c foo.cxx
mpif90 -c foo.f
```

and

```
mpicc -o foo foo.o
mpif77 -o foo foo.o
mpicxx -o foo foo.o
mpif90 -o foo foo.o
```

Commands for the linker may include additional libraries. For example, to use routines from the C math library library, use

```
mpicc -o foo foo.o -lm
```

Combining compilation and linking in a single command, as shown here,

```
mpicc -o foo foo.c
mpif77 -o foo foo.f
mpicxx -o foo foo.cxx
mpif90 -o foo foo.f
```

may also be used (on most systems).

Note that while the suffixes `.c` for C programs and `.f` for Fortran-77 programs are standard, there is no consensus for the suffixes for C++ and Fortran-90 programs. The ones shown here are accepted by many but not all systems. `MPICH` tries to determine the accepted suffixes, but may not always be able to.

Earlier versions of MPICH used `mpiCC` to compile C++ programs. However, on some operating systems, such as Mac OS X, filenames that differ only in case, such as '`mpicc`' and '`mpiCC`', refer to the same file. To avoid problems caused when scripts and Makefiles that use the MPICH compilation scripts are ported to such systems, we have chnaged to using only `mpicxx` for compiling and linking C++ programs. We also encourage users to use '`.cxx`' or '`.cc`' as the file extension for C++ programs rather than '`.C`' (because it is the same as '`.c`' on OS X) or '`.cpp`' (because some applications use that suffix for files created with the C preprocessor).

You can override the choice of compiler by specifying the environment variable `MPICH_-CC`, `MPICH_F77`, `MPICH_CCC`, or `MPICH_F90`. However, be warned that this will work only if the alternate compiler is compatible with the default one (by compatible, we mean that is uses the same sizes for all datatypes and layouts, and generates object code that can be used with the `MPICH` libraries). If you wish to override the linker, use the environment variables `MPICH_CLINKER`, `MPICH_F77LINKER`, `MPICH_CCLINKER`, or `MPICH_F90LINKER`.

If you want to see the commands that would be used without actually running them, add the command line argument `-show`.

In addition, the following special options are supported for accessing some of the features of the MPE environment for monitoring MPI calls from within an application:

-mpilog Build version that generates MPE log files.

-mpitrace Build version that generates traces.

-mpianim Build version that generates real-time animation.

These are described in more detail in Section 3.6.

3.1.1 Compiling and Linking without the Scripts

In some cases, it is not possible to use the scripts supplied by MPICH for compiling and linking programs. For example, another tool may have its own compilation scripts. In this case, you can use `-compile_info` and `-link_info` to have the MPICH compilation scripts indicate the compiler flags and linking libraries that are required for correct operation of the MPICH routines. For example, when using the `ch_shmem` device on Solaris systems, the library `thread` (`-lthread`) must be linked with the application. If the `thread` library is not provided, the application will still link, but essential routines will be replaced with dummy versions contained within the Solaris C library, causing the application to fail.

For example, to determine the flags used to compile and link C programs, you can use these commands, whose output for the `ch_p4` device on a Linux workstation is shown.

```
% mpicc -compile_info
cc -DUSE_STDARG -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1
-DHAVE_STDARG_H=1 -DUSE_STDARG=1 -DMALLOC_RET_VOID=1
-I/usr/local/mpich/include -c

% mpicc -link_info
cc -DUSE_STDARG -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1
-DHAVE_STDARG_H=1 -DUSE_STDARG=1 -DMALLOC_RET_VOID=1
-L/usr/local/mpich/lib -lmpich
```

3.2 Running programs with `mpirun`

To run an MPI program, use the `mpirun` command, which is located in '`/usr/local/mpich-1.2.6/bin`'. For almost all systems, you can use the command

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

to run the program '`a.out`' on four processors. The command `mpirun -help` gives you a complete list of options, which may also be found in Appendix E.

On exit, `mpirun` returns the status of one of the processes, usually the process with rank zero in `MPI_COMM_WORLD`.

3.3 Symmetric Multiprocessors (SMPs) and the ch_shmem device

On many of the shared-memory implementations, MPICH reserves some shared memory in which messages are transferred back and forth. By default, MPICH reserves roughly four MBytes of shared memory. You can change this with the environment variable MPI_-GLOBMEMSIZE. For example, to make it 8 MB, enter

```
setenv MPI_GLOBMEMSIZE 8388608
```

Large messages are transfered in fragments, so MPI_GLOBMEMSIZE does not limit the maximum message size but increasing it may improve performance. Also note that systems may limit the amount of shared memory available.

By default, MPICH limits the number of processes for the ch_shmem device to 32, unless it determines at configure time that the machine has more processors. You can override this limit by setting the environment variable PROCESSOR_COUNT to the maximum number of processes that you will want to run, and then reconfigure and remake MPICH.

3.4 MPMD Programs

It is possible to run a parallel program with different executables with several of the devices including the ch_p4, ch_mpl, and globus2 devices. It is currently not possible to do this with the ch_shmem or ch_p4mpd devices. This style of parallel programming is often called MPMD for "multiple program multiple data". In many cases, it is easy to convert a MPMD program into a single program that uses the rank of the process to invoke a different routine; doing so makes it easier to start parallel programs and often to debug them. If converting a MPMD program to a SPMD (single program multiple data, not to be confused with single *instruction* multiple data, or SIMD) is not feasible, then you can run MPMD programs using MPICH. However, you will not be able to use mpirun to start the programs; instead, you will need to follow the instructions for each device.

3.5 Debugging

Debugging parallel programs is notoriously difficult. Parallel programs are subject not only to the usual kinds of bugs but also to new kinds having to do with timing and synchronization errors. Often, the program "hangs," for example when a process is waiting for a message to arrive that is never sent or is sent with the wrong tag. Parallel bugs often disappear precisely when you adds code to try to identify the bug, which is particularly frustrating. In this section we discuss several approaches to parallel debugging.

3.5.1 The printf Approach

Just as in sequential debugging, you often wish to trace interesting events in the program by printing trace messages. Usually you wish to identify a message by the rank of the process emitting it. This can be done explicitly by putting the rank in the trace message.

3.5.2 Error handlers

The MPI Standard specifies a mechanism for installing one's own error handler, and specifies the behavior of two predefined ones, MPI_ERRORS_RETURN and MPI_ERRORS_ARE_FATAL. As part of the MPE library, we include two other error handlers to facilitate the use of command-line debuggers such as dbx in debugging MPI programs.

```
MPE_Errors_call_dbx_in_xterm
MPE_Signals_call_debugger
```

These error handlers are located in the MPE directory. A configure option (-mpedbg) includes these error handlers into the regular MPI libraries, and allows the command-line argument -mpedbg to make MPE_Errors_call_dbx_in_xterm the default error handler (instead of MPI_ERRORS_ARE_FATAL).

3.5.3 Starting jobs with a debugger

The -dbg=<name of debugger> option to mpirun causes processes to be run under the control of the chosen debugger. For example, enter

```
mpirun -dbg=gdb or mpirun -dbg=gdb a.out
```

invokes the mpirun_dbg.gdb script located in the 'mpich/bin' directory. This script captures the correct arguments, invokes the gdb debugger, and starts the first process under gdb where possible. There are five debugger scripts; ddd, gdb, xxgdb, ddd, and totalview. These may need to be edited depending on your system. There is another debugger script for dbx, but this one will always need to be edited as the debugger commands for dbx varies between versions. You can also use this option to call another debugger; for example, -dbg=mydebug. All you need to do is write a script file, 'mpirun_dbg.mydebug', which follows the format of the included debugger scripts, and place it in the 'mpich/bin' directory. More information on using the Totalview debugger with MPICH can be found in Section 3.5.6.

3.5.4 Starting the debugger when an error occurs

It is often convenient to have a debugger start when a program detects an error. If MPICH was configured with the option --enable-mpedbg, then adding the command-line option -mpedbg to the program will cause MPICH to attempt to start a debugger (usually dbx or gdb) when an error that generates a signal (such as SIGSEGV) occurs. For example,

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

If you are not sure if your MPICH provides this service, you can use -mpiversion to see if MPICH was built with the --enable-mpedbg option. This feature may not be available with all devices.

3.5.5 Attaching a debugger to a running program

On workstation clusters, you can often attach a debugger to a running process. For example, the debugger `dbx` often accepts a process id (pid) which you can get by using the `ps` command. The form may be either

```
dbx a.out 1234
```

or

```
dbx -pid 1234 a.out
```

where `1234` is the process id[1].

To do this with `gdb`, start `gdb` and at its prompt do

```
file a.out
attach 1234
```

One can also attach the TotalView debugger to a running program (See Section 3.5.6 below).

3.5.6 Debugging MPI programs with TotalView

TotalView© [18] is a powerful, commercial-grade, portable debugger for parallel and multithreaded programs, available from Etnus (`http://www.etnus.com/`). TotalView understands multiple MPI implementations, including `MPICH`. By "understand" is meant that if you have TotalView installed on your system, it is easy to start your `MPICH` program under the control of TotalView, even if you are running on multiple machines, manage your processes both collectively and individually through TotalView's convenient GUI, and even examine internal `MPICH` data structures to look at message queues [3]. The general operation model of TotalView will be familiar to users of command-line-based debuggers such as `gdb` or `dbx`.

Starting an `MPICH` program under TotalView control. To start a parallel program under TotalView control, simply add '`-dbg=totalview`' to your `mpirun` arguments:

```
mpirun -dbg=totalview -np 4 cpi
```

TotalView will come up and you can start the program by typing '`G`'. A window will come up asking whether you want to stop processes as they execute `MPI_Init`. You may find it more convenient to say "no" and instead to set your own breakpoint after `MPI_Init` (see Section 3.5.6). This way when the process stops it will be on a line in your program instead of somewhere inside `MPI_Init`.

[1] The unnecessary changes in the command syntax for programs such as `dbx` contributes to the poor state of software.

Attaching to a running program. TotalView can attach to a running MPI program, which is particularly useful if you suspect that your code has deadlocked. To do this start TotalView with no arguments, and then press 'N' in the root window. This will bring up a list of the processes that you can attach to. When you dive through the initial MPICH process in this window TotalView will also acquire all of the other MPICH processes (even if they are not local). See the TotalView manual for more details of this process.

Debugging with TotalView. You can set breakpoints by clicking in the left margin on a line number. Most of the TotalView GUI is self-explanatory. You select things with the left mouse button, bring up an action menu with the middle button, and "dive" into functions, variables, structures, processes, etc., with the right button. Pressing `cntl-?` in any TotalView window brings up help relevant to that window. In the initial TotalView window it brings up general help. The full documentation (The TotalView *User's Guide*) is available from the Etnus web site.

You switch from viewing one process to the next with the arrow buttons at the top-right corner of the main window, or by explicitly selecting (left button) a process in the root window to re-focus an existing window onto that process, or by diving (right button) through a process in the root window to open a new window for the selected process. All the keyboard shortcuts for commands are listed in the menu that is attached to the middle button. The commands are mostly the familiar ones. The special one for MPI is the 'm' command, which displays message queues associated with the process.

Note also that if you use the MPI-2 function `MPI_Comm_set_name` on a communicator, TotalView will display this name whenever showing information about the communicator, making it easier to understand which communicator is which.

3.6 Log and tracefile tools

The MPE libraries that are distributed with MPICH contain several libraries for either logging information about the execution of each MPI call to a file for later analysis or for tracing each call as it occurs. These libraries may be accessed by simply providing a command line argument to the compilation scripts; further, this needs only be done when linking the program. For example, to create a log file of a program such as `cpi`, the following steps are needed:

```
mpicc -c cpi.c
mpicc -o cpi -mpilog cpi.o
```

The log file will be written to a file with the name '`cpi.clog`' or '`cpi.slog`', depending on the value of the environment variable `MPE_LOG_FORMAT` (clog is the default). The logfile can be graphically displayed using the Jumpshot program, described in Section 3.6.1.

Another tool of interest is FPMPI [5]; this tool reports a summary of all MPI calls, providing information about message lenghts, communication patterns.

3.6.1 Jumpshot

Jumpshot is a program for displaying logfiles produced using the MPE logging libraries. Jumpshot is described in more detail in the *MPE Installation and User's manual* [2]. Jumpshot is a Java program and requires a functioning Java environment. You can view a logfile such as 'cpi.clog' in the example above by using

```
logviewer cpi.clog
```

There are several available log formats and `logviewer` selects the version of jumpshot appropriate for a particular logfile. See the MPE manual, distributed along with this manual, for more details.

3.7 Execution tracing

Execution tracing is easily accomplished using the `-mpitrace` command line argument while linking:

```
mpicc -c cpi.c
mpicc -o cpi -mpitrace cpi.o
```

3.8 Performance measurements

The 'mpich/examples/perftest' directory contains a sophisticated tool for measuring latency and bandwidth for `MPICH` on your system. Simply change to the `mpich/examples/perftest` directory and do

```
make
mpirun -np 2 mpptest -gnuplot > out.gpl
```

The file 'out.gpl' will then contain the necessary `gnuplot` commands. The file 'mppout.gpl' will contain the data. To view the data with `gnuplot`, use:

```
gnuplot out.gpl
```

or use

```
load 'out.gpl'
```

from within `gnuplot`. Depending on your environment and version of `gnuplot`, you may need to start `gnuplot` first and issue the command `set terminal x11` before executing '`load 'out.gpl'`'. You can use

```
gnuplot
set term postscript eps
set output "foo.eps"
load 'out.gpl'
```

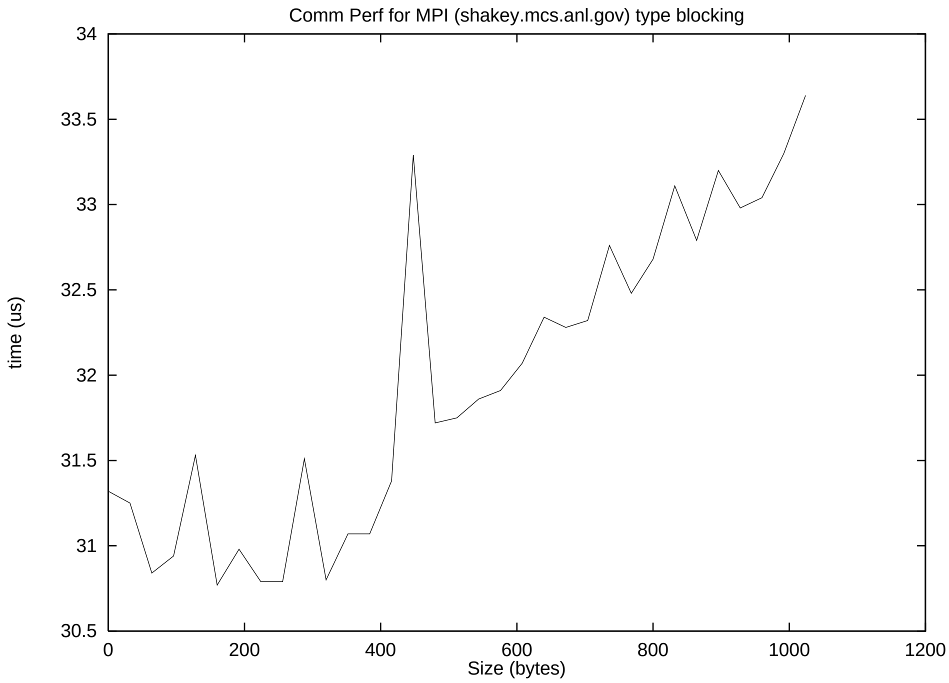

Figure 1: Sample output from `mpptest`

to create an Encapsulated Postscript graph such as the one in Figure 1.

The programs `mpptest` and `goptest` have a wide variety of capabilities; the option `-help` will list them. For example, `mpptest` can automatically pick message lengths to discover any sudden changes in behavior and can investigate the ability to overlap communication with computation. These programs are written using MPI, and may be used with *any* MPI implementation, not just `MPICH`. (See the `configure` file in the '`examples/perftest`' directory.) More information is available at `http://www.mcs.anl.gov/mpi/mpptest`.

Benchmarking can be very tricky. Some common benchmarking errors are discussed at `http://www.mcs.anl.gov/mpi/mpptest/hownot.html`. The paper [13] discusses these issues at more length.

4 Details

This section covers details that you should not normally need. Read this section if you need more detailed control over configuring, building, installing, or operating `MPICH`.

4.1 Configure options

The configure script documents itself in the following way. If you type

```
configure -usage
```

you will get a complete list of arguments and their meanings; these are also shown in Appendix D. The most important options are

`--prefix=dir` The installation prefix. `configure` understands all of the usual GNU installation directory arguments, including `--libdir` and `--mandir`. We recommend that all users specify an installation directory with `--prefix`.

`--with-device=devname` Set the `MPICH` device to use. `devname` must be the name of one of the directories in the '`mpid`' directory, such as `ch_p4`, `ch_shmem`, `globus2`, or `ch_p4mpd`.

`--enable-g` Add the `-g` option to the compile scripts. This is a prerequisite for using most debuggers, including `dbx`, and `gdb`.

`--enable-debug` Turn on support for the Totalview© Debugger. This allows Totalview to display information on message queues.

`--enable-sharedlib` Build both static and shared libraries for `MPICH`. This supports only a few systems, including those using `gcc` (e.g., most Linux Beowulf systems).

In addition, `configure` makes use of environment variables such as `MAKE`, `CC`, `FC`, `CFLAGS`, and `FFLAGS`.

Normally, you should use `configure` with as few arguments as you can. If you leave all arguments off, `configure` will usually guess the correct architecture (`arch`) unless you are in a cross-compiling environment, and will usually choose an appropriate device (`device`) as well. Where TCP/IP is an appropriate mechanism for communication, the TCP device (`ch_p4`) will be chosen by default.

4.1.1 MPI and PMPI routines

The MPI standard requires that each routine be available with both the MPI and PMPI prefix; for example, `MPI_Send` and `PMPI_Send`. `MPICH` attempts to use *weak symbols* to provide this feature; this reduces the size of the `MPICH` library. You can force `MPICH` to make separate libraries for the MPI and PMPI versions by adding the configure option `--disable-weak-symbols`:

```
configure --disable-weak-symbols ...
```

Some MPI routines are implemented in terms of other MPI routines. For example, in `MPICH`, `MPI_Bcast` is implemented using `MPI_Send`. When weak symbols are used, even the PMPI versions of the routines are implemented using the MPI (not PMPI) versions. If you want the PMPI routines to only use the PMPI routines, use `--disable-weak-symbols` when configuring `MPICH`. Note that this behavior may change in later releases.

4.1.2 Configuring `MPICH` for use with threads

The `MPICH` implementation of MPI is currently not threadsafe. It may, however, be possible to use `MPICH` in a threaded application as long as all `MPICH` calls are made by a single thread.

An example of this is OpenMP used for loop parallelism, combined with MPI.

4.1.3 Signals

In general, users should avoid using signals with MPI programs. The manual page for `MPI_Init` describes the signals that are used by the MPI implementation; these should not be changed by the user.

Because Unix does not chain signals, there is the possibility that several packages will attempt to use the same signal, causing the program to fail. For example, by default, the `ch_p4` device uses `SIGUSR1`; some thread packages also use `SIGUSR1`.

In a few cases, you can change the signal *before* calling `MPI_Init`. In those cases, your signal handler will be called after the `MPICH` implementation acts on the signal. For example, if you want to change the behavior of `SIGSEGV` to print a message, you can establish such a signal handler before calling `MPI_Init`. With devices such as the `ch_p4` device that handle `SIGSEGV`, this will cause your signal handler to be called after `MPICH` processes it.

HP Exemplar For a HP Exemplar, please get the official version from HP (formerly Convex). This was originally based on `MPICH`, but has been tuned for better performance on the Exemplar. If for some reason you want to use the shared memory version of `MPICH` on the HP, use

```
configure --with-device=ch_shmem --with-arch=hpux
```

SGI multiprocessors such as the Origin 2000. SGI's own MPI is highly tuned for these machines. If you do want to use `MPICH`, use

```
configure --with-device=ch_shmem
```

Configure attempts to determine the number of processors that are available; you can override this by setting the environment variable `PROCESSOR_COUNT` before running `configure`. Alternately, you can edit the file '`mpich/mpid/ch_shmem/shdef.h`' to adjust the maximum number of processors and memory that is used for communicating messages through shared memory. If you need to generate a particular version that corresponds to the `-32`, `-n32`, or `-64` compiler/linker options on SGI, use the architectures `IRIX32`, `IRIXN32`, or `IRIX64` respectively instead of SGI. Specifically, use the following for an R10000 or R12000 SGI:

```
./configure --with-arch=IRIX32
./configure --with-arch=IRIXN32
./configure --with-arch=IRIX64
```

If it is necessary to specify the specific compiler options, they must be specified by setting the `CC` and `FC` environment variables:

```
setenv CC "cc -32"
setenv FC "f77 -32"
```

```
configure --with-arch=IRIX32 \
    -opt="-O2" \
    --with-device=ch_shmem

setenv CC "cc -n32 -mips4 -r10000"
setenv FC "f77 -n32 -mips4 -r10000"
configure --with-arch=IRIXN32 \
    -opt="-O2" \
    --with-device=ch_shmem

setenv CC "cc -64 -mips4 -r10000"
setenv FC "f77 -64 -mips4 -r10000"
configure --with-arch=IRIX64 \
    -opt="-O2" \
    --with-device=ch_shmem
```

(The optimization level is optional; -O2 has worked for some users. Be careful of aggressive optimization, particularly in the '`mpid/ch_shmem`' code.)

See the comments under SGI workstations for different 32 and 64 bit options.

The configure option `--enable-yield` allows you to control how the `ch_shmem` device yields the processor when it is waiting for an incoming message. By default, it uses the POSIX function `sched_yield` if it is available. The option `--enable-yield=dynamic` allows you to use the environment variable `MPICH_YIELD` to control the choice of yield function. The value of `MPICH_YIELD` are

`sched_yield` Use POSIX `sched_yield`. This is the default, and is recommended.

`select` Use the `select` call. This should be used only for experimentation; in our tests, it has not worked well.

`none` Do not yield. This is appropriate when there are more than enough processors for the MPI environment and minimum latency is the overridding goal.

4.2 Alternate C Compilers

More and more systems, particularly clusters, come with multiple compilers. In many cases, you can build `MPICH` using just one of these compilers and then allow users to use their favorite compiler when building their code. In many cases, no extra steps are needed. Users can simply use the command-line argument `-cc` or the environment variable `MPICH_CC` to specify a different compiler.

Unfortunately, this won't always work. For example, in some cases, different libraries may be needed by different compilers when linking programs. Some compilers may need different options to support ANSI/ISO C. In particular, support for the longer datatypes such as `long long` and `long double` may rely on runtime library support routines that are specific to each compiler. To handle all of these cases, For C and C++, you can create a file with the name 'mpicc-<compilename>.conf', e.g., 'mpicc-gcc.conf' or 'mpicc-pgcc.conf', that contains any specifications that are needed by the mpicc command. mpicxx uses 'mpicxx-<name>.conf'. Currently, these files must be created by hand, starting from the file 'mpichlib.conf' that is created in the 'mpich/util' directory by the MPICH configure. The simplest way to create each file is to use configure:

```
setenv CC cc
configure --prefix=/usr/local/mpich-1.2.6
make
make install
cp util/mpichlib.conf /usr/local/mpich-1.2.6/etc/mpicc-cc.conf
setenv CC gcc
configure --prefix=/usr/local/mpich-1.2.6
make
cp util/mpichlib.conf /usr/local/mpich-1.2.6/etc/mpicc-gcc.conf
```

In this example, the default value of sysconfdir, $prefix/etc, is used.

For example, if MPICH was built with cc as the compiler but a user wanted to use gcc instead, either the commands

```
MPICH_CC=gcc
mpicc ...
```

or

```
mpicc -config=gcc ...
```

would case mpicc to load 'mpich-gcc.conf' and use the appropriate definitions.

4.3 Fortran Compilers

MPICH provides support for both Fortran 77 and Fortran 90. Because MPICH is implemented in C, using MPICH from Fortran can sometimes require special options. This section discusses some of the issues. Note that configure tries to determine the options needed to support Fortran. You need the information in this section only if you have problems. Section 4.3.6 discusses how to support multiple Fortran compilers (e.g., g77 and pgf77) with a single MPICH installation.

4.3.1 What if there is no Fortran compiler?

The configure program should discover that there is no Fortran compiler. You can force configure to not build the Fortran parts of the code with the option --disable-f77. In this case, only the C programs will be built and tested.

4.3.2 Fortran 90

During configuration, a number of Fortran 90-specific arguments can be specified. See the output of `configure -help`. In particular, when using the NAG Fortran 90 compiler, you should specify `-f90nag`.

4.3.3 Fortran 77 and Fortran 90

Selecting Fortran 90 with Fortran 77 should be done only when the two compilers are compatible, supporting the same datatypes and calling conventions. In particular, if the Fortran 90 compiler supports an 8-byte integer type, the Fortran 77 compiler must support `integer*8` (this is needed by the MPI-IO routines for the value of `MPI_OFFSET_KIND`). In addition, both compilers must support the same functions for accessing the command line, and the code for those commands must reside in the same library. If the two Fortran compilers are not compatible, you should either select the Fortran 90 compiler as both the Fortran 77 and Fortran 90 compiler (relying on the upward compatibility of Fortran), or build two separate configurations of `MPICH`. For example,

```
setenv FC f90
setenv F90 f90
configure
```

will use `f90` for both Fortran 77 and Fortran 90 programs. In many systems, this will work well. If there are reasons to have separate Fortran 90 and Fortran 77 builds, then execute the following commands (where `MPICH` is to be installed into the directory '`/usr/local`'):

```
setenv FC f77
configure --disable-f90 -prefix=/usr/local/mpich-1.2.6/f77-nof90
make
make install

setenv FC f90
setenv F90 f90
configure -prefix=/usr/local/mpich-1.2.6/f90
make
make install
```

This sequence of commands will build and install two versions of `MPICH`. An alternative approach that installs only a single version of `MPICH` is decribed in Section 4.3.6.

4.3.4 Fortran 90 Modules

If `configure` finds a Fortran 90 compiler, by default `MPICH` will try to create a Fortran 90 module for MPI. In fact, it will create two versions of an `mpi` module: one that includes only the MPI routines that do not take "choice" arguments and one that does include choice argument. A choice argument is an argument that can take any datatype; typically, these are the buffers in MPI communication routines such as `MPI_Send` and `MPI_Recv`.

The two different modules can be accessed with the `-nochoice` and `-choice` option to `mpif90`. The choice version of the module supports a limited set of datatypes (numeric scalars and numeric one- and two-dimensional arrays). This is an experimental feature; please send mail to `mpi-bugs@mcs.anl.gov` if you have any trouble. Neither of these modules offer full "extended Fortran support" as defined in the MPI-2 standard.

The reason for having two versions of the MPI module is that it is very difficult to provide a completely correct module that includes all of the functions with choice arguments. As it is, on many systems, the size of the Fortran 90 module to handle the routines with choice arguments will be larger than the entire C version of the MPI library. If you are uninterested in the Fortran 90 MPI module, or you wish to keep the installed version of `MPICH` small, you can turn off the creation of the Fortran 90 MPI module with the configure option `--disable-f90modules`.

4.3.5 Configuring with the Absoft Fortran Compiler

The Absoft compiler can be told to generate external symbols that are uppercase, lowercase, and lowercase with a trailing underscore (the most common case for other Unix Fortran compilers), or use mixed case (an extension of Fortran, which is only monocase). Each of these choices requires a *separate* `MPICH` configure and build step. `MPICH` has been tested in the mode where monocase names are generated; this case is supported because only this case supports common (and necessary for `MPICH`) extensions such as `getarg` and `iargc`. By default, `MPICH` forces the Absoft compiler to use lowercase; this matches most Unix Fortran compilers. `MPICH` will find the appropriate versions of `getarg` and `iargc` for this case. Because the examples and the test suite assume that the Fortran compiler is case-insensitive; the Fortran library produced by `MPICH` will only work with source code that uses monocase (either upper or lower) for all MPI calls.

In addition, you may need to use `-N90` if you use `character` data, because the `MPICH` Fortran interface expects the calling convention used by virtually all Unix Fortran systems (Cray UNICOS is handled separately). If you are building shared libraries, you will also need to set the environment variable `FC_SHARED_OPT` to `none`.

Early versions of the Absoft compiler could not handle multiple `-I` options. If you have trouble with this, you should get an update from Absoft.

4.3.6 Configuring for Multiple Fortran Compilers

In some environments, there are several different Fortran compilers, all of which define Fortran datatypes of the same size, and which can be used with the same C libraries. These compilers may make different choices for Fortran name mappings (e.g., the external format of the names given to the linker) and use different approaches to access the command line. This section describes how to configure `MPICH` to support multiple Fortran compilers. However, if any of these steps fails, the best approach is to build a separate `MPICH` installation for each Fortran compiler.

The first step is to configure MPICH with the `--with-flibname` option. For example, if one of the compilers is `g77`, use

```
setenv FC g77
./configure --with-flibname=mpich-g77 ... other options ...
```

After you build, test, *and install* this version of MPICH, you can configure support for additional Fortran compilers as follows:

1. Change directory to '`src/fortran`'
2. Execute

```
setenv FC pgf77
./configure --with-mpichconfig --with-flibname=mpich-pgf77
make
make install-alt
```

To use a particular Fortran compiler, either select it on the `mpif77` command line with the `-config=name` option (e.g., `-config=pgf77`) or by selecting a particular `mpif77` command (e.g., `mpif77-pgf77`).

4.4 C++

The C++ support in MPICH has been provided by Indiana University (formerly the University of Notre Dame), and uses its own configure process (it also supports other MPI implementations). This version supports only the MPI-1 functions, and does not include support for the MPI-2 functions such as I/O or the functions for manipulating `MPI_Info`. Questions, comments, suggestions, and requests for additional information should be sent to `mpi2cpp-devel@osl.iu.edu`. Bug reports should also be sent to `mpi-bugs@mcs.anl.gov`. More information is available at the web site `http://www.osl.iu.edu/research/mpi2c++/`.

Use the environment variable `CXX` to specify an alternate C++ compiler.

4.5 Using Shared Libraries

Shared libraries can help reduce the size of an executable. This is particularly valuable on clusters of workstations, where the executable must normally be copied over a network to each machine that is to execute the parallel program. However, there are some practical problems in using shared libraries; this section discusses some of them and how to solve most of those problems. Currently, shared libraries are not supported from C++.

In order to build shared libraries for MPICH, you must have configured and built MPICH with the `--enable-sharedlib` option. Because each Unix system and in fact each compiler uses a different and often incompatible set of options for creating shared objects and libraries, MPICH may not be able to determine the correct options. Currently, MPICH understands Solaris, GNU `gcc` (on most platforms, including Linux and Solaris), and IRIX.

Information on building shared libraries on other platforms should be sent to `mpi-bugs@mcs.anl.gov`.

Once the shared libraries are built, you must tell the `MPICH` compilation and linking commands to use shared libraries (the reason that shared libraries are not the default will become clear below). You can do this either with the command line option `-shlib` or by setting the environment variable `MPICH_USE_SHLIB` to `yes`. For example,

```
mpicc -o cpi -shlib cpi.c
```

or

```
setenv MPICH_USE_SHLIB yes
mpicc -o cpi cpi.c
```

Using the environment variable `MPICH_USE_SHLIB` allows you to control whether shared libraries are used without changing the compilation commands; this can be very useful for projects that use makefiles.

Running a program built with shared libraries can be tricky. If you have trouble, particular if programs will either not start or start and issue error messages about missing libraries, see Section 6.4.

4.6 File System Issues

Most users do not need to worry about file systems. However, there are two issues: using NFS (the Network File System) with MPI-IO and using NFS with some automounters. These issues are covered in the next two sections.

4.6.1 NFS and MPI-IO

To use MPI-IO multihost on NFS file systems, NFS should be version 3, and the shared NFS directory must be mounted with the "no attribute caching" (noac) option set (the directory cannot be automounted). If NFS is not mounted in this manner, the following error could occur:

```
MPI_Barrier: Internal MPI error: No such file or directory
File locking messages
```

In order to reconfigure NFS to handle MPI-IO properly, the following sequence of steps are needed (root permission required):

1. confirm you are running NFS version 3

   ```
   rpcinfo -p `hostname` | grep nfs
   ```

```
for example, there should be a '3' in the second column
fire >rpcinfo -p fire | grep nfs
100003    3   udp   2049  nfs
```

2. edit '/etc/fstab' for each NFS directory read/written by MPI-IO on each machine used for multihost MPI-IO. The following is an example of a correct fstab entry for /epm1:

   ```
   root >grep epm1 /etc/fstab
   gershwin:/epm1 /rmt/gershwin/epm1 nfs bg,intr,noac 0 0
   ```

   If the "noac" option is not present, add it and then remount this directory on each of the machines that will be used to share MPI-IO files.

   ```
   root >umount /rmt/gershwin/epm1
   root >mount /rmt/gershwin/epm1
   ```

3. confirm that the directory is mounted noac

   ```
   root >grep gershwin /etc/mnttab
   gershwin:/epm1 /rmt/gershwin/epm1 nfs
   noac,acregmin=0,acregmax=0,acdirmin=0,acdirmax=0 0 0 899911504
   ```

Turning off of attribute caching may reduce performance of MPI-IO applications as well as other applications using this directory. The load on the machine where the NFS directory is hosted will increase.

4.7 Building MPICH

Once `configure` has determined the features of your system, all you have to do now is

```
make
```

This will clean all the directories of previous object files (if any), compile both profiling and non-profiling versions of the source code, including Romio and the C++ interface, build all necessary libraries, and link both a sample Fortran program and a sample C program as a test that everything is working. If anything goes wrong, check Section 6 to see if there is anything said there about your problem. If not, follow the directions in Section 6.2 for submitting a bug report. To simplify checking for problems, it is a good idea to use

```
make |& tee make.log
```

Specific (non-default) targets can also be made. See the '`Makefile`' to see what they are.

After running this `make`, the size of the distribution will be about 45 Megabytes (depending on the particular machine it is being compiled for and the selected options), before building any of the examples or the extensive test library. The '`Makefile`'s are built for the various example subdirectories, but the example programs themselves have to be made "by hand".

4.8 Installing MPICH for Others to Use

This step is optional. However, if you are installing MPICH, you should make sure that you specified the directory into which MPICH is to be installed when you configure MPICH by using the -prefix option. For example, if you plan to install MPICH into '/usr/local/mpich-1.2.6/', then you should configure with the option -prefix='/usr/local/mpich-1.2.6/'. If there is any possibility at all that you will build MPICH for several systems and/or devices, you should include that information in the prefix. For example, by using

```
-prefix='/usr/local/mpich-1.2.6/solaris/ch_p4',
```

you can later add

```
-prefix='/usr/local/mpich-1.2.6/solaris/ch_p4smp'
```

for a version that is built with the configure option -comm=shared (suitable for clusters of symmetric multiprocessors, hence the "smp" in the directory name). Once you have tested all parts of the MPI distribution (including the tools, particularly upshot and/or nupshot), you may install MPICH into a publically available directory, and disseminate information for other users, so that everyone can use the shared installation. To install the libraries and include files in a publicly available place, change to the top-level MPICH directory, and do

```
make install
```

The man pages will have been copied with the installation, so you might want to add '/usr/local/mpich-1.2.6/man' to the default system MANPATH. The man pages can be conveniently browsed with the mpiman command, found in the mpich/bin directory.

It is possible to specify the directory into which MPICH should be installed after building MPICH by setting the value of PREFIX when executing the installation step:

```
make install PREFIX=/usr/local/mpich-1.2.6
```

However, some features, particularly the ability of Totalview to show MPICH message queues, will work only if MPICH is configured with the prefix set to the installation directory.

A good way to handle multiple releases of MPICH is to install them into directories whose names include the version number and then set a link from mpi to that directory. For example, if the current version is 1.2.6, the installation commands to install into '/usr/local' are

```
make install PREFIX=/usr/local/mpi-1.2.6
rm /usr/local/mpi
ln -s /usr/local/mpi-1.2.6 /usr/local/mpi
```

The script 'bin/mpiinstall' provides more control over the installation of MPICH (in fact, make install just runs this script). For example, you can change the protection of the files when they are installed with the options `-mode=nnnn` (for regular files) and `-xmode=nnnn` (for executables and directories). You can set the directory into which the man pages will be placed with `-manpath=<path>`. The option `-help` shows the full set of options for `mpiinstall`.

Installing `nupshot` can sometimes be troublesome. You can use the switch `-nonupshot` to `mpiinstall` to not install `nupshot`; alternately, you can use the switch `-cpnupshot` to install the copy in '`mpich/profiling/nupshot`'. Normally, `mpiinstall` builds a new version of `nupshot` to insure that all of the paths are correct (`nupshot` needs to find files where it is installed). If you need to "manually" build `nupshot` for installation, the `-cpnupshot` switch will allow you to install that version.

You can test the installation by using the `configure` in '`mpich/examples/test`'. For example, if you have installed `MPICH` into '`/usr/local/mpich-1.2.6/`' for architecture `solaris` and device `ch_p4`, execute

```
cd examples/test
./configure -mpichpath=/usr/local/mpich-1.2.6/solaris/ch_p4/bin
make testing
```

The test codes in the '`mpich/examples/test`' directory may be used with *any* implementation of MPI, not just the `MPICH` implementation.

4.8.1 User commands

The commands `mpirun`, `mpicc`, `mpif77`, `mpicxx`, `mpif90`, and `mpiman` should be in the user's search path. Note that if several architectures and/or `MPICH` devices are supported, it is important that the correct directory be added to the user's path. These are installed into the '`bin`' directory. If multiple architectures or devices are being used, be sure that the installation path distinguishes these. For example, if both the `ch_p4` and `ch_shmem` devices are built on a Solaris system, set the installation prefix to include these names:

> `-prefix=`'`/usr/local/mpich-1.2.6/solaris/ch_p4`', and
>
> `-prefix=`'`/usr/local/mpich-1.2.6/solaris/ch_shmem`'

respectively. Alternately, set the prefix to '`/usr/local/mpich-1.2.6/`' and set the execprefix to

> '`/usr/local/mpich-1.2.6/solaris/ch_p4`' and
>
> '`/usr/local/mpich-1.2.6/solaris/ch_shmem`'

respectively.

4.8.2 Installing documentation

The `MPICH` implementation comes with several kinds of documentation. Installers are encouraged to provide site-specific information, such as the location of the installation (particularly if it is not in '`/usr/local/mpich-1.2.6/`').

4.8.3 Man pages

A complete set of Unix man pages for the `MPICH` implementation are in '`mpich/man`'. '`man/man1`' contains the commands for compiling, linking, and running MPI programs; '`man/man3`' contains the MPI routines; '`man/man4`' contains the MPE routines, and '`man/man5`' contains the MPID routines (these are for the low-level part of the `MPICH` implementation, are not of interest to users). The command '`mpich/bin/mpiman`' is a script that can present the manual pages for `MPICH` in various forms, using either the terminal-style `man`,`xman`, or one of several HTML browsers.

4.8.4 Examples

Users often prefer working from example '`Makefile`'s and programs. The directory that is installed in the '`examples`' directory contains a C and Fortran version of the 'pi' program, along with a '`Makefile.in`'. Other examples there include a simple parallel I/O program and an MPI program written using the C++ bindings for the MPI functions. Users may be interested in some of the examples that are in the source tree, also in the '`examples`' directory.

4.9 Special Considerations for Running with Shared Memory

When using the `ch_shmem` or `ch_lfshmem` devices with System V shared memory, processes that exit abnormally (e.g., with a segmentation violation) may leave System V semaphores or shared memory segments allocated[2]. Since there is usually a limited number of these objects, it is important to recover them. The Unix command `ipcs` can be used to list the allocated semaphores and shared memory segments, and `ipcrm` can be used to delete them. The script '`bin/cleanipcs`' can be used to identify and delete *all* System V IPCs owned by the calling user; the use is simply

```
bin/cleanipcs
```

4.10 Thorough Testing

The '`examples/test`' directory contains subdirectories of small programs that systematically test a large subset of the MPI functions. The command

```
make testing
```

in the `MPICH` directory will cause these programs to be compiled, linked, executed, and their output to be compared with the expected output. Linking all these test programs takes up considerable space, so you might want to do

```
make clean
```

[2]Surprisingly, the System V IPC (interprocess communication) mechanisms do not have a "delete on unreferenced" attribute.

in the test directory afterwards. The individual parts of MPI (point-to-point, collective, topology, etc.) can be tested separately by

```
make testing
```

in the separate subdirectories for `examples/test`.

If you have a problem, first check the troubleshooting guides and the lists of known problems. If you still need help, send detailed information to `mpi-bugs@mcs.anl.gov`.

4.11 Internationalization

`MPICH` has support for providing error messages in different languages. This makes use of the X/Open message catalog services, which are a standard way of providing multi-language support. This multi-language support is often called NLS, for National Language Support. `MPICH` comes with error messages in US English; additional languages will be provided as we get the translations (if you wish to provide one, please send mail to `mpi-bugs@mcs.anl.gov`). More precisely, `MPICH` uses an English version that uses the ISO Latin-1 character set (ISO8859-1). We expect to provide other versions that also use the Latin-1 character set, subject to getting translations of the messages.

To create a new message catalog, copy the file '`mpich.En_US.msg`' to '`mpich.mylanguage.msg`' and translate the entries. The value of '`mylanguage`' should match the ones used for your system; for example, '`mpich.De_DE.msg`' for German. Many systems put their NLS files in '`/usr/lib/nls/msg`'; you can also check the value of the environment variable `NLSPATH` on your system. Note that some systems provide the routines and programs to support NLS, but do not make use of it and do not provide a initial `NLSPATH` value.

For emacs users, check the Emacs info under "European Display". The commands

```
M-x standard-display-european
M-x iso-accents-mode
```

can be used to input most European languages. You can also load '`iso-transl`' and use `C-x 8` to compose characters (this sets the high bit in the character). `MPICH` currently does not support languages that require multi-byte character sets (such as Japanese). However, the only changes needed are in the file '`src/env/errmsg.c`'; if you are interested in developing a multi-byte character set version, please let us know.

By default, `MPICH` uses the value of '`NLSPATH`' to find the message catalogs. If this fails, it tries '`MPICHNLSPATH`', and if that fails, it uses English language versions that are coded into the library.

The catalogs are not, however, installed into these directories. Instead, you will find them in the library directory for a particular architecture; for example, '`mpich/rs6000/lib`'.

5 Documentation

This distribution of `MPICH` comes with complete `man` pages for the MPI routines, commands for compiling and linking MPI programs, and the MPE extensions. The command `mpiman` in '`mpich/bin`' is a good interface to the man pages.[3] The '`mpich/www`' directory contains HTML versions of the man pages for MPI and MPE. The '`mpich/doc`' directory contains this *Installation and User's Guide*. All documentation is also available on the web at `www.mcs.anl.gov/mpi/mpich/docs.html`.

Information about MPI is available from a variety of sources. Some of these, particularly WWW pages, include pointers to other resources.

- The Standard itself:
  - As a Technical report [4].
  - As Postscript and HTML at `www.mpi-forum.org`, for both MPI-1 and MPI-2.
  - As a journal article in the Fall 1994 issue of the Journal of Supercomputing Applications [15] for MPI-1 and as a journal article in the International Journal of High Performance Computing Applications in 1998.
- MPI Forum discussions
  - The MPI Forum email discussions and both current and earlier versions of the Standard are available from `www.netlib.org`. MPI-2 discussions are available at `www.mpi-forum.org`.
- Books:
  - *Using MPI: Portable Parallel Programming with the Message-Passing Interface, Second Edition*, by Gropp, Lusk, and Skjellum [11].
  - *Using MPI-2: Advanced Features of the Message-Passing Interface*, by Gropp, Lusk, and Thakur [12].
  - *MPI—The Complete Reference: Volume 1, The MPI Core*, by Snir, et al. [17].
  - *MPI—The Complete Reference: Volume 2, The MPI-2 Extensions*, by Gropp, et al. [6].
  - *Parallel Programming with MPI*, by Peter S. Pacheco [16].
- Newsgroup:
  - `comp.parallel.mpi`
- Mailing lists:
  - `mpi-comments@mpi-forum.org`: The MPI Forum discussion list.
  - `mpi-impl@mcs.anl.gov`: The implementors' discussion list.
  - `mpi-bugs@mcs.anl.gov` is the address to which you should report problems with `mpich`.

[3] The `mpiman` command is created by the configure process.

- Implementations available from the web:
  - `MPICH` is available from `http://www.mcs.anl.gov/mpi/mpich` or by anonymous `ftp` from `ftp.mcs.anl.gov` in the directory '`pub/mpi/mpich`', file '`mpich.tar.gz`'.
  - Links to other implementations are available at `www.mcs.anl.gov/mpi/implementations.html`.
- Test code repository:
  - `ftp://ftp.mcs.anl.gov/pub/mpi/mpi-test`

6 In Case of Trouble

This section describes some of the problems that you may run into and some solutions, along with information on submitting bug reports.

6.1 Things to try first

If something goes wrong, the first thing to do is to check the output of `configure` or `make` for some obvious problem, such as an improperly specified compiler or inadequate disk space. Next, check the web page `www.mcs.anl.gov/mpi/mpich/buglist-tbl.html` for any recent patches that may fix your problem. After that, check Section 6.3 for common problems. If you still can't find a solution to your problem, submit a bug report and we will try to help you.

6.2 Submitting bug reports

Send any problem that you can not solve by checking this section to `mpi-bugs@mcs.anl.gov`.

Please include:

- The version of `MPICH` (e.g., 1.2.6)
- The output of running your program with the `-mpiversion` argument (e.g., `mpirun -np 1 a.out -mpiversion`)
- The output of

  ```
  uname -a
  ```

  for your system. If you are on an SGI system, also

  ```
  hinv
  ```

- The files '`config.log`' and '`src/fortran/config.log`'.

- If the problem is with a script such as `configure` or `mpirun`, run the script with the `-echo` argument (e.g., `mpirun -echo -np 4 a.out`).

If you have more than one problem, please send them in separate messages; this simplifies our handling of problem reports.

The rest of this section contains some information on trouble-shooting MPICH. Some of these describe problems that are peculiar to some environments and give suggested work-arounds. Each section is organized in question and answer format, with questions that relate to more than one environment (workstation, operating system, etc.) first, followed by questions that are specific to a particular environment. Problems with workstation clusters are collected together as well. To make it easier to find solutions, the most common problems are described first.

6.3 The Most Common Problems

This section describes some of the most common problems encountered when building and using MPICH. See also Section A which covers frequently asked questions, including some additional problems.

Missing symbols when linking. The most common source of missing symbols is a failure of the MPICH configure step to determine how to pass command line arguments to Fortran. Check the output of the configure step for any error messages or warnings about building the Fortran libraries. If you do not require Fortran, reconfigure MPICH using the configure option `--disable-f77` and remake MPICH. If you need Fortran and cannot figure out how to make MPICH work with Fortran, send a bug report to `mpi-bugs@mcs.anl.gov`.

Another common problem with programs that mix Fortran and C is missing libraries. The MPICH configure attempts to determine the libraries that are necessary when linking C with Fortran, but may miss some. There are additional suggestions for this problem in Section 6.9.

`SIGSEGV`. Any message that mentions `SIGSEGV` is refering to a "segmentation violation" during program execution. This is usually due to an error in the user's program, such as an array overwrite or use of an uninitialized variable in referencing storage.

6.4 Troubleshooting Shared Libraries

Shared libraries provide a useful and powerful tool for speeding linking and execution of programs. However, the implementation of shared libraries in most operating systems leaves much to be desired. This section covers some of the problems and workarounds for them.

Some (most?) systems *do not remember* where the shared library was found when the executable was linked![4] Instead, they depend on finding the shared library in either a

[4] There is a reason for this unfortunate choice that has to do with distributing executables linked with shared libraries. However, the capability to save the location should be provided, at least as an option, by the linker.

default location (such as '/lib') or in a directory specified by an environment variable such as LD_LIBRARY_PATH or by a command line argument such as -R or -rpath (more on this below). The MPICH configure tests for this and will report whether an executable built with shared libraries remembers the location of the libraries. It also attemps to use a compiler command line argument to force the executable to remember the location of the shared library.

If you need to set an environment variable to indicate where the MPICH shared libraries are, you need to ensure that both the process that you run **mpirun** from and any processes that **mpirun** starts gets the enviroment variable. The easiest way to do this is to set the environment variable within your '.cshrc' (for csh or tcsh users) or '.profile' (for sh and ksh users) file.

However, setting the environment variable within your startup scripts can cause problems if you use several different systems. For example, you may have a single '.cshrc' file that you use with both an SGI (IRIX) and Solaris system. You do not want to set the LD_LIBRARY_PATH to point the SGI at the Solaris version of the MPICH shared libraries[5]. Instead, you would like to set the environment variable before running **mpirun**:

```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/usr/local/mpich/lib/shared
mpirun -np 4 cpi
```

Unfortunately, this won't always work. Depending on the method that **mpirun** and MPICH use to start the processes, the environment variable may not be sent to the new process. This will cause the program to fail with a message like

```
ld.so.1: /home/me/cpi: fatal: libpmpich.so.1.0: open failed: No such
file or directory
Killed
```

Some devices support starting new processes with the current environment; check the documentation for **mpirun** (Section 3.2).

An alternative to using LD_LIBRARY_PATH and the secure server is to add an option to the link command that provides the path to use in searching for shared libraries. Unfortunately, the option that you would like is "append this directory to the search path" (such as you get with -L). Instead, many compilers provide only "replace the search path with this path."[6] For example, some compilers allow -Rpath:path:...:path to specify a replacement path. Thus, if both MPICH and the user provide library search paths with -R, one of the search paths will be lost. Eventually, **mpicc** and friends can check for -R options and create a unified version, but they currently do not do this. You can, however, provide a complete search path yourself if your compiler supports an option such as -R.

The preceeding may sound like a lot of effort to go to, and in some ways it is. For large clusters, however, the effort will be worth it: programs will start faster and more reliably, because there is less network and file system traffic.

[5] You can make '.cshrc' check for the kind of system that you are running on and pick the paths appropriately. This isn't as flexible as the approach of setting the environment variable from the running shell.

[6] Even though the linker may provide the "append to search path" form.

6.5 Other Problems

The following items describe miscellaneous problems that we have encountered. These may help solve less common problems.

6.6 Problems configuring

6.6.1 General

1. **Q:** Configure reports that floating point is not commutative! How do I fix it?

   A: Check your compiler's documentation. On RS/6000's, the `-qnomaf` (no multiply-add floating point) option. On some other systems, intermediate results may be stored in 80-bit registers (Intel CPUs do this); this can also lead to inconsistent rounding. You may be able to force the compiler to round to 64 bits.

6.6.2 Linux

1. **Q:** The configure step issues the message:

   ```
   checking that the compiler f77 runs... no
   Fortran compiler returned non-zero return code
   Output from test was
   f2ctmp_conftest.f:
      MAIN main:
   ```

   A: This is probably caused by a problem in the Fortran compiler in older versions of Linux. The `f77` command in Linux was often a shell script that uses the `f2c` program to convert the Fortran program to C, and then compile it with the C compiler. In many versions of Linux, this script has an error that causes a non-zero return code even when the compilation is successful.

   To fix this problem, you need a corrected `f77` script. If you can edit the script yourself, change the last 3 lines from

   ```
   case $cOPT in 2) $CC $G -o $OUTF $OFILES -lf2c -lm;; esac
   rc=$?
   exit $rc
   ```

   to

   ```
   case $cOPT in 2) $CC $G -o $OUTF $OFILES -lf2c -lm;; esac
   rc=$?
   trap 0
   exit $rc
   ```

2. **Q:** The link test fails on Linux with messages like

```
overtake.o(.text+0x59): undefined reference to `MPI_COMM_WORLD'
overtake.o(.text+0x81): undefined reference to `MPI_COMM_WORLD'
...
```

A: This is probably caused by a problem in the Fortran compiler in Linux. In some early versions of Linux, the `f77` command in Linux is often a shell script that uses the `f2c` program to convert the Fortran program to C, and then compile it with the C compiler. In many versions of Linux, this script has an error that causes a non-zero return code even when the compilation is successful.

To fix this problem, you need a corrected `f77` script. If you can edit the script yourself, change the last 3 lines from

```
case $cOPT in 2) $CC $G -o $OUTF $OFILES -lf2c -lm;; esac
rc=$?
exit $rc
```

to

```
case $cOPT in 2) $CC $G -o $OUTF $OFILES -lf2c -lm;; esac
rc=$?
trap 0
exit $rc
```

3. **Q:** During the configure step, messages like

```
/homes/me/mpich/configure: 134956160: Permission denied
```

sometimes appear. What is wrong?

A: This is a bug in the Linux (actually GNU) `sh` shell. The shell is attempting to create a file with the name '`/tmp/t<processid>-sh`' (e.g., '`/tmp/t11203-sh`'), but was unable to. This may happen if '`/tmp`' is full; however, it can also happen when the shell created the same file for another user days before. (You can see this happen by running `configure` under `strace -f`). The only fix is to have your systems administrator clean old files out of '`/tmp`'.

6.7 Problems building MPICH

6.7.1 General

1. **Q:** When running make on MPICH, I get this error:

```
ar: write error: No such file or directory
*** Error code 1
```

I've looked, and all the files are accessible and have the proper permissions.

A: Check the amount of space in '`/tmp`'. This error is sometimes generated when there is insufficient space in '`/tmp`' to copy the archive (this is a step that `ar` takes when updating a library). The command `df /tmp` will show you how much space is available. Try to insure that at least twice the size of the library is available.

2. **Q:** When running make on `MPICH`, I get errors when executing `ranlib`.

   A: Many systems implement `ranlib` with the `ar` command, and they use the '`/tmp`' directory by default because it "seems" obvious that using '`/tmp`' would be faster ('`/tmp`' is often a local disk). Unfortunately, some systems have ridiculously small '`/tmp`' partitions, making any use of '`/tmp`' very risky. In some cases, the `ar` commands used by `MPICH` will succeed because they use the `l` option—this forces `ar` to use the local directory instead of '`/tmp`'. The `ranlib` command, on the other hand, may use '`/tmp`' and cannot be fixed.

   In some cases, you will find that the `ranlib` command is unnecessary. In these cases, you can reconfigure with `-noranlib`. If you must use `ranlib`, either reduce the space used in '`/tmp`' or increase the size of the '`/tmp`' partition (your system administrator will need to do this). There should be at least 20–30 MBytes free in '`/tmp`'.

3. **Q:** When doing the link test, the link fails and does not seem to find any of the MPI routines:

```
  /homes/me/mpich/IRIX32/ch_p4/bin/mpicc -o overtake overtake.o test.o
ld: WARNING 126: The archive /homes/me/mpich/IRIX32/ch_p4/lib/libmpi.a
  defines no global symbols. Ignoring.
ld: WARNING 84: /usr/lib/libsun.a is not used for resolving any symbol.
ld: ERROR 33: Unresolved data symbol "MPI_COMM_WORLD" -- 1st referenced
  by overtake.o.
ld: ERROR 33: Unresolved text symbol "MPI_Send" -- 1st referenced by
  overtake.o.
...
```

   A: Check that the `ar` and `ranlib` programs are compatible. One site installed the Gnu `ranlib` in such a way that it would be used with the vendors `ar` program, with which it was incompatible. Use the `-noranlib` option to `configure` if this is the case.

6.7.2 Workstation Networks

1. **Q:** When building `MPICH`, the make fails with errors like this:

```
making p4 in directory lib
        make  libp4.a
        cc -Aa -g -I../include -I../../../../include   -c p4_globals.c
cc: "/usr/include/netinet/in.h", line 69: error 1000: Unexpected symbol: "u_long".
cc: "/usr/include/netinet/in.h", line 127: error 1000: Unexpected symbol: "u_short".
```

   etc.

   A: The problem here is that the system (not `MPICH`) include files are incompatible with your choice of compiler. Check to see if `cc` is aliased (in C shell, do `alias cc`). If it is, either unalias it or set the environment variable `CC` to the full path for the compiler. To get the full path, do

```
unalias cc
setenv CC `which cc`
```

   and then reconfigure.

6.7.3 Cray T3D

1. **Q:** When linking I get

   ```
   mppldr-133 cf77: CAUTION
       Unsatisfied external references have been encountered.

   Unsatisfied external references
   Entry name      Modules referencing entry

   GETARG (equivalenced to $USX1)
                   MPIR_GETARG
   ```

   A: You may have specified the Fortran compiler with the `F77` environment variable or the `-fc` argument to configure. The `MPICH` Fortran implemenation of MPI uses a common Fortran extension, `GETARG`, to get the command line arguments. Most Fortran runtime systems support this, but Cray uses `call pxfgetarg(i,s,len(s),ierr)` instead. You can change the file '`src/env/farg.f`' manually to call the correct routine (but note that configure builds a new '`farg.f`' from '`farg.f.in`' each time that it is run).

   `MPICH` now attempts to determine the correct names of the routines to access the command line. If you find that `MPICH` fails to determine the names correctly, please send a bug report to `mpi-bugs@mcs.anl.gov`.

6.7.4 SGI

1. **Q:** The build on an SGI Power Challenge fails with

   ```
   Signal: SIGSEGV in Back End Driver phase.
   > ### Error:
   > ### Signal SIGSEGV in phase Back End Driver -- processing aborted
   > f77 ERROR:  /usr/lib64/cmplrs/be died due to signal 4
   > f77 ERROR:  core dumped
   > *** Error code 2 (bu21)
   > *** Error code 1 (bu21)
   > *** Error code 1 (bu21)
   ```

   A: Our information is that setting the environment variable `SGI_CC` to `-ansi` will fix this problem.

2. **Q:** The build on an SGI with architecture `IRIXN32` fails with

   ```
   cc: Warning: -c should not be used with ucode -O3 -o32
                on a single file; use -j instead to get inter-module
                optimization.
   ```

   A: Amazingly, the standard `-c` option is *not valid* for the SGI compilers when both `-O3` and `-n32` are specified. This is a "feature" of the SGI compiler, and there is no way to work around this for `MPICH` (other than a massive and non-portable rewrite of all the '`Makefile`'s). Your only option is to not use the `-O3` option.

6.7.5 Linux

1. **Q:** The link test failed on Linux with

   ```
   ...
   cc  -o overtake overtake.o test.o -L/usr/local/mpich/LINUX/ch_p4/lib
   -lmpi
   overtake.o(.text+0x71): undefined reference to `MPI_COMM_WORLD'
   overtake.o(.text+0x82): undefined reference to `MPIR_I_DOUBLE'
   overtake.o(.text+0xe1): undefined reference to `MPI_COMM_WORLD'
   ...
   ```

   A: We have been informed that there is a error in the `f77` script in some versions of Linux which causes this problem. Try either getting a patch for the `f77` script or reconfiguring with `-nof77`.

2. **Q:** The build fails for the `ch_p4` device when using the Compaq C compiler.

   A: There is an incompatibility with the system include files (not the MPICH include files). If you can modify '`/usr/include/rpc/xdr.h`', add the following near the top of that file:

   ```
       #if defined(__DECC) || defined(__DECCXX)
       typedef long int int64_t;
       #endif
   ```

6.7.6 Compaq ULTRIX and Tru64

1. **Q:** When trying to build, the `make` aborts early during the cleaning phase:

   ```
   amon:MPICH/mpich>make clean
           /bin/rm -f *.o *~ nupshot
   *** Error code 1
   ```

   A: This is a bug in the shell support on some Compaq ULTRIX systems. You may be able to work around this with

   ```
   setenv PROG_ENV SYSTEM_FIVE
   ```

   Configuring with `-make=s5make` may also work.

6.8 Problems in testing

The MPICH test suite, in '`examples/test`', performs a fairly complete test of an MPI implementation. If there is an error, it usually indicates a problem with the implementation of MPI; if you encounter such a problem, please report it to `mpi-bugs@mcs.anl.gov`. However, there are a few exceptions that are described here.

6.8.1 General

1. **Q:** The test `pt2pt/structf` fails with

   ```
   0 - Error in MPI_ADDRESS : Invalid argument: Address of location
   given to MPI_ADDRESS does not fit in Fortran integer
   [0] Aborting program!
   ```

   A: This is not an error; it is a gap in the MPI-1 definition that is fixed in MPI-2 (with the routine `MPI_Get_address`. This indicates that Fortran integers are not large enough to hold an address. This does indicate that MPI programs written in Fortran should not use the `MPI_Address` function on this system.

2. **Q:** The test `env/timers` fails with

   ```
   Timer around sleep(1) did not give 1 second; gave 0.399949
   ```

   A: The low-level software that `MPICH` uses probably makes use of the `SIGALRM` signal, thus denying it to the user's program. This is not an error (the standard permits systems to make use of any signals) though it is unfortunate.

6.9 Problems compiling or linking Fortran programs

6.9.1 General

1. **Q:** When linking the test program, the following message is generated:

   ```
   f77 -g -o secondf secondf.o -L/usr/local/mpich/lib -lmpich
   invalid option -L/usr/local/mpich/lib
   ld: -lmpich: No such file or directory
   ```

   A: This `f77` program does not accept the `-L` command to set the library search path. Some systems provide a shell script for `f77` that is very limited in its abilities. To work around this, use the full library path instead of the `-L` option:

   ```
   f77 -g -o secondf secondf.o /usr/local/mpich/lib/libmpich.a
   ```

   As of the `MPICH` 1.2.0 release, the `MPICH` configure attempts to find the correct option for indicating library paths to the Fortran compiler. If you find that the `MPICH` configure has made an error, please submit a bug report to `mpi-bugs@mcs.anl.gov`.

2. **Q:** When linking Fortran programs, I get undefined symbols such as

   ```
   f77  -c secondf.f
   secondf.f:
    MAIN main:
   f77  -o secondf secondf.o -L/home/mpich/lib/solaris/ch_shmem -lmpich
   Undefined                       first referenced
    symbol                              in file
   ```

```
getdomainname
/home/mpich/lib/solaris/ch_shmem/libmpi .a(shmempriv.o)
ld: fatal: Symbol referencing errors. No output written to secondf
```

There is no problem with C programs.

A: This means that your C compiler is providing libraries for you that your Fortran compiler is not providing. Find the option for the C compiler and for the Fortran compilers that indicate which library files are being used (alternately, you may find an option such as `-dryrun` that shows what commands are being used by the compiler). Build a simple C and Fortran program and compare the libraries used (usually on the `ld` command line). Try the ones that are present for the C compiler and missing for the Fortran compiler. `MPICH` attempts to determine these libraries for you. If you have trouble, send a bug report and include the contents of the file '`mpich/src/fortran/config.log`'.

3. **Q:** When trying to compile Fortran code with a Fortran 90 or Fortran 95 compiler, I get error messages like

   ```
   Error: foo.f, line 30: Inconsistent datatype for argument 1 in MPI_SEND
   ```

   A: The Fortran language requires that in two calls to the same subroutine, the types of arguments must be the same. That is, if you called `MPI_SEND` with a `REAL` buffer as the first argument and then called it with an `INTEGER` buffer as the first argument, a Fortran compiler can consider this an error. Few Fortran 77 compilers would complain about this; more Fortran 90 and Fortran 95 compilers check for this. There are two solutions. One is to use the MPI module (in the "choice" version: use the `-choicemod` option for `mpif90`); the other is to use an option to tell the Fortran 90 compiler to allow argument mismatches. For example, the argument `-mismatch` will cause the NAG Fortran compilers to allow mismatched arguments. Using the MPI module is the preferred approach.

   Fortran 77 users may sometimes see a similar message, particularly with later versions of `g77`. The option `-Wno-globals` will suppress these warning messages.

6.10 Problems Linking C Programs

6.10.1 General

1. **Q:** When linking programs, I get messages about `__builtin_saveregs` being undefined.

   A: You may have a system on which C and Fortran compilers are incompatible (for example, using `gcc` and the vendor's Fortran compiler). If you do not plan to use Fortran, the easiest fix is to rebuild with the `-nof77` option to configure.

   You should also look into making your C compiler compatible with your Fortran compiler. Because this is very dependent on the specific system and compilers that you are using, you may need to find a local expert who can help you.

The easiest but ugliest possibility is use `f2c` to convert Fortran to C, then use the C compiler to compile everything. If you take this route, remember that *every* Fortran routine has to be compiled using `f2c` and the C compiler.

Alternatively, you can use various options (check the man pages for your compilers) to see what libraries that add when they link. Add those libraries to the link line for the *other* compiler. If you find a workable set of libraries, edit the appropriate scripts (e.g., `mpicc`) to include the necessary libraries. `MPICH` attempts to find all the libaries that you need but is not always successful.

6.10.2 HPUX

1. **Q:** When linking on HPUX, I get an error like this:

   ```
   cc -o pgm pgm.o -L/usr/local/mpich/lib/hpux/ch_p4 -lmpich  -lm
   /bin/ld: Unsatisfied symbols:
   sigrelse (code)
   sigset (code)
   sighold (code)
   *** Error code 1
   ```

   A: You need to add the link option `-lV3`. The `ch_p4` device uses the System V signals on the HP; these are provided in the '`V3`' library.

6.10.3 LINUX

1. **Q:** When linking a Fortran program, I get

   ```
   Linking:
   foo.o(.data+0x0): undefined reference to 'pmpi_wtime_'
   ```

   A: This is a bug in the `pgf77` compiler (which is itself a workaround for a bug in the LINUX `ld` command). You can fix it by either adding `-lpmpich` to the link line or modifying the '`mpif.h`' to remove the `external pmpi_wtime, pmpi_wtick` statement.

   The `MPICH` configure attempts to determine if `pmpi_wtime` and `pmpi_wtick` can be declared in '`mpif.h`' and removes them if there is a problem. If this happens and you use `pmpi_wtime` or `pmpi_wtick` in your program, you will need to declare them as functions returning `double precision` values.

6.11 Problems starting programs

6.11.1 General

1. **Q:** When trying to start a program with

   ```
   mpirun -np 2 cpi
   ```

either I get an error message or the program hangs.

A: On some systems such as IBM SPs, there are many mutually exclusive ways to run parallel programs; each site can pick the approach(es) that it allows. The script `mpirun` tries one of the more common methods, but may make the wrong choice. Use the `-v` or `-t` option to `mpirun` to see how it is trying to run the program, and then compare this with the site-specific instructions for using your system. You may need to adapt the code in `mpirun` to meet your needs. See also the next question.

2. **Q:** When trying to run a program with, e.g., `mpirun -np 4 cpi`, I get

```
usage : mpirun [options] <executable> [<dstnodes>] [-- <args>]
```

or

```
mpirun [options] <schema>
```

A: You have a command named `mpirun` for a different implementation of MPI in your path ahead of the `MPICH` version. Execute the command

```
which mpirun
```

to see which command named `mpirun` was actually found. The fix is to either change the order of directories in your path to put the `MPICH` version of `mpirun` first, or to define an alias for `mpirun` that uses an absolute path. For example, in the csh shell, you might do

```
alias mpirun /usr/local/mpich/bin/mpirun
```

to set `mpirun` to the `MPICH` version.

3. **Q:** When I try to run a program on more than one processor, I get an error message

```
mpirun -np 2 cpi
/home/me/cpi: error in loading shared libraries: libcxa.so.1:
cannot open shared object file: No such file or directory
```

There is no trouble running on one processor.

A: This means that some shared library used by the system cannot be found on remote processors. There are two possibilities:

(a) The shared libraries are not installed on the remote processor. To fix this, have your system administrators install the libraries.

(b) The shared libraries are not in the default path. This can happen if the path is in an environment variable that is set in your current shell but that is not part of your default or remote shell environment. The fix in this case is harder, because you must communicate the location of the shared library to the executable (a major deficiency in most Unix shared-library designs is that executables do not, by default, remember where a shared library was found when linking). The simplest fix may be to have your system administrators place the necessary shared libraries into one of the directories that is searched by default. If this is not possible, then you will need to help the compiler and linker out.

Many linkers provide a way to specify the search path for shared libraries. The trick is to (a) pass this command to the linker program and (b) specify all of the libraries that are needed.

For example, on Linux systems, the linker command to specify the shared library search path is `-rpath path`, e.g., `-rpath /usr/lib:/usr/local/lib`. To pass this command to the linker through the Intel C compiler `icc`, the command `-Qoption,link,-rpath,path` is used. By default, the Linux linker looks in '`/usr/lib`' and the directories specified by the environment variable `LD_LIBRARY_PATH`. Thus, to force the linker to include the path to the shared library, you can use

```
mpicc -o cpi cpi -Qoption,link,-rpath,$LD_LIBRARY_PATH:/usr/lib
```

If this works, then consider editing the value of `LDFLAGS` in the compiler scripts (e.g., `mpicc`) to include this option.

Unfortunately, each compiler has a different way of passing these arguments to the linker, and each linker has a different set of arguments for specifying the shared library search path. You will need to check the documentation for your system to find this options.

4. **Q:** When attempting to run `cpilog` I get the following message:

```
ld.so.1: cpilog: fatal: libX11.so.4: can't open file: errno 2
```

A: The X11 version that `configure` found isn't properly installed. This is a common problem with Sun/Solaris systems. One possibility is that your Solaris machines are running slightly different versions. You can try forcing static linking (-Bstatic on Solaris).

Alternately, consider adding these lines to your '`.login`' (assuming C shell):

```
setenv OPENWINHOME /usr/openwin
setenv LD_LIBRARY_PATH /opt/SUNWspro/lib:/usr/openwin/lib
```

(you may want to check with your system administrator first to make sure that the paths are correct for your system). Make sure that you add them *before* any line like

```
if ($?USER == 0 || $?prompt == 0) exit
```

5. **Q:** My program fails when it tries to write to a file.

A: If you opened the file *before* calling `MPI_INIT`, the behavior of MPI (not just the `MPICH` implementation of MPI) is undefined. In the `ch_p4` device, only process zero (in `MPI_COMM_WORLD`) will have the file open; the other processes will not have opened the file. Move the operations that open files and interact with the outside world to after `MPI_INIT` (and before `MPI_FINALIZE`).

6. **Q:** Programs seem to take forever to start.

   A: This can be caused by any of several problems. On systems with dynamically-linked executables, this can be caused by problems with the file system suddenly getting requests from many processors for the dynamically-linked parts of the executable (this has been measured as a problem with some DFS implementations). You can try statically linking your application.

6.11.2 IBM RS6000

1. **Q:** When trying to run on an IBM RS6000 with the `ch_p4` device, I got

```
% mpirun -np 2 cpi
Could not load program /home/me/mpich/examples/basic/cpi
Could not load library libC.a[shr.o]
Error was: No such file or directory
```

   A: This means that `MPICH` was built with the `xlC` compiler but that some of the machines in your '`util/machines/machines.rs6000`' file do not have `xlC` installed. Either install `xlC` or rebuild `MPICH` to use another compiler (either `xlc` or `gcc`; `gcc` has the advantage of never having any licensing restrictions).

6.11.3 IBM SP

1. **Q:** When starting my program on an IBM SP, I get this:

```
$ mpirun -np 2 hello
ERROR: 0031-124  Couldn't allocate nodes for parallel execution.  Exiting ...
ERROR: 0031-603  Resource Manager allocation for task: 0, node:
me1.myuniv
.edu, rc = JM_PARTIONCREATIONFAILURE
ERROR: 0031-635  Non-zero status -1 returned from pm_mgr_init
```

   A: This means that either `mpirun` is trying to start jobs on your SP in a way different than your installation supports or that there has been a failure in the IBM software that manages the parallel jobs (all of these error messages are from the IBM `poe` command that `mpirun` uses to start the MPI job). Contact your system administrator for help in fixing this situation. You system administrator can use

```
dsh -av "ps aux | egrep -i 'poe|pmd|jmd'"
```

   from the control workstation to search for stray IBM POE jobs that can cause this behavior. The files `/tmp/jmd_err` on the individual nodes may also contain useful diagnostic information.

2. **Q:** When trying to run on an IBM SP, I get the message from `mpirun`:

```
ERROR: 0031-214  pmd: chdir </a/user/gamma/home/mpich/examples/basic>
ERROR: 0031-214  pmd: chdir </a/user/gamma/home/mpich/examples/basic>
```

A: These are messages from tbe IBM system, not from `mpirun`. They may be caused by an incompatibility between POE, the automounter (especially the AMD automounter) and the shell, especially if you are using a shell other than `ksh`. There is no good solution; IBM often recommends changing your shell to `ksh`!

3. **Q:** When trying to run on an IBM SP, I get this message:

   ```
   ERROR: 0031-124  Less than 2 nodes available from pool 0
   ```

   A: This means that the IBM POE/MPL system could not allocate the requested nodes when you tried to run your program; most likely, someone else was using the system. You can try to use the environment variables `MP_RETRY` and `MP_RETRYCOUNT` to cause the job to wait until the nodes become available. Use `man poe` to get more information.

4. **Q:** When running on an IBM SP, my job generates the message

   ```
   Message number 0031-254 not found in Message Catalog.
   ```

   and then dies.

   A: If your user name is eight characters long, you may be experiencing a bug in the IBM POE environment. The only fix at the time this was written was to use an account whose user name was seven characters or less. Ask your IBM representative about PMR 4017X (poe with userids of length eight fails) and the associated APAR IX56566.

6.12 Programs fail at startup

6.12.1 General

1. **Q:** With some systems, you might see

   ```
   /lib/dld.sl: Bind-on-reference call failed
   /lib/dld.sl: Invalid argument
   ```

   (This example is from HP-UX), or

   ```
   ld.so: libc.so.2: not found
   ```

   (This example is from SunOS 4.1; similar things happen on other systems).

   A: The problem here is that your program is using shared libraries, and the libraries are not available on some of the machines that you are running on. To fix this, relink your program without the shared libraries. To do this, add the appropriate command-line options to the link step. For example, for the HP system that produced the errors above, the fix is to use `-Wl,-Bimmediate` to the link step. For Solaris, the appropriate option is `-Bstatic`.

6.12.2 Workstation Networks

1. **Q:** I can run programs using a small number of processes, but once I ask for more than 4–8 processes, I do not get output from all of my processes, and the programs never finish.

   A: We have seen this problem with installations using AFS. The remote shell program, `rsh`, supplied with some AFS systems limits the number of jobs that can use standard output. This seems to prevent some of the processes from exiting as well, causing the job to hang. There are four possible fixes:

   (a) Use a different `rsh` command. You can probably do this by putting the directory containing the non-AFS version first in your `PATH`. This option may not be available to you, depending on your system. At one site, the non-AFS version was in '`/bin/rsh`'.

   (b) Use the secure server (`serv_p4`). See the discussion in the Users Guide.

   (c) Redirect all standard output to a file. The MPE routine `MPE_IO_Stdout_to_file` may be used to do this.

   (d) Get a fixed `rsh` command. The likely source of the problem is an incorrect usage of the `select` system call in the `rsh` command. If the code is doing something like

   ```
   int mask;
   mask |= 1 << fd;
   select( fd+1, &mask, ... );
   ```

   instead of

   ```
   fd_set mask;
   FD_SET(fd,&mask);
   select( fd+1, &mask, ... );
   ```

   then the code is incorrect (the `select` call changed to allow more than 32 file descriptors many years ago, and the `rsh` program (or programmer!) hasn't changed with the times).

   A fourth possiblity is to get an AFS version of `rsh` that fixes this bug. As we are not running AFS ourselves, we do not know whether such a fix is available.

2. **Q:** Not all processes start.

   A: This can happen when using the `ch_p4` device and a system that has extremely small limits on the number of remote shells you can have. Some systems using "Kerberos" (a network security package) allow only three or four remote shells; on these systems, the size of `MPI_COMM_WORLD` will be limited to the same number (plus one if you are using the local host).

   The only way around this is to try the secure server; this is documented in the `MPICH` installation guide. Note that you will have to start the servers "by hand" since the `chp4_servs` script uses remote shell to start the servers.

6.13 Programs fail after starting

6.13.1 General

1. **Q:** I use `MPI_Allreduce`, and I get different answers depending on the number of processes I'm using.

   A: The MPI collective routines may make use of associativity to achieve better parallelism. For example, an

   ```
   MPI_Allreduce( &in, &out, MPI_DOUBLE, 1, ... );
   ```

   might compute
   $$(((((((a + b) + c) + d) + e) + f) + g) + h)$$
   or it might compute
   $$((a + b) + (c + d)) + ((e + f) + (g + h)),$$
   where $a, b, \ldots$ are the values of `in` on each of eight processes. These expressions are equivalent for integers, reals, and other familar objects from mathematics but are *not* equivalent for datatypes, such as floating point, used in computers. The association that MPI uses will depend on the number of processes, thus, you may not get exactly the same result when you use different numbers of processes. Note that you are not getting a wrong result, just a different one (most programs assume the arithmetic operations are associative). All processes do get the same result from a single call to `MPI_Allreduce`.

2. **Q:** My Fortran program fails with a BUS error.

   A: The C compiler that `MPICH` was built with and the Fortran compiler that you are using have different alignment rules for variables of type like `DOUBLE PRECISION`. For example, the GNU C compiler `gcc` may assume that all `double`s are aligned on eight-byte boundaries, but the Fortran language requires only that `DOUBLE PRECISION` align with `INTEGER`s, which may be four-byte aligned.

   There is no good fix. Consider rebuilding `MPICH` with a C compiler that supports weaker data alignment rules. Some Fortran compilers will allow you to force eight-byte alignment for `DOUBLE PRECISION` (for example, `-dalign` or `-f` on some Sun Fortran compilers); note though that this may break some correct Fortran programs that exploit Fortran's storage association rules.

   Some versions of `gcc` may support `-munaligned-doubles`; `MPICH` should be rebuilt with this option if you are using `gcc`, version 2.7 or later. `MPICH` attemps to detect and use this option where available.

3. **Q:** I'm using `fork` to create a new process, or I'm creating a new thread, and my code fails.

   A: The `MPICH` implementation is not thread safe and does not support either fork or the creation of new processes. Note that the MPI specification is thread safe, but implementations are not required to be thread safe. At this writing, few implementations are thread-safe, primarily because this reduces the performance of the MPI implementation (you at least need to check to see if you need a thread lock, actually getting and releasing the lock is even more expensive).

The MPICH implementation supports the MPI_Init_thread call; with this call, new in MPI-2, you can find out what level of thread support the MPI implementation supports. As of version 1.2.0 of MPICH, only MPI_THREAD_SINGLE is supported. We believe that version 1.2.0 and later support MPI_THREAD_FUNNELED, and some users have used MPICH in this mode (particularly with OpenMP), but we have not rigourously tested MPICH for this mode. Future versions of MPICH will support MPI_THREAD_MULTIPLE.

Q: C++ programs execute global destructors (or constructors) more times than expected. For example:

```
class Z {
public:
  Z()  { cerr << "*Z" << endl; }
  ~Z() { cerr << "+Z" << endl; }
};

Z z;

int main(int argc, char **argv) {
  MPI_Init(&argc, &argv);
  MPI_Finalize();
}
```

when running with the ch_p4 device on two processes executes the destructor twice for each process.

A: The number of processes running before MPI_Init or after MPI_Finalize is not defined by the MPI standard; you can not rely on any specific behavior. In the ch_p4 case, a new process is forked to handle connection requests; it terminates with the end of the program.

You can use the threaded listener with the ch_p4 device, or use the ch_p4mpd device instead. Note, however, that this code is not portable because it relies on behavior that the MPI standard does not specify.

6.13.2 HPUX

1. **Q:** My Fortran programs seem to fail with SIGSEGV when running on HP workstations.

   A: Try compiling and linking the Fortran programs with the option +T. This *may* be necessary to make the Fortran environment correctly handle interrupts used by MPICH to create connections to other processes.

6.13.3 ch_shmem device

1. **Q:** My program sometimes hangs when using the ch_shmem device.

A: Make sure that you are linking with *all* of the correct libraries. If you are not using `mpicc`, try using `mpicc` to link your application. The reason for this is that correct operation of the shared-memory version may depend on additional, system-provided libraries. For example, under Solaris, the thread library must used, otherwise, non-functional versions of the mutex routines critical to the correct functioning of the MPI implementation are taken from '`libc`' instead.

6.13.4 LINUX

1. **Q:** Processes fail with messages like

   ```
   p0_1835:  p4_error: Found a dead connection while looking for messages: 1
   ```

   A: What is happening is that the TCP implementation on this platform is deciding that the connection has "failed" when it really hasn't. The current MPICH implementation assumes that the TCP implementation will not close connections and has no code to reanimate failed connections. Future versions of MPICH will work around this problem.

   In addition, some users have found that the single processor Linux kernel is more stable than the SMP kernel.

6.14 Trouble with Input and Output

6.14.1 General

1. **Q:** I want output from `printf` to appear immediately.

   A: This is really a feature of your C and/or Fortran runtime system. For C, consider

   ```
   setbuf( stdout, (char *)0 );
   ```

   or

   ```
   setvbuf( stdout, NULL, _IONBF, 0 );
   ```

6.14.2 Workstation Networks

1. **Q:** I want standard output (`stdout`) from each process to go to a different file.

   A: MPICH has no built-in way to do this. In fact, it prides itself on gathering the stdouts for you. You can do one of the following:

   (a) Use Unix built-in commands for redirecting `stdout` from inside your program (`dup2`, etc.). The MPE routine `MPE_IO_Stdout_to_file`, in '`mpe/src/mpe_io.c`', shows one way to do this. Note that in Fortran, the approach of using `dup2` will work only if the Fortran `PRINT` writes to `stdout`. This is common but by no means universal.

(b) Write explicitly to files instead of to `stdout` (use `fprintf` instead of `printf`, etc.). You can create the file name from the process's rank. This is the most portable way.

6.14.3 HP-UX

1. **Q:** When trying to run `upshot` under HP-UX, I get error messages like

```
set: Variable name must begin with a letter.
```

or

```
upshot: syntax error at line 35: `(' unexpected
```

A: Your version of HP-UX limits the shell names to very short strings. `Upshot` is a program that is executed by the `wish` shell, and for some reason HP-UX is both refusing to execute in this shell and then trying to execute the `upshot` program using your current shell (e.g., '`sh`' or '`csh`'), instead of issuing a sensible error message about the command name being too long. There are two possible fixes:

(a) Add a link with a much shorter name, for example

```
ln -s /usr/local/tk3.6/bin/wish /usr/local/bin/wish
```

Then edit the `upshot` script to use this shorter name instead. This may require root access, depending on where you put the link.

(b) Create a regular shell program containing the lines

```
#! /bin/sh
/usr/local/tk3.6/bin/wish -f /usr/local/mpi/bin/upshot
```

(with the appropriate names for both the '`wish`' and '`upshot`' executables).

Also, file a bug report with HP. At the very least, the error message here is wrong; also, there is no reason to restrict general shell choices (as opposed to login shells).

6.15 Special debugging arguments

These are currently undocumented, and some require configure options to have been specified (like `-mpipktsize` and `-chmemdebug`). The option `-mpiversion` is useful for finding out how your installation of `MPICH` was configured and exactly what version it is. The options include:

-mpedbg If an error occurs, start `xterms` attached to the process that generated the error. Requires the `MPICH` be configured with `-mpedbg` and works on only some workstations systems.

-mpiversion Print out the version and configuration arguements for the `MPICH` implementation being used.

These arguments are provided to the program, not to `mpirun`. For example,

```
mpirun -np 2 a.out -mpiversion
```

Acknowledgments

The work described in this report has benefited from conversations with and use by a large number of people. We also thank those that have helped in the implementation of MPICH, particularly Patrick Bridges and Edward Karrels. Particular thanks goes to Nathan Doss and Anthony Skjellum for valuable help in the implementation and development of MPICH. Debbie Swider, who worked with the `MPICH` group for several years, helped support and enrich the `MPICH` implementation. More recent extensive contributions have been made by Omer Zaki (for `Jumpshot`). Anthony Chan has helped with SLOG and `Jumpshot-3`. David Ashton, who has developed the Windows NT version of `MPICH` and was supported by a grant from the Microsoft corporation, has also contributed to `MPICH`. The C++ bindings have been contributed by Andrew Lumsdaine and Jeff Squyres of Notre Dame. Rajeev Thakur of Argonne is responsible for the ROMIO implementation of the MPI-2 I/O functions. The Globus2 device is due to Nick Karonis of Northern Illinois University and Brian Toonen of Argonne National Laboratory.

Appendices

A Frequently Asked Questions

- Introduction
- Installing MPICH
- Using MPICH
- Permission Denied
- Notes on getting MPICH running Under Linux
- poll: protocol failure during circuit creation
- Using SSH
- Mac OS X and hostname
- SIGSEGV
- semop lock failed
- Compiler Switches
- C++ Builds Fail
- Fortran programs give errors about mismatched types

- Missing Symbols When Linking
- Warning messages while building MPICH
- MPMD (Multiple Program Multiple Data) Programs
- Reporting problems and support
- Algorithms used in MPICH
- Jumpshot and X11

A.1 Introduction

MPICH is a freely available, portable implementation of MPI, the Standard for message-passing libraries.

A.2 Installing MPICH

Building and installing MPICH often requires only

```
./configure --prefix=/home/me/mpich
make
make install
```

where the value of the `--prefix` argument to configure is the directory in which MPICH should be installed. See the *Installation Guide* for more detailed instructions.

A.3 Using MPICH

Information on using MPICH can be found in the Installation and Users' Guide. There are versions of this guide for each MPICH device; make sure that you read the one the applies to your device. You can find out which device is installed by running a simple MPI program with the option -mpiversion:

```
mpirun -np 1 a.out -mpiversion
```

All of the guides are available at `http://www.mcs.anl.gov/mpi/mpich/docs.html` .

A.4 Permission Denied

Question:
When I use mpirun, I get the message `Permission denied`, `connection reset by peer`, or `poll: protocol failure in circuit setup` when trying to run `MPICH`.

Answer:
If you see something like this

```
% mpirun -np 2 cpi
Permission denied.
```

(or `connection reset by peer` or `poll: protocol failure in circuit setup`) when using the `ch_p4` device, it probably means that you do not have permission to use rsh to start processes. The script tstmachines can be used to test this. For example, if the architecture type (the -arch argument to configure) is sun4, then try

```
tstmachines sun4
```

If this fails, then you may need a '`.rhosts`' or '`/etc/hosts.equiv`' file (you may need to see your system administrator) or you may need to use the p4 server. Another possible problem is the choice of the remote shell program; some systems have several. Check with your systems administrator about which version of rsh or remsh you should be using.

If your system allows a '`.rhosts`' file, do the following:

- Create a file .rhosts in your home directory
- Change the protection on it to user read/write only: chmod og-rwx .rhosts.
- Add one line to the .rhosts file for each processor that you want to use. The format is

```
host username
```

For example, if your username is doe and you want to user machines a.our.org and b.our.org, your .rhosts file should contain

```
a.our.org doe
b.our.org doe
```

Note the use of fully qualified host names (some systems require this).

On networks where the use of .rhosts files is not allowed, (such as the one in MCS at Argonne), you should use the p4 server to run on machines that are not trusted by the machine that you are initiating the job from.

Finally, you may need to use a non-standard rsh command within `MPICH`. `MPICH` must be reconfigured with `-rsh=command_name`, and perhaps also with `-rshnol` if the remote shell command does not support the `-l` argument. Systems using Kerberos and/or AFS may need this.

A.5 Notes on getting MPICH running Under Linux

Introduction The purpose of this document is to describe the steps necessary to allow MPICH processes to be started and to communicate with one another. The installation that we are focusing on is a RedHat 7.2 installation with medium security (the default).

While other distributions will certainly vary, this is a good example of the sorts of problems that one might run across.

There are three methods for starting MPICH processes that are typically used on clusters today. These are rsh, ssh, and mpd.

We will first describe getting the rsh service working. We will include rlogin in this process because it is helpful for testing. Next we will describe getting ssh working and enabling the ssh-agent to allow for logins without password typing. Finally we will discuss issues related to process communication in MPICH and firewalls.

Enabling rsh By default the rsh server is not installed, and it is necessary for use of the rsh service in starting MPICH processes. The rsh server, in.rshd, is part of the rsh-server RPM. This RPM is located on the first disc of the RedHat 7.2 distribution. The rlogin server, in.rlogind, is also included in this package.

The xinetd server controls the availability of the rsh and rlogin services. This server is installed by default, but by default rsh and rlogin services are disabled. To enable these services, you must edit the files '`/etc/xinetd.d/rsh`' and '`/etc/xinetd.d/rlogin`'. Here is the rsh file as it looks by default:

```
# default: on
# description: The rshd server is the server for the rcmd(3) routine and, \
#       consequently, for the rsh(1) program.  The server provides \
#       remote execution facilities with authentication based on \
#       privileged port numbers from trusted hosts.
service shell
{
        socket_type             = stream
        wait                    = no
        user                    = root
        log_on_success          += USERID
        log_on_failure          += USERID
        server                  = /usr/sbin/in.rshd
        disable                 = yes
}
```

You must enable the service by changing "disable = yes" to "disable = no". The same must be done to the rlogin config file to enable that service.

At this point the xinetd daemon must be restarted to register these changes:

```
/etc/rc.d/init.d/xinetd restart
```

At this point you should receive a "Permission denied." if you attempt a command such as "rsh localhost hostname" as a non-root user (or as root for that matter).

To allow users to rsh without passwords you need to edit '`/etc/hosts.equiv`', the system-wide host file for rsh and rlogin. This file should hold hostnames of machines that you would like users to be able to start MPICH processes from. For example, simply adding:

```
localhost.localdomain
```

Should allow users to perform the command "rsh localhost hostname" successfully. Likewise adding other hostnames will allow users on those hosts to rsh to this host.

However, there is another catch! By default (with medium security) packet filtering is enabled as well, and this will prevent users from remote hosts from connecting to this machine using the rsh or rlogin services. This packet filter, or firewall, is administered using the ipchains package (which is installed by default).

The firewall configuration is written out by a program called lokkit at installation time (I think). The configuration is stored in '`/etc/sysconfig/ipchains`' and by default looks like this:

```
# Firewall configuration written by lokkit
# Manual customization of this file is not recommended.
# Note: ifup-post will punch the current nameservers through the
#       firewall; such entries will *not* be listed here.
:input ACCEPT
:forward ACCEPT
:output ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth0 -j ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth1 -j ACCEPT
-A input -s 0/0 -d 0/0 -i lo -j ACCEPT
-A input -p tcp -s 0/0 -d 0/0 0:1023 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 2049 -y -j REJECT
-A input -p udp -s 0/0 -d 0/0 0:1023 -j REJECT
-A input -p udp -s 0/0 -d 0/0 2049 -j REJECT
-A input -p tcp -s 0/0 -d 0/0 6000:6009 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 7100 -y -j REJECT
```

While an in-depth discussion of ipchains rules is outside the context of this document, it's worth talking about how this works a bit. First, the rules are applied in order from top of the list to the bottom of the list. The argument to -j says what to do if a packet matches; it's usually either ACCEPT (let the packet in), or REJECT (toss it out). If a packet makes it through the entire list then the default policy is applied. In this case the default policy is ACCEPT.

The following line tells the packet filter to allow all localhost (-i lo) traffic to pass unmolested:

```
-A input -s 0/0 -d 0/0 -i lo -j ACCEPT
```

This line blocks all new TCP connections going to ports 0-1023, which is the range of most services, including rsh/rlogin:

```
-A input -p tcp -s 0/0 -d 0/0 0:1023 -y -j REJECT
```

We're going to modify this file to allow rsh and rlogin traffic.

```
# Firewall configuration written by lokkit
# Manual customization of this file is not recommended.
# Note: ifup-post will punch the current nameservers through the
#       firewall; such entries will *not* be listed here.
:input ACCEPT
:forward ACCEPT
:output ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth0 -j ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth1 -j ACCEPT
-A input -s 0/0 -d 0/0 -i lo -j ACCEPT
#
# New rules for rlogin/rsh traffic, incoming or outgoing
#
-A input -p tcp -s 0/0 -d 0/0 513 -b -j ACCEPT
-A input -p tcp -s 0/0 -d 0/0 514 -b -j ACCEPT
#
# End of new rules
#
-A input -p tcp -s 0/0 -d 0/0 0:1023 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 2049 -y -j REJECT
-A input -p udp -s 0/0 -d 0/0 0:1023 -j REJECT
-A input -p udp -s 0/0 -d 0/0 2049 -j REJECT
-A input -p tcp -s 0/0 -d 0/0 6000:6009 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 7100 -y -j REJECT
```

At this point users on remote systems with accounts on this system should be able to rsh/rlogin to this machine without using a password.

Enabling ssh Enabling ssh is somewhat easier.

First the ssh server, sshd, must be installed. This is part of the openssh-server RPM. This RPM is located on the first disc of the RedHat 7.2 distribution.

Once the server is installed, it must be started:

```
/etc/rc.d/init.d/sshd start
```

The service will be automatically started on reboot.

At this point ssh on the localhost should work, although a password will still be required. However, our firewall rules will be preventing connections from other machines.

We again modify '`/etc/sysconfig/ipchains`', this time to allow ssh traffic in and out. See the above section for a discussion of what we are doing here.

```
# Firewall configuration written by lokkit
# Manual customization of this file is not recommended.
# Note: ifup-post will punch the current nameservers through the
#       firewall; such entries will *not* be listed here.
:input ACCEPT
:forward ACCEPT
:output ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth0 -j ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth1 -j ACCEPT
-A input -s 0/0 -d 0/0 -i lo -j ACCEPT
#
# New rules for ssh traffic, incoming or outgoing
#
-A input -p tcp -s 0/0 -d 0/0 22 -b -j ACCEPT
#
# End of new rules
#
-A input -p tcp -s 0/0 -d 0/0 0:1023 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 2049 -y -j REJECT
-A input -p udp -s 0/0 -d 0/0 0:1023 -j REJECT
-A input -p udp -s 0/0 -d 0/0 2049 -j REJECT
-A input -p tcp -s 0/0 -d 0/0 6000:6009 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 7100 -y -j REJECT
```

At this point users on remote systems should be able to ssh into the machine, but they will still need a password.

Users should set up a private/public authentication key pair in order for ssh to operate without passwords. This process is documented in the installation guide, but a summary of the steps for RH7.2 will be included here.

First run the "ssh-keygen -t rsa" application to create the private/public key pair. By default this will create the files '`~/.ssh/id_rsa`' and '`~/.ssh/id_rsa.pub`'. Use a password.

Next place the public key ('`~.ssh/id_rsa.pub`') in the file '`~/.ssh/authorized_keys`'. If more than one machine is going to be used, then this key must be put in the '`~/.ssh/authorized_keys`' file on each machine. The permissions on the .ssh directory should be set to 700; otherwise the sshd may choose to not accept the keys.

This will allow you to connect using rsa keys rather than simple UNIX passwords. The next step is to enable an SSH agent so that you do not need to repeatedly type your password.

The agent is started with "`ssh-agent <cmd>`". Typically `<cmd>` is `$SHELL`, so that your default shell is started. The agent will then handle authentication on your behalf any time you attempt to use ssh from this shell. To give the ssh-agent your password, type "ssh-add". This will query you for the passphrase that accompanies your rsa key.

Once you have completed this, you will be able to ssh to other systems on which your key is authorized without typing a password.

Interprocess communication MPICH processes use the standard UNIX mechanisms for allocating ports for intercommunication. Using this mechanism processes are given ports in the range of 1024–65535.

Unfortunately for us, the default firewall configuration blocks some port ranges that our MPICH processes might be given to use for communication. This leads to a situation where MPICH applications will occasionally fail to communicate (when they happen to get the wrong port value).

We're going to modify the ipchains configuration file to remove lines disabling ranges of ports that our processes might use for intercommunication.

The two default rules of interest are the following:

```
-A input -p tcp -s 0/0 -d 0/0 6000:6009 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 7100 -y -j REJECT
```

The first blocks incoming TCP connections to ports 6000-6009 (often used by X), while the second blocks incoming TCP connections to port 7100 (often used by the X font server).

We simply remove these rules:

```
# Firewall configuration written by lokkit
# Manual customization of this file is not recommended.
# Note: ifup-post will punch the current nameservers through the
#       firewall; such entries will *not* be listed here.
:input ACCEPT
:forward ACCEPT
:output ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth0 -j ACCEPT
-A input -s 0/0 67:68 -d 0/0 67:68 -p udp -i eth1 -j ACCEPT
-A input -s 0/0 -d 0/0 -i lo -j ACCEPT
-A input -p tcp -s 0/0 -d 0/0 0:1023 -y -j REJECT
-A input -p tcp -s 0/0 -d 0/0 2049 -y -j REJECT
-A input -p udp -s 0/0 -d 0/0 0:1023 -j REJECT
-A input -p udp -s 0/0 -d 0/0 2049 -j REJECT
#
# Removed these rules to eliminate chance of MPICH comm. failure
#
# -A input -p tcp -s 0/0 -d 0/0 6000:6009 -y -j REJECT
# -A input -p tcp -s 0/0 -d 0/0 7100 -y -j REJECT
#
# End of removed rules
#
```

This modification, in conjunction with one to allow process startup, should prepare your system for MPICH jobs.

A.6 poll: protocol failure during circuit creation

You may see this message if you attempt to run too many MPI programs in a short period of time. For example, in Linux and when using the `ch_p4` device (without the secure server or ssh), `MPICH` uses rsh to start the MPI processes. Depending on the particular Linux distribution and verison, there may be a limit of as few as 40 processes per minute. When running the `MPICH` test suite or starting short parallel jobs from a script, it is possible to exceed this limit.

To fix this, you can do one of the following:

1. Wait a few seconds between running parallel jobs. You may need to wait up to a minute.
2. Modify '`/etc/inetd.conf`' to allow more processes per minute for `rsh`. For example, change

   ```
   shell stream tcp nowait root /etc/tcpd2 in.rshd
   ```

   to

   ```
   shell stream tcp nowait.200 root /etc/tcpd2 in.rshd
   ```

3. Use the `ch_p4mpd` device or the secure server option of the `ch_p4` device instead. Neither of these relies on `inetd`.

A.7 Using SSH

The secure shell (ssh) may be used with the `ch_p4` device, but requires careful setup. See *configuring with `ssh`* in the *Installation and User's* manual.

Make sure that `ssh` is set up to *not* require a password. The command

```
ssh -n `hostname` date
```

should return the date without any prompts for passwords. See the installation manual if you have problems.

A.8 Mac OS X and hostname

Under Mac OS X, the hostname handling is unusual. The hostname that the `hostname` command or the `gethostname` function return is determined by the name set in the "Sharing preference" pane. If this name contains a space, you may get only the leading part of the name, upto but not including the first space. This name is stored in the file `/etc/hostconfig`. MPICH, for the `ch_p4` and `ch_p4mpd` device, needs a sensible name returned by both `gethostname` and `hostname`. If you are using a single machine in a standalone configuration, the easiest option is to change the name of the machine in

/etc/hostconfig to localhost (use sudo). If you wish to use another name, you need to make the name in /etc/hostconfig match the name in the NetInfo database.

Thanks to Victor Eijkhout for this information.

A.9 SIGSEGV

If your program fails with

```
p4_error: interrupt SIGSEGV
```

the problem is probably not with MPI. Instead, check for program bugs including

1. Array overwrites or accesses beyond array bounds. Be particularly careful of a[size] in C, where a is declared as int a[size].
2. Invalid pointers, including null pointers.
3. Missing or mismatched parameters to subroutines or functions. Fortran users should check that all MPI calls include the integer error return parameter and that any status variable is dimensioned as an array of size MPI_STATUS_SIZE.

A.10 semop lock failed

When running the ch_p4 device with SMP support (-comm=shared), you may occasionally see the message

```
"p1_13043:  p4_error: OOPS: semop lock failed"
```

To fix this, try running the script cleanipcs that is included with MPICH. You can also use the command ipcs to list the shared memory and semaphore resources that are in use on a node. This can help you track down resources that are held by a different user that are preventing your MPI program from running.

A.11 Compiler Switches

Normally, you should let configure determine compiler switches. However, you can use the configure options -cflags=... and -fflags=... to specify special flags. See also *compiler switches*.

A.12 C++ Builds Fail

If the C++ build fails with messages about ambiquities in the definitions, try reconfiguring and rebuilding with the –without-profiling options to the C++ configure. In later versions of MPICH, a different C++ interface will be provided that should fix this problem.

A.13 Fortran programs give errors about mismatched types

Some Fortran compilers will complain if a subroutine is passed arguments of different types in the same parameter position. For example, if `MPI_Bcast` is used with an integer in one place and a double precision value in another, you may see

```
Argument #1 of 'mpi_bcast' is one type at (2) but is some other type at (1)
```

This is a strict interpretation of the Fortran standard. To fix this, you will need to tell the compiler to allow this usage. For the GNU g77 compiler, add the command-line option `-Wno-globals`, as in

```
mpif77 -Wno-globals mycode.f
```

An alternative is to use a Fortran 90 or Fortran 95 compiler with the MPI module instead of the '`mpif.h`' header file.

A.14 Missing Symbols When Linking

If you get errors about missing symbols, such as

```
mpicc  -o overtake overtake.o test.o
ld: 0711-317 ERROR: Undefined symbol: MPIR_F_TRUE
ld: 0711-317 ERROR: Undefined symbol: .MPIR_InitFortranDatatypes
ld: 0711-317 ERROR: Undefined symbol: MPIR_F_FALSE
ld: 0711-317 ERROR: Undefined symbol: .MPIR_InitFortran
ld: 0711-317 ERROR: Undefined symbol: MPIR_I_DCOMPLEX
ld: 0711-317 ERROR: Undefined symbol: .MPIR_Free_Fortran_dtes
ld: 0711-317 ERROR: Undefined symbol: .MPIR_Free_Fortran_keyvals
```

this usually indicates an error in the `make` process. For some reason, the Fortran part (which is where these symbols come from) is particularly fragile. To fix this, try the following steps:

```
cd src/fortran/src
make clean
make
ar ../../../lib/libmpich.a *.o
ranlib ../../../lib/libmpich.a
```

If weak symbols are not supported, then *in addition*, do these additional steps:

```
make clean
make profile
ar ../../../lib/libpmpich.a *.o
ranlib ../../../lib/libpmpich.a
```

The problem is that some versions of make have logic errors that cause them to create files but not to act on them; this causes make to build the object file but then fail to include it in the archive. The above steps should work around this problem.

A.15 Warning messages while building MPICH

Some compilers may generate a large number of warnings of the form

```
"commreq_free.c", line 70: warning #187: use of "=" where "==" may have been intended
```

There is nothing wrong with these statements. The compiler is warning about a legal, but often misused, feature of the C language. The statements have been crafted so that most compilers recognize that the "=" was used intentionally; unfortunately, some compilers insists on warning about this valid use of C and provide no way to indicate to the compiler that the warning is unnecessary.

A.16 MPMD (Multiple Program Multiple Data) Programs

MPICH, depending on the device, supports MPMD programs. However, the `mpirun` script currently does not support MPMD programs. For the `ch_p4` device, the user must create a procgroup file and invoke the program that will have rank zero in `MPI_COMM_WORLD` with the command-line option `-p4pg filename`. See the *Installation and User's Guide* for more information.

A.17 Reporting problems and support

1. First, check *the list of known bugs and patches* for the problem you are seeing.
2. Also check the troubleshooting guides in the *Installation and Users guides.*
3. If that doesn't help, send mail to *mpi-bugs@mcs.anl.gov*.

A.18 Algorithms used in MPICH

1. Does MPICH use IP Multicast for `MPI_Bcast`?

   No. In principle, MPICH could use multicast, but in practice this would be very difficult. To start with, IP multicast is unreliable; additional code to make it reliable needs to be added. In fact, there is an effort to provide a reliable multicast, built ontop of the unreliable multicast. The second problem is that not all systems allow user programs (or *any* program) to perform an IP multicast. In fact, that is the case for the systems that we have been developing on. Thus, we will always need the point-to-point version. There is a fairly easy way to replace any collective routine in MPI, but no-one has offered us a multicast-based `MPI_Bcast` yet...

A.19 Jumpshot and X11

Jumpshot relies on the AWT/Swing toolkit to perform its graphics operations. We have noticed some problems with AWT/Swing and X11 window system support on some PC's running Windows. In particular, the combination of SecureCRT and Exceed or SecureCRT and Xfree86 often have problems (sometimes serious, including hangs and crashes). For now, the only workaround is to use OpenSSH and Xfree86 (with Cygwin) if possible when using a Windows PC.

B History of `MPICH`

`MPICH` was developed during the MPI standards process to provide feedback to the MPI Forum on implementation and usability issues. With the release of the MPI standard, `MPICH` was designed to provide an implementation of the MPI standard that could replace the proprietary message-passing systems on the massively parallel computers of the day, such as the Intel Paragon, IBM SP, and TMC CM5. `MPICH` used an early version of the abstract device interface (ADI), based on the Chameleon [14] portability system, to provide a light-weight implementation layer. To enable development on desktop systems, a device layered on top of the P4 [1] system was used. Over time, other devices were developed; as systems have vanished (e.g., the TMC CM5 and the Ncube), these devices have been dropped from our distribution. Because `MPICH` used P4 for workstation networks, `MPICH` has supported both MIMD programming and heterogeneous clusters from the very beginning.

Because `MPICH` was designed to enable ports to other systems, many parallel computer vendors and research groups have used `MPICH` as the basis for their implementation. Many users are now familar only with the version of `MPICH` that uses the `ch_p4` device for workstation and Beowulf clusters. However, `MPICH` continues to support other systems and continues to serve as a platform for research into MPI implementations.

C File Manifest

This section briefly describes the files and directories at the top level of the `MPICH` source tree.

`COPYRIGHT` Copyright statement. This code is free but not public domain. It is copyrighted by the University of Chicago and Mississippi State University.

`Makefile.in` Template for the '`Makefile`', which will be produced when you run `configure`.

`MPI-2-C++` The C++ system from Notre Dame. It includes the C++ bindings for the MPI-1 functions.

`README` Basic information and instructions for configuring.

`aclocal.m4` Used for building '`configure`' from '`configure.in`'; not needed for most installations. The file '`aclocal_tcl.m4`' is included by '`aclocal.m4`'.

`ccbugs` Directory for programs that test the C compiler during configuration, to make sure that it will be able to compile the system.

`configure` The script that you run to create Makefiles throughout the system.

`configure.in` Input to `autoconf` that produces `configure`.

`doc` Assorted tools for producing documentation, together with this manual.

`examples` Directory containing further directories of example MPI programs. Of particular note are `basic`, with a few small examples to try first, `test`, with a test suite for exercising `MPICH`, and `perftest`, containing benchmarking code.

`include` The include files, both user and system.

`bin` Contains the programs and executable scripts, such as `mpicc` and `mpirun`, used to build and run MPI programs.

`man` Man pages for `MPI`, `MPE`, and internal routines.

`mpe` The source code for the `MPE` extensions for logging and X graphics. The `contrib` directory contains examples. Best are the `mandel` and `mastermind` subdirectories. The `profiling` subdirectory contains the profiling subsystem, including a system for automatically generating the "wrappers" for the MPI profiling interface. MPE also includes the performance visualization programs, such as `jumpshot` (see Section 3.6.1).

`mpid` The source code for the various "devices" that customize `MPICH` for a particular machine, operating system, and environment.

`romio` The ROMIO parallel I/O system, which includes an implementation of most of the MPI-2 parallel I/O standard.

`src` The source code for the portable part of `MPICH`. There are subdirectories for the various parts of the MPI specification.

`util` Utility programs and files.

`www` HTML versions of the `man` pages.

D Configure Usage

The command `configure -help` will print out

```
Configuring with args --help
Configuring MPICH Version 1.2.3 (pre-alpha) of : 2001/08/20 18:08:23
Usage: ./configure [--with-arch=ARCH_TYPE] [--with-comm=COMM_TYPE]
            [--with-device=DEVICE]
            [--with-mpe] [--without-mpe]
            [--disable-f77] [--disable-f90] [--with-f90nag] [--with-f95nag]
            [--disable-f90modules]
            [--disable-gencat] [--disable-doc]
            [--enable-cxx ] [--disable-cxx]
            [--enable-mpedbg] [--disable-mpedbg]
            [--enable-devdebug] [--disable-devdebug]
            [--enable-debug] [--disable-debug]
            [--enable-traceback] [--disable-traceback]
            [--enable-long-long] [--disable-long-long]
            [--enable-long-double] [--disable-long-double]
            [-prefix=INSTALL_DIR]
```

```
            [-c++[=C++_COMPILER] ] [noc++]
            [-opt=OPTFLAGS]
            [-cc=C_COMPILER] [-fc=FORTRAN_COMPILER]
            [-clinker=C_LINKER] [-flinker=FORTRAN_LINKER]
            [-c++linker=CC_LINKER]
            [-cflags=CFLAGS] [-fflags=FFLAGS] [-c++flags=CCFLAGS]
            [-optcc=C_OPTFLAGS] [-optf77=F77_OPTFLAGS]
            [-f90=F90_COMPILER] [-f90flags=F90_FLAGS]
            [-f90inc=INCLUDE_DIRECTORY_SPEC_FORMAT_FOR_F90]
            [-f90linker=F90_LINKER]
            [-f90libpath=LIBRARY_PATH_SPEC_FORMAT_FOR_F90]
            [-lib=LIBRARY] [-mpilibname=MPINAME]
            [-mpe_opts=MPE_OPTS]
            [-make=MAKEPGM ]
            [-memdebug] [-ptrdebug] [-tracing] [-dlast]
            [-listener_sig=SIGNAL_NAME]
            [-cross]
            [-adi_collective]
            [-automountfix=AUTOMOUNTFIX]
            [-noranlib] [-ar_nolocal]
            [-rsh=RSHCOMMAND] [-rshnol]
            [-noromio] [-file_system=FILE_SYSTEM]
            [-p4_opts=P4_OPTS]
where
   ARCH_TYPE       = the type of machine that MPI is to be configured for
   COMM_TYPE       = communications layer or option to be used
   DEVICE          = communications device to be used
   INSTALL_DIR     = directory where MPI will be installed (optional)
   MPE_OPTS        = options to pass to the mpe configure
   P4_OPTS         = options to pass to the P4 configure (device=ch_p4)
   C++_COMPILER    = default is to use xlC, g++, or CC (optional)
   OPTFLAGS        = optimization flags to give the compilers (e.g. -g)
   CFLAGS          = flags to give C compiler
   FFLAGS          = flags to give Fortran compiler
   MAKEPGM         = version of make to use
   LENGTH          = Length of message at which ADI switches from short
                     to long message protocol
   AUTOMOUNTFIX    = Command to fix automounters
   RSHCOMMAND      = Command to use for remote shell
   MPILIBNAME      = Name to use instead of mpich in the name of the MPI
                     library.  If set, libMPILIBNAME will be used instead
                     or libmpich.  This can be used on systems with
                     several different MPI implementations.
   FILE_SYSTEM     = name of the file system ROMIO is to use.  Currently
                     supported values are nfs, ufs, pfs (Intel),
                     piofs (IBM), hfs (HP), sfs (NEC), and xfs (SGI).
   SIGNAL_NAME     = name of the signal for the P4 (device=ch_p4) device to
                     use to indicate that a new connection is needed.  By
                     default, it is SIGUSR1.

All arguments are optional, but if 'arch', 'comm', or 'prefix' arguments
are provided, there must be only one.  'arch' must be specified before
'comm' if they both appear.
```

```
Packages that may be included with MPICH

--with-device=name     - Use the named device for communication.  Known
                         names include ch_p4, ch_mpl, ch_shmem, and globus2.
                         If not specified, a default is chosen.  Special
                         options for the device are specified after the
                         device name, separated by a colon.  E.g.,
                         --with-device=globus2:-flavor=mpi,nothreads
--with-romio[=OPTIONS] - Use ROMIO to provide MPI-I/O from MPI-2 (default).
                         The options include -file_system=FSTYPE, where
                         fstype can be any combination of nfs, ufs,
                         pfs (intel), piofs (IBM), hfs (HP), sfs (NEC), and
                         xfs (SGI), combined with '+'.  If romio is not
                         included, the Fortran 90 modules cannot be built.
--with-mpe             - Build the MPE environment (default)
--with-f90nag          - Choose the NAG f90 compiler for Fortran
                         (preliminary version intended for use *instead*
                         of a Fortran 77 compiler)
--with-f95nag          - Choose the NAG f95 compiler for Fortran
--with-cross=file      - Use the file for cross compilation. The file
                         should contain assignments of the form
                            CROSS_SIZEOF_INT=4
                         for each cross compilation variable.  The command
                            egrep 'CROSS_[A-Z_]*=' configure | sed 's/=.*//g'
                         will list each variable.

You can use --without-<featurename> to turn off a feature (except for
device).
Options for device ch_lfshmem:
--with-device=ch_lfshmem[:-usesysv]
The option '-usesysv' applies to the ch_shmem device, and causes the device
to attempt and use System V shared memory and semaphore routines, rather
than what would be chosen by default (often mmap or a system-specific method).

Options for device ch_meiko:
--with-device=ch_meiko

Options for device ch_mpl:
--with-device=ch_mpl

Options for device ch_p4:
./configure: ./mpid/ch_p4/setup_ch_p4: Permission denied

Options for device ch_p4mpd:
--with-device=ch_p4mpd[:-listener_sig=SIGNALNAME][-dlast][-socksize=BYTES]

The option '-listener_sig' applies to the ch_p4mpd device, and changes the
signal that is used to signal that a new connection should be made.  By
default, SIGUSR1 is used.

The option '-dlast' causes the p4 device to remember the last few debugging
messages, printing them out only when the job terminates abnormally.

The option '-socksize' changes the size of the socket buffers used.
```

```
Options for device ch_shmem:
--with-device=ch_shmem[:-usesysv]
The option '-usesysv' applies to the ch_shmem device, and causes the device
to attempt and use System V shared memory and semaphore routines, rather
than what would be chosen by default (often mmap or a system-specific method).

Options for device globus:
--with-device=globus[:-globusdir=GLOBUSDIR]
'-globusdir=GLOBUS' allows one to specify the location of an installed
version of Globus.  You can acquire Globus from http://www.globus.org ."

Options for device globus2:
*# GLOBUS_INSTALL_PATH must be set

Features that may be included with MPICH

--enable-c++          - Build C++ interfaces to the MPI-1 routines
                        (default)
--enable-f77          - Build Fortran 77 interfaces to the MPI
                        routines (default)
--enable-weak-symbols - Use weak symbols for MPI/PMPI routines.  This uses
                        weak symbols, if available, for the profiling
                        interface (default)
--enable-debug        - Enable support for debuggers to access message
                        queues
--enable-traceback    - Enable printing of a call stack when MPI and the
                        user's program is built with certain compilers
                        (currently only some versions of gcc are supported).
--enable-mpedbg       - Enable the -mpedbg command-line argument (e.g.,
                        errors can start an xterm running a debugger).
                        Only works with some workstation systems.
--enable-sharedlib    - Attempt to build shared libraries.  Static
--enable-sharedlib=dir  libraries are always built.  If a directory is
                        specified, the shared libraries will be placed in
                        that directory.  This can be used to place the
                        shared libraries in a uniform location in local
                        disks on a cluster.
--enable-f90modules   - Build Fortran 90 module support (default if a
                        Fortran 90 or 95 compiler is found).  If ROMIO
                        is not built, no Fortran 90 modules will be built.

The following are intended for MPI implementors and debugging of configure
--enable-strict       - Try and build MPICH using strict options in Gnu gcc
--enable-echo         - Cause configure to echo what it does
--enable-devdebug     - Enable debugging code in the ADI.

You can use --disable-<featurename> to turn off a feature.

Notes on configure usage:

The suggestions for GNU configure usage suggest that configure not be used
to build different tools, only controlling some basics of the features
enabled or the packages included.  Our use of configure does not follow
these rules because configure is too useful but we need the flexibility
```

that allows the user to produce variations of MPICH.

More notes on command-line parameters:

You can select a different C and Fortran compiler by using the '-cc' and 'fc' switches. The environment variables 'CC' and 'FC' can also provide values for these but their settings may be overridden by the configure script. Using '-cc=$CC -fc=$FC' will force configure to use those compilers.

The option '-opt' allows you to specify optimization options for the compilers (both C and Fortran). For example, '-opt=-O' chooses optimized code generation on many systems. '-optcc' and '-optf77' allow you to specify options for just the C or Fortran compilers. Use -cflags and -fflags for options not related to optimization.

Note that the '-opt' options are not passed to the 'mpicc', 'mpif77', 'mpiCC', and 'mpif90' scripts. The '-opt' options are used only in building MPICH.

The option '-lib' allows you to specify the location of a library that may be needed by a particular device. Most devices do NOT need this option; check the installation instructions for those that might.

The option '-make' may be used to select an alternate make program. For example, to make use of VPATH builds (building in one directory with the source in a different directory), -make=gnumake may be required.

The option '--disable-short-longs' may be used to suppress support for the C types 'long long' (a common extension) and 'long double' (ANSI/ISO C) when they are the same size as 'long' and 'double' respectively. Some systems allow these long C types, but generate a warning message when they are used; this option may be used to suppress these messages (and support for these types). '--disable-long-long' disables just 'long long'; '--disable-long-double' disables just 'long double'.

The option '-ar_nolocal' prevents the library archive command from attempting to use the local directory for temporary space. This option should be used when (a) there isn't much space (less than 20 MB) available in the partition where MPICH resides and (b) there is enough space in /tmp (or wherever ar places temporary files by default).

The option '-noranlib' causes the 'ranlib' step (needed on some systems to build an object library) to be skipped. This is particularly useful on systems where 'ranlib' is optional (allowed but not needed; because it is allowed, configure chooses to use it just in case) but can fail (some 'ranlib's are implemented as scripts using 'ar'; if they don't use the local directory, they can fail (destroying the library in the process) if the temporary directory (usually '/tmp') does not have enough space. This has occured on some OSF systems.

The option '-rsh' allows you to select an alternative remote shell command (by default, configure will use 'rsh' or 'remsh' from your 'PATH'). If your remote shell command does not support the '-l' option (some AFS versions of 'rsh' have this bug), also give the option '-rshnol'. These options are useful only when building a network version of MPICH (e.g., '--with-device=ch_p4').

Special Tuning Options:

There are a number of options for tuning the behavoir of the ADI (Abstract Device Interface) which is the low-level message-passing interface. These should NOT be used unless you are sure you know what you are doing.

The option '-pkt_size=LENGTH' allows you to choose the message length at which the ADI (Abstract Device Interface) switches from its short to long message format. LENGTH must be positive.

The option '-adi_collective' allows the ADI to provide some collective operations in addition to the basic point-to-point operations. Currently, most systems do not support this option (it is ignored) and on the others it has not been extensively tested.

Options for Experts:

The option '-memdebug' enables extensive internal memory debugging code. This should be used only if you are trying to find a memory problem (it can be used to help find memory problems in user code as well). Running programs with the option '-mpidb memdump' will produce a summary, when 'MPI_Finalize' is called, of all unfreed memory allocated my MPI. For example, a user-created datatype that was not later freed would be reported.

The option '-tracing' enables tracing of internal calls. This should be used only for debugging the MPICH implementation itself.

The option '-dlast' enables tracing of the most recent operations performed by the device. These can be output when a signal (like SIGINT), error, or call to a special routine occurs. There is a performance penalty for this option, but it can be very useful for implementors attempting to debug problems.

Sample Configure Usage:

To make for running on Sun's running Solaris with ch_p4 as the device, and with the installation directory equal to the current directory:

```
  ./configure --with-device=ch_p4 --with-arch=solaris
  make
```

```
Known devices are
        ch_nx     (native Intel NX calls),
        ch_mpl    (native IBM EUI or MPL calls),
        ch_p4     (p4)
        ch_p4mpd  (p4 with the MPD startup system)
        globus2   (Globus: globus_io/vMPI)
        ch_meiko  (for Meiko CS2, using NX compatibility library),
```

```
        ch_shmem  (for shared memory systems, such as SMPs),
        ch_lfshmem(for shared memory systems, such as SMPs; uses
                   lock-free message buffers),
        ch_cenju3 (native NEC Cenju-3 calls)

The following devices were supported with ADI-1, but are currently
unsupported.  Please contact us if you are interested in helping us
support these devices:
        meiko     (for Meiko CS2, using elan tport library), and
        nx        (for Intel Paragon),
        t3d          (for the Cray T3D, using Cray shmem library).
        ch_nc     (native nCUBE calls, requires -arch=ncube),
        ch_cmmd   (native TMC CM-5 CMMD calls)
These are no longer distributed with the MPICH distribution.

Known architectures include (case is important)
        alpha     (Compaq alpha)
        CRAY      (CRAY XMP, YMP, C90, J90, T90)
        cray_t3d  (CRAY T3D)
        EWS_UX_V  (NEC EWS4800/360AD Series workstation.  Untested.)
        freebsd   (PC clones running FreeBSD)
        hpux      (HP UX)
        intelnx   (Intel i860 or Intel Delta)
        IRIX      (synonym for sgi)
        IRIX32    (IRIX with 32bit objects -32)
        IRIXN32   (IRIX with -n32)
        IRIX64    (IRIX with 64bit objects)
        ksr       (Kendall Square KSR1 and KSR2)
        LINUX     (PC clones running LINUX)
        LINUX_ALPHA (Linux on Alpha processors)
        meiko     (Meiko CS2)
        netbsd    (PC clones running NetBSD)
        paragon   (Intel Paragon)
        rs6000    (AIX for IBM RS6000)
        sgi       (Silicon Graphics IRIX 4.x, 5.x or 6.x)
        sgi5      (Silicon Graphics IRIX 5.x on R4400's, for the MESHINE)
        solaris   (Solaris)
        solaris86 (Solaris on Intel platforms)
        sppux     (SPP UX)
        sun4      (SUN OS 4.x)
        SX_4_float0
                  (NEC SX-4; Floating point format float0
                             Conforms IEEE 754 standard.
                   C:       sizeof (int)     = 4; sizeof (float) = 4
                   FORTRAN: sizeof (INTEGER) = 4; sizeof (REAL)  = 4)
        SX_4_float1
                  (NEC SX-4; Floating point format float1
                             IBM floating point format.
                   C:       sizeof (int)     = 4; sizeof (float) = 4
                   FORTRAN: sizeof (INTEGER) = 4; sizeof (REAL)  = 4)
        SX_4_float2
                  (NEC SX-4; Floating point format float2
                             CRAY floating point format.
                   C:       sizeof (int)     = 4; sizeof (float) = 8
```

```
                   FORTRAN: sizeof (INTEGER) = 8; sizeof (REAL)  = 8)
                   !!! WARNING !!! This version will not run
                                   together with FORTRAN routines.
                                   sizeof (INTEGER) != sizeof (int)
        SX_4_float2_int64
                   (NEC SX-4; Floating point format float2 and
                              64-bit int's)
                    C:       sizeof (int)     = 8; sizeof (float) = 8
                    FORTRAN: sizeof (INTEGER) = 8; sizeof (REAL)  = 8)
        tflops    (Intel TFLOPS)
        UXPM      (UXP/M. Untested.)
        uxpv      (uxp/v. Untested.)

    Others may be recognized.

Special notes:

For SGI (--with-arch=IRIX) multiprocessors running the ch_p4 device, use
-comm=ch_p4 to disable the use of the shared-memory p4 communication
device, and -comm=shared to enable the shared-memory p4 communication
device.  The default is to enable the shared-memory communication device.
```

E Mpirun Usage

The options for **mpirun** as shown by **mpirun -help**, are (note that not all options are supported by all devices). Depending on the specific device, the output of **mpirun -help** may differ; the following is for the **globus2** device.

```
mpirun [mpirun_options...] <progname> [options...]

  mpirun_options:
    -arch <architecture>
            specify the architecture (must have matching machines.<arch>
            file in /usr/local/mpich/bin/machines) if using the execer
    -h      This help
    -machine <machine name>
            use startup procedure for <machine name>
mpirun [mpirun_options...] <progname> [options...]

  mpirun_options:
    -arch <architecture>
            specify the architecture (must have matching machines.<arch>
            file in /usr/local/mpich/bin/machines) if using the execer
    -h      This help
    -machine <machine name>
            use startup procedure for <machine name>
            Currently supported:
              paragon
              p4
```

```
          sp1
          ibmspx
          anlspx
          sgi_mp
          ipsc860
          inteldelta
          cray_t3d
          execer
          smp
          symm_ptx
-machinefile <machine-file name>
        Take the list of possible machines to run on from the
        file <machine-file name>. This is a list of all available
        machines; use -np <np> to request a specific number of machines.
-np <np>
        specify the number of processors to run on
-nodes <nodes>
        specify the number of nodes to run on (for SMP systems,
        currently only ch_mpl device supports this)
-nolocal
        don't run on the local machine (only works for ch_p4 jobs)
-all-cpus, -allcpus
        Use all available CPUs on all the nodes.
-all-local
        Run all processes on the master node.
-exclude <list>
        Exclude nodes in a colon delimited list.
-map <list>
        Use the colon delimited list to specify which rank
        runs on which nodes.
-stdin filename
        Use filename as the standard input for the program.  This
        is needed for programs that must be run as batch jobs, such
        as some IBM SP systems and Intel Paragons using NQS (see
        -paragontype below).
        use
              -stdin /dev/null
        if there is no input and you intend to run the program in the
        background.  An alternate is to redirect standard input from
        /dev/null, as in
              mpirun -np 4 a.out < /dev/null
-t      Testing - do not actually run, just print what would be
        executed
-v      Verbose - throw in some comments
-dbg    The option '-dbg' may be used to select a debugger.  For example,
        -dbg=gdb invokes the mpirun_dbg.gdb script located in the
        'mpich/bin' directory.  This script captures the correct arguments,
        invokes the gdb debugger, and starts the first process under gdb
        where possible.  There are 4 debugger scripts; gdb, xxgdb, ddd,
        totalview.  These may need to be edited depending on your system.
        There is another debugger script for dbx, but this one will always
        need to be edited as the debugger commands for dbx varies between
        versions.  You can also use this option to call another debugger;
        for example, -dbg=mydebug.  All you need to do is write a script
```

```
        file, 'mpirun_dbg.mydebug', which follows the format of the
        included debugger scripts, and place it in the mpich/bin directory.
-ksq    Keep the send queue. This is useful if you expect later
        to attach totalview to the running (or deadlocked) job, and
        want to see the send queues. (Normally they are not maintained
        in a way which is visible to the debugger).
```

```
Options for the globus2 device:

With the exception of -h, these are the only mpirun options supported
by the globus device.

-machinefile <machine-file name>
        Take the list of possible machines to run on from the
        file <machine-file name>
-np <np>
        specify the number of processors to run on
-dumprsl - display the RSL string that would have been
          used to submit the job.  using this option
          does not run the job.
-globusrsl <globus-rsl-file name>
        <globus-rsl-file name> must contain a Globus RSL
        string.  When using this option all other mpirun options
        are ignored.


Special Options for Batch Environments:

-mvhome Move the executable to the home directory.  This
        is needed when all file systems are not cross-mounted
        Currently only used by anlspx
-mvback files
        Move the indicated files back to the current directory.
        Needed only when using -mvhome; has no effect otherwise.
-maxtime min
        Maximum job run time in minutes.  Currently used only
        by anlspx.  Default value is $max_time minutes.
-nopoll Do not use a polling-mode communication.
        Available only on IBM SPs.

Special Options for IBM SP2:

-cac name
        CAC for ANL scheduler.  Currently used only by anlspx.
        If not provided will choose some valid CAC.
```

On exit, `mpirun` returns a status of zero unless `mpirun` detected a problem, in which case it returns a non-zero status.

When using the `ch_p4` device, multiple architectures may be handled by giving multiple `-arch` and `-np` arguments. For example, to run a program on 2 sun4s and 3 rs6000s, with the local machine being a sun4, use

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program
```

This assumes that program will run on both architectures. If different executables are needed, the string '%a' will be replaced with the arch name. For example, if the programs are `program.sun4` and `program.rs6000`, then the command is

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program.%a
```

If instead the executables are in different directories; for example, '`/tmp/me/sun4`' and '`/tmp/me/rs6000`', then the command is

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 /tmp/me/%a/program
```

It is important to specify the architecture with `-arch` *before* specifying the number of processors. Also, the *first* `arch` command must refer to the processor on which the job will be started. Specifically, if `-nolocal` is *not* specified, then the first `-arch` must refer to the processor from which mpirun is running.

F Change summary from the previous version

This section lists the changes to `MPICH` since the previous version. These changes are taken from the CVS log entries that accompany each change and may not be complete or entirely accurate.

- '#elif defined(FOO)' is better than '#elif FOO' if FOO might be undefined
- (romio) clean up the output a bit more
- (various configure.in files) for autoconf 2.13 and on, AC_SEARCH_LIBS adds any necessary libraries to the LIBS variable, so I modified our configure scripts to no longer rely on internal autoconf variables (ac_vc_search_*) and instead use LIBS.
- a start at adding pvfs2-specific hints
- Add DESTDIR to installation paths
- Add NETBSD uniformly to timer directory
- add PVFS2 to the list of filesystems that don't support shared file pointers
- Add a test for no f90 compiler
- Add code to save/set status on split collectives
- Add locations for Java

- Add mpi include path when testing for MPI_Offset size and only perform that test for Fortran
- Add mpichversion to MPICH and ensure that change directory commands in Makefile are successful before executing the next step
- Add new methods for data type SLOG_strlist_t
- Add support for io tests and upgrade runbase
- added 'struct gridftp_params' for MPICH-G2's use of Globus GridFTP
- Added .deps to .cvsignore
- Added AC_OBJEXT and AC_EXEEXT changes to Makefiles to support cygwin.
- Added AC_OBJEXT and AC_EXEEXT to configure.in and related changes to aclocal_mpi.m4, this is done to support cygwin.
- added API for IBM
- Added BUILDING_MPI symbol to suppress link tests in the midst of building
- Added CPPFLAGS to PAC_TEST_PROGRAM
- Added EXEEXT to move targets for cygwin
- Added FREEBSD_PPC machine time (like FREEBSD except for data representation
- Added FREEBSD_PPC to known arches for p4
- Added FREEBSD_PPC to known systems
- Added Fortran binding for MPE_Log_send() and MPE_Log_receive()
- Added LINUX_PPC to known arches
- Added MPE_LOGFILE_PREFIX to allow specification of MPE output filename prefix.
- added MPI_Pcontrol for illustration
- Added RANLIB
- Added Rabenseifner's algorithm for long-message Reduce and Allreduce. Also added a lgp recursive halving algorithm for Reduce_scatter in the case of commutative ops
- Added TAGS targets
- added __LINE__ to MPIR_Err_create_code
- Added a check for time.h and that sys/time.h and time.h may be used together
- Added a comment about determining the largest fd
- Added a default for F77 if no architecture is recognized
- Added a dummy definition of GREQUEST to make the romioconf.h.in files the same for configure and configure2
- Added a few comments about fields in File structure
- Added a fix for zap_p4_processes
- Added a freebsd_ppc type (likely MacOSX)
- Added a macro defn so that other parts of MPICH can tell if MPIO_Request has been defiend

- Added a more forgiving check for a valid arch in the MACHINES file
- Added a new test with an increment of two for group creation
- Added a note about problems with the AC_EXEEXT macro in autoconf 2.13
- Added a short comment about MPI_CC
- Added a test for READ_ONLY file
- Added a test for a failure in ROMIO configure
- Added a test that struct statfs can be compiled, because of problems with the Intel C compiler and glibc
- Added a way to skip the does-it-run-cleanly step, needed for coverage tests (which rarely run cleanly)
- added ad_iopen.c
- Added additional positional args for gcc
- Added additional search dirs
- added an MPI_Type_free for an MPI_Allgather subroutine
- Added an additional step to drain incoming messages and a comment about ordering head-to-head rendezvous messages
- Added an empty coverage target to MPE
- Added an option for an environment variable to set the stopfile
- Added another Linux synonym
- Added another test for cancel of Issend based on an MPICH bug report (currently MPICH1 fails this test with a SEGV in ch_p4 device)
- Added autom4te.cache to .cvsignore
- added barrier to MPI_File_close to avoid deletion of shared file pointer file while others were still accessing it.
- added bug fix sent in by Wei-keng on atomicity in collective writes (Req #516)
- added casts to int from functions that return size_t and added MPIU_dbg_init() to initthread
- Added check for getopt.h
- Added code to determine the argc/argv support needed for the ROMIO Fortran tests
- added code to propagate environment variables for Nick. Nothing happens unless P4_SETS_ALL_ENVVARS is set
- Added code to provide accurate MPI error classes for failure to open file (e.g., path too long, missing directory, no such file)
- Added comment about incompatible demon options in mpd1 and mpd2
- Added comments about setting the working directory
- Added copyright statements to ROMIO test programs
- Added coverage options and standard enable/disable for f77
- Added coverage output files to cvsignore

- Added coverage targets
- Added definitions to allow ROMIO programs that use MPIO_Wait to work with ROMIO under MPICH2, where generalized requests allow MPI_Wait et al to work
- added documentation comments, reported memory errors, put the pmi database on the root process
- added error checking blocks
- added error code for local success but remote file_open failure
- added error message catalog dll to ch_nt device
- Added exec_prefix and sbindir to mpirun variables so that we can invoke tools like cleanipcs in sbindir
- Added explanation when error seen
- Added export of OPTFLAGS and OPTFLAGSF so that all parts of MPICH get -opt=flags
- Added failure return if manager cannot be found
- Added fixes for autoconf later than 2.13; also make a failure to build slogsdk non-fatal
- Added function prototype and stdlib.h header to eliminate compiler warning on exit()
- added globus_core to globus-makefile-header command as a workaroundto newly introduced bug in either globus-makefile-header (notfinding the implicit dependency for it) or gpt-build (not seeingthe stuff from globus-makefile-header)
- Added hooks for ROMIO to propagate shared library information
- Added hooks for generating shared libraries for ROMIO adio/common directory
- Added if test to ROMIO configure to bypass check for Fortran if –disable-f77 is selected
- Added ifc to f90 compilers
- Added large file support for 32-bit OS: Added test to configure.in to check if O_LARGEFILE is available (Ref: http://ftp.sas.com/standards/large.file/xopen/x_open.05Mar96.html) and required macros to mpeconf.h.in.
- Added markers for error-checking code to aid in coverage analysis
- Added master_top_srcdir and man page documentation targets
- Added missing test for MPI_STATUS_IGNORE
- Added more support for shared libraries
- Added most of the multiple completion waits and tests in MPIO_Waitany etc. form
- Added mpicxx to installed files, and handle case-insensitive file systems (for Mac OS X)
- Added mpicxx to known compiler names
- added new algorithms for alltoall, allgather, and allgatherv
- Added optional definition of S_ISLNK if strict ANSI selected for C
- Added patch sent in by multiple people for uninitialized data bug
- Added pointer in ADIO_File structure for file system specific use.

- Added quotes to protect against an empty variable
- Added some tests for struct flock problems
- Added special support for TFLOPS architecture and devices that do not have MPID_Status_set_bytes
- Added stdlib.h
- Added stdlib.h header to eliminate compiler warning on exit()
- Added support for gcc dependency creation
- Added support for including patch information in mpichversion
- Added support for mpichversion
- Added support for setting group id
- Added support to make SLOG1 API's library and related tools depending on the successful return of the SLOG1 API's configure. This is added to support Cray. Also, modify some global subroutines that have CLOG_ prefix to C2S1_ instead(abbi's code).
- Added temporary ac_subst items to enable the rest of the ROMIO tests
- Added test for CPP
- Added test for errno == 0 because of Solaris bug (return -1 but errno not set)
- Added test for pvfs header and int64_t definition fix
- added text on needing lockd daemon to NFS file lock error message
- Added the package slog2sdk to the configure.in and Makefile.in. Also added support for drop & configure support for slog_api, viewers and slog2sdk.
- adding ch_p4mpd device files
- adding files for lib in p4mpd
- adding files in include dir for p4mpd
- adding mpd source configure- and make-related files
- adding mpd source files
- adding top-level p4mpd files
- adjusted error handling block
- Allow -n as a synonym for -np in mpirun
- Allow a path list for mpd and make sure that it is propagated from the top-level configure
- Allow compile commands with options in command tests
- Allow empty arguments in mpeinstall
- Allow make to be set from the environment
- Assert that we support funneled thread safety
- autoconf already substitutes @prefix@, so don't manually AC_SUBST(PREFIX)
- Be sure to copy all data out of a short received packet before returning it to the available pool (where another process may use it)

- Better handle misssing comparison file
- cache local ip string
- Capability now there to use dataloop code for flattening, but it is currently disabled (at the top of flatten.c).
- ch_nt: added ad_set_view.c
- ch_nt: added gencheck.c to the project files
- ch_nt: added host name to error output
- ch_nt: added some retry code to avoid thread creation failures
- ch_nt: added synchronization to shmem queue initialization
- ch_nt: added the ability to share memory accross local windows workstation boundaries
- ch_nt: fixed mpptest in the examples\nt directory
- ch_nt: updated error messages
- Change := to = in LIBS assignment
- Change mpptest output to use gnuplot by default
- Change p4_dprintf to p4_dprintfl in p4_sock_cr.c so that messages about environment are not printed by default
- Changed AUX_DIR to point to local version
- changed MPIR_ALLREDUCE_TAG to MPIR_REDUCE_SCATTER_TAG in a few places in reduce_scatter
- changed MPI_File_open to fail on all processes if any process returns error
- changed NULL to MPI_DATATYPE_NULL in ad_init.c
- Changed error message for intracomm expected
- changed faulty var name hostname -> host_name in DEBUG
- Changed max msg queues from 8 to 64 in p4_MD.h
- changed maximum number of processes from 32 to 256
- changed minimum spanning tree to binary tree in algorithm description for collectives
- changes to utilize GridFTP
- Check for PRINT_ERR_MSG defined
- Check for STATUS_IGNORE
- Check for and use errno.h
- Check for and use stdlib.h
- Check for broken autoconf (without support for AM macros)
- Check for f90 include files before trying to install them
- Check for mpichversion.exe for cygwin installations
- Check for needs int64_t definition
- Check for read-only access to the file before attempting to write

- Check the status of most MPI_File operations.. added some commented-out hints we should use whenever we teach coll_test how to handle more command-line arguments than just -fname
- Clean up ROMIO makefile
- Clean up ROMIOs aclocal.m4
- Cleanup use of getopt and use header file when available
- close a small memory leak
- closes romioreq #515: with pete wycoff's suggested fix: don't override user-specified include path.
- configure test for fstypename means we don't have to test for specific bsd versions anymore
- correct the markup for the deferred open hints
- corrected bug in mpirun.globus2 script: -np X with X>255 should work on any platform – bug uncovered by then98@sohu.com, sent to mpi@globus.org on Tue, 25 Nov 2003 13:19:57 +0800 (CST)
- Corrected message about number of allowed processes in test
- Create the links for the sharedlib names for use in-place
- DESTDIR for mpiinstall.in
- declared integer toterrs in isendf.f
- deferred open: now open the file if independent io attempted
- deleted trailing spaces in fortran/configure.in
- Deleted useless symbols: double precision MPE_WTIME, MPE_PTIME
- derrive the 'deferred_open' hint if the "no independent io" and "use collecitve buffering" hints are turned on.
- Detect a wtick of 0
- Do not define MPI_Info_xxx as PMPI_Info_xxx when building the MPI_Info routines in mpi2-other/info
- do not do link tests if building withing an MPI implementation
- Do not fail if we cannot remove a link to mpif.h that we believe we have created
- do not override includedir
- document no_indep_rw a bit better
- documentation to use SLOG-API to add arrow
- don't assume MPI_INFO_NULL is zero. explicitly compare against MPI_INFO_NULL
- don't mung MPI_LIB if you can help it. if ROMIO needs extra (non-mpi) libraries to link, use ROMIO_LIBLIST
- don't set C_OPT_FLAGS if user provides CFLAGS (closes: mpich2-req 238)
- Eliminating common uninitialized symbols in ROMIO common/ad_init.c.
- Ensure that CFLAGS is exported as well as CC in case the user has placed flags that change the compilers behavior in significant ways into CFLAGS instead of CC

- Ensure that MPI->PMPI mapping is used for internal routines
- Ensure that PAC macros are all expanded or not present in generated configure (like AC macro names, they should not even be in comments to simplify error checks for unexpanded macros)
- Ensure that all tests use FFLAGS or F90FLAGS
- Ensure that definitions of HAVE_MPI_INFO have the same value
- Ensure that long long is 8 bytes before using it for int64_t
- Ensure that sockets are only closed if they have been opened (some logic paths might not set end_1 and/or end_2)
- Ensure that status is set correctly for count == 0
- Ensure that temporary test files are removed
- Ensure that the ENABLE_SHLIBS variable is none if not building shared libraries in ROMIO
- Ensure that the Fortran interface libraries have the ranlib step applied
- Filter additional processes from aborttest checks
- fine-tuning hint precedence for deferred open
- first cut at using generalized requests, if avaliable
- Fix Makefile.in so it won't build mpereconfig from mpereconfig.in
- Fix ROMIOs scheme for detecting Fortran naming convention to use - in strings (-a isnt enough under Mac OSX)
- Fix [] bug in C++ autoconf macros (missing changequote statements)
- Fix array length
- Fix begin->end in error handling comment
- Fix for –enable-boot for ch_p4mpd testing
- Fix for MPICH variable in mpeinstall
- Fix for ROMIO under MPICH1
- Fix for exclude and machine files with ncpus specifiers
- Fix for gcc 3.2 weak symbols
- Fix for mpich-1 and romio
- fix for new mpiinstall
- Fix for read of uninitialized storage and error in the macro to check if a send has been cancelled
- Fix for shared libraris in ROMIO
- Fix for some broken automakes
- Fix for the second use of aborttest
- Fix ifdef for verbose output
- Fix incorrect use of config.log
- Fix invokation of romio configure

- Fix long-standing problem with IO tests in test suite
- Fix missing ierr on Allreduce call
- Fix missing semicolon
- Fix problems with dependency generation code
- Fix test for send datatype to apply only at the root process
- fix the unparameterized MPIR_Err_create_code calls
- Fix to allow totalview to work when -nolocal is selected
- Fix to ensure that include path is included in tests
- Fix typo in one of the wtime declarations
- fixed -hosts option bug
- fixed MPI_FILE_NULL warning and configuration bugs in ROMIO Fortran tests
- fixed ROMIO Fortran interface for gcc 3.2
- Fixed a bug in init_g.c caused by globus_duroc_intra_subjob_sendthat put a limit on the number of procs that could be startedANDcommented out "sanity checking" code that prevented MPICH-G2 fromlinking against the Globus Toolkit 2.4.3 libraries that were distributedin GT v3.2.
- fixed a logic problem comparing fd->info to MPI_INFO_NULL
- fixed bug in MPID_Comm_init so that it will now work with new MPI-2 functions MPI_Comm_connect,accept on machines with vMPI.
- fixed bug in MPI_Reduce_scatter
- fixed bug in alltoall
- fixed bug in bnr.c: reset global fence flag
- fixed bug in file_set_errhandler
- Fixed bug in noncontig memory/contig file listio code (not calling status set bytes).Added new combining write functionality into non-listio strided write, removing writev calls.
- fixed bug in reduce scatter
- Fixed bug in semop call when interrupts received
- fixed bug in type_create_darray
- fixed bug in weak symbols test in ROMIO's configure.in
- fixed bug related to MPI_DISPLACEMENT_CURRENT
- fixed bug with 0-size messages in collectives
- Fixed bugs in the new ROMIO configure
- Fixed build for configure due to new files in confdb
- Fixed documentation that refered to MPI_STATUS_NULL or MPI_STATUSES_NULL (should be IGNORE, not NULL)
- Fixed emacs header and added comments about f77 failures

- Fixed error in pmpi version of object comparisons that caused the Solaris C++ compiler to give fatal errors
- fixed list freeing code
- Fixed longstanding typo in a define.
- fixed some error printint statements
- fixed string for executable staging ... added '#' which is RSL concat operator
- Fixed the GCC ever changing treatment of va_arg by changing the test from AC_TRY_COMPILE to AC_TRY_RUN to avoid the unexpected coredump due to non int va_arg.
- Fixed the GCC3XX test
- Fixed the test for the master_top_srcdir
- fixes a segfault in noncontig_coll2 test
- Fixes for ROMIO tests from MPICH test suite
- Fixes for caching and enable strict plus fix for bogus GNU version of head and tail
- Fixes for mpichversion
- Fixes for the multiple completion routines in ROMIO
- fixup handling of profiling target
- Flatten now supports subarray and darray combiners.
- Further improve error reporting from perftest/makeconfigure
- Further update the ROMIO aclocal.m4 for autoconf 2.57
- get value directly from the hint structure instead of parsing the Info object
- getting all Globus header files above all others to fix annoyingOS-based compiler bugs/warnings are various archs ... most notably AIX
- hack around the fact that := isn't avaliable everywhere, and added a .o suffix rule to clean up some of the targets
- Handle more bugs in aclocal
- honor setting romio_ds_write to "disable"
- if indepenent io attempted during deferred open, abort
- If we use generalized requests, these conversion functions don't have to do anything
- Ignore config.cache in ROMIO since autoconf does not correctly detect critical changes to the environment
- implement scalable MPI_File_set_size for pvfs1
- implement scalable sync for pvfs1
- implement set_view as a function, not as fs-specific fcntl.
- Improve filtering for irrelevant processes
- Improve handling of IO tests
- Improve test for alternate libraries to support getarg
- Improve the handling of the IO tests in the mpich test suite when romio is not built

- improved configure to handle missing Fortran compiler
- Improved handling of shared libraries
- Improved spec file
- Increase secure server timeout to 2 seconds and add more debugging output
- Increase wait time in secure server code to 5 seconds
- Increased max keyvals from 64 to 256.
- initial deferred open work. skips the open for now. will actually defer later
- Initial deposit, to get autoheader to work correctly
- Initialize autodx and autorel (not actually needed) to keep some compilers happy
- introduce a 'worker_rank' member to ADIO_FileD.
- Issue a configure-time error message if comm=bproc but bproc is not available
- lam-submitted patch to use 'unsinged int' to hold the umask(2) output
- lam-supplied portability fixes
- Let C++ configure know about versions 1.2.5 and 1.2.6
- look for large file summit headers and defines; add proper magic to CFLAGS if we find them
- MPI_File_seek won't actually seek: we keep track of where we should be and seek before reading or writing. now only has to open the file in the MPI_SEEK_END case
- Made sure that all tests guard variables that might be empty with both a -n test and an && concatenation, along with quotes everywhere
- Made the MPIO_Request routines optional (not built for MPICH2 which should be using generalized requests instead) and added LIBTOOL definitions
- make clean_execer routine have a conforming internal name
- Make error test use standard output format
- Make test for error message less sensitive to the exact wording
- Make the C++ configure understand mpicxx as well as mpiCC
- Make the include of mpiof.h optional
- Make the release level current so that patches can be more easily generated
- Minor fix to HPUX stuff.
- Minor fix to man page for Type_indexed
- minor fixes to MPIR_Err_create_code() calls
- missing prototype for MPI_Status_set_elements
- modified CheckRecvErr() to check the error class in status->MPI_ERROR rather than the whole error code
- Modified ROMIO tests to generate standard output on no errors
- Move BasicTypes to a better place
- Move use of USE_STDARG after the point where it is defined by mpidefs.h

- moved MPID_Type_free from the "default" case to just after thewhole switch statement so that it is now called for _all_ the casesin the switch statement ... not just the "default" case
- Moved compiler selection ahead of other arch-specific tests to workaround a limitation in AC_PROG_CC and anything that requires it, including AC_CHECK_HEADERS
- Moved conf file to top
- must edit mpio.h to define HAVE_MPI_GREQUEST if avaliable
- new error handling
- new versions of pvfs have fewer members in the pvfs_filestat structure. icctreats 'too many initializer values' as an error.
- no need for worker_rank after all. ranklist[0] has what we need
- OS-based compiler bugs/warnings are various archs ... most notably AIX
- one more deferred open file
- Pass CFLAGS to mpd configure to ensure shared library options are propagated
- Pass includedir to mpe configure
- Pass libdir and bindir to mpe configure
- passed the correct size of the buffer into MPI_Buffer_attach()
- Preliminary update of MPE README for inclusion of slog2sdk
- Protect results from pwd step in case it fails
- prototopye for ADIOI_GEN_WriteStrided_naive
- Provide a default if MPIRUN_DEVICE not set
- Provide an ifdef for using Generalized requests in MPI-IO code (e.g., use MPI_Wait instead of MPIO_Wait)
- Provide more detailed error messages and error classes on open failures
- Put #if 0 #endifpair around doing chdir since that is handled by mpd in the ch_p4mpd device.
- Re-establish a connection with the listener before sending the DIE message
- Rearranged AC_PROG_CC to the beginning of configure.in so AC macros that needs CC can be invoked without complaints when running autoconf.
- Remember status return on IO operation so that status field on split collective can be set correctly on the xxx_end routine
- Remove .lo files as well as .o files in clean step
- Remove MPI_Type_count
- Remove bogus setting of default file system
- Remove duplicate ioreq_c2f and ioreq_f2c
- Removed all mpereconfig references. Explicitly creating examplesbuild_dir/Makefile using build_dir/contrib/test/Makefile as a template.
- removed all the symbols that were commented out with #if 0-#endif block

- Removed mpich2_fileutil.c from list of files to compile with the name-shifting interface
- removed mpiimpl.h
- Removed rmprog target for mpereconfig.*
- Removed spurious \ from line
- Removed test for file locks under NFS
- Renamed internal function with p4 prefix
- replaced –enable-slog2source by –enable-slog2=build.
- fixes for the HP bug (closes romio req #570)
- romio: a few more error code fixes
- romio: bandaid fix for mpiimpl.h include problem
- romio: nfs error code updates
- romio: ufs error code updates
- romio: updated error handling for ad_ntfs - needs to be done for all the other ad_xxx's and the common files
- Run cleanipcs if possible after running comm=shared p4 jobs
- SL: corrected bug for allgather, allgatherv, alltoall when called with buffer=NULL and counts=0
- Save output from aborttest for the case where there is a problem
- Set error class to MPI_ERR_ARG for improper filetype
- Set group
- setbuf with a null arg does not always turn off buffering; also add tests that write to socket succeeded
- slog2sdk is installed in -copymode=core
- Some aggregator-aware operations need to know the rank of the 'io_worker', sointroduce a 'worker_rank' member to ADIO_FileD.
- Some enhancements for shared libraries
- Start adding casts to handle compiler warnings about varags and uint_16 items
- statuses[?].MPI_ERROR was not being reset when waiting after a MPI_ERR_PENDING was returned from the MPI_Waitall()
- Support .S as well as .s
- Switch from MPI_ERR_UNSUPPORTED to ERR_ACCESS for access failures
- Switch from MPI_INFO_SRC to HAVE_MPI_INFO_SRC definition name for codes that need the ROMIO implementation of MPI Info
- Switch to using autoconf2 configure for ROMIO
- teach romio about new pvfs2 device
- Test for getwd and only define getwd in terms of getcwd if necessary (to avoid problems with different versions of getcwd)

- that put a limit on the number of procs that could be startedANDcommented out "sanity checking" code that prevented MPICH-G2 fromlinking against the Globus Toolkit 2.4.3 libraries that were distributedin GT v3.2.
- the chek for fstypename cannot be AC_TRY_COMPILE. autoconf1.7 uses AC_COMPILE_CHECK
- the fd is inside a request structure
- the fd is part of the request structure, not standing by itself
- Took out line setting ind_wr_buffer_size to -1 in disabled case. Using thatvalue in a more generic sense now.
- Turn off the test for the MPI library when building as part of MPICH
- typedefed the ADIOI_FileD structure
- umode(2) returns a mode_t, not an int. changed type of old_mask accordingly
- Undefine MPI_xxx_c2f and f2c before defining them if necessary
- Update .cvsignore for 2.57 cache
- Update ancient nupshot configure
- Update configuration of MPE for LAM
- Update makeconfigure to be more robust
- update man page comment
- updated #define file to rename global symbols
- updated ROMIO code to take into account deletion of SGI files
- updated ROMIO's configure for weak symbols
- Updated User's Guide to include additional information on ROMIO and NFS.
- updated algo cutoff point in bcast
- updated algorithms for allgather, allgatherv, and alltoall
- updated bcast algorithm
- updated collective algorithms
- Updated configure.in in MPE for better autoheader and for mpich2
- updated description of collective algorithms
- Updated java search paths
- updated romio for mpich-2
- Updated some error handling in ROMIO
- updated test to use prototype form
- Updated the config.guess script
- Updates to get the ROMIO_HOME srcdir
- Upgrade the PAC_TEST_PROGRAM macro
- Use F90FLAGS instead of FFLAGS for f90 compilation/linking
- Use Fortran to link Fortran programs

- Use IPC_PRIVATE instead of getpid to get a unique id for new sysv mem and semaphores
- use LDFLAGS and LIBS if provided by the environment variables
- Use MPI_Wait for requests when generalized requests available for ROMIO
- Use XSL Transform xmlns for stylesheet
- Use a more careful check for problems in MPI_Wtime
- use default cflags on alpha that work with either gcc or native compiler
- Use distinct send and receive buffers
- use mpicxx in preference to mpiCC for Mac OS X
- Use mpicxx instead of mpiCC in examples/basic makefile
- Use relative links
- Use rm instead of /bin/rm for consistency with the other packages
- Use rm instead of /bin/rm in MPE makefiles
- use the PVFS readstrided function so we can use listio
- use the proper type for oldtype_extent
- use the real function name, since myname isn't always avaliable
- Used env perl instead of a hard-coded perl path
- Used environmental variables in enabling features of slog2sdk.
- we now have a PVFS2_SUPER_MAGIC
- winmpd: added -localroot option to mpirun to spawn the root process w/o mpd. This allows the root process to create visible windows and it allows the debugger to be attached to the root process
- winmpd: added -nocache option to mpiregister
- winmpd: added -nodetails flag to mpiregister
- winmpd: added TCP_NODELAY to easy_sockets
- winmpd: added check for WSAEINTT and WSAEINPROGRESS as non-fatal errors
- winmpd: added check for error return value from ContextWriteString
- winmpd: added checking of wall clock time to verify that functions don't timeout too early
- winmpd: added error checking
- winmpd: added host name to error output
- winmpd: added new microsoft project files
- winmpd: added smart caching to mpdupdate
- winmpd: added smpd
- winmpd: added the ability to specify the root host and the output redirection host names as ip addresses instead of host names
- winmpd: added timeout to stat command to prevent hangs

- winmpd: added try/catch to prevent crashing while traversing a bogus list
- winmpd: better tmp file cleanup for -localonly option
- winmpd: changed function call so it can still run on WindowsNT
- winmpd: fixed -priority option
- winmpd: fixed buffer overrun
- winmpd: fixed errors caught by Insure
- winmpd: fixed errors detected by insure++
- winmpd: fixed launch counter by atomically incrementing it
- winmpd: fixed multiply defined symbols error
- winmpd: fixed redirectio port and ip string lookup
- winmpd: made IsHostLocal() more robust and misc updates
- winmpd: made the thread creation code more reliable
- winmpd: more robust io thread creation
- winmpd: mpirun now can launch both mpich1 and mpich2 applications with the addition of two new flags -mpich1 and -mpich2. The default is mpich1
- winmpd: mpirun now uses the mpd database to determine if mpich2 applications have made it to mpi_finalize
- winmpd: prints error code along with string
- winmpd: remove cached password when 'mpiregister -remove' issued
- winmpd: remove current user registry entry when uninstalling mpd
- winmpd: reordered launch result message to avoid a potential timeout
- winmpd: replaced stricmp with getopt
- winmpd: reset variables after freeing to prevent use after release
- winmpd: robust io thread creation
- winmpd: the update tool now puts the new mpich dlls in the system32 directory if it cannot find the old dlls in the path
- winmpd: update now always gets the files from the ftp site ignoring locally cached copies

G Deprecated Features

During the development of `MPICH`, various features were developed for the installation and use of `MPICH`. Some of these have been superceeded by newer features that are described above. This section archives the documentation on the deprecated features.

G.1 Getting Tcl, Tk, and wish

These software packages are available by anonymous `ftp` from `ftp.scriptics.com/pub/tcl/tcl\_old`. They are needed only for the `upshot` and `nupshot` programs; you do not need them in order to install `MPICH`. If you cannot find them at `ftp.scriptics.com`, copies of Tcl and Tk are available at `ftp://ftp.mcs.anl.gov/mpi/tcltk`.

You should get `tcl7.3.tar.Z` and `tk3.6.tar.Z` (and patch `tk3.6p1.patch`). Later versions of both Tcl and Tk are incompatible with these and do not work with `nupshot`. The `upshot` program has been modified to work with either Tk 3.6 or Tk 4.0. `Upshot` may work with later versions, but we are no longer tracking the changes to Tcl/Tk.

It is necessary that the `wish` program be accessible to users; the other parts of Tcl and Tk do not need to be installed (but make sure that everything that `wish` itself needs is installed).

To build Tcl and Tk, we recommend the following approach:

1. Fetch the compressed tar files and the patch file into an empty directory, preferably in a local (not NFS) file system such as '`/tmp`' (but make sure that you have enough space in that file system; xxx should be adequate).

2. Unpack the tar files:

```
gunzip -c tcl7.3.tar.Z | tar xf -
gunzip -c tk3.6.tar.Z | tar xf -
```

3. Apply the patch to Tk:

```
cd tk3.6
patch -p 1 < ../tk3.6p1patch
cd ..
```

(Note that the instructions say to use `patch -p`; newer versions of patch require an argument and the correct value in this case is one; other versions of patch will want `-p1` (no space between p and the one).)

4. Configure Tcl. Pick an installation directory that clearly indicates the Tcl and Tk versions. For example, to build Tcl to install into '`/usr/local/tcl73tk36`', use

```
cd tcl7.6
./configure -prefix=/usr/local/tcl73tk36
```

5. Build and install Tcl. Before you execute the `make install` step, make sure that the directory specified in the `-prefix` argument to `configure` exists.

```
mkdir /usr/local/tcl73tk36
make
make install
```

6. Configure, build, and install Tk. Use the same installation directory for Tk that you used for Tcl:

   ```
   cd ../tcl7.6
   ./configure -prefix=/usr/local/tcl73tk36
   make
   make install
   ```

This will provide you with a Tcl and Tk installation that can be used with the Tcl and Tk tools provided with MPICH. If you have installed these into a non-standard location (such as the one used above), you can set the environment variable TCL73TK36_DIR to the location used as the prefix in the configure commands:

```
setenv TCL73TK36_DIR /usr/local/tcl73tk36
```

This will allow MPICH to find these versions of Tcl and Tk. We no longer use or recommend Tcl/Tk because of the lack of compatibility between versions of Tcl/Tk.

G.2 Obsolete Systems

To `configure` for the Intel Paragon, use

```
configure --with-device=ch_nx --with-arch=paragon
```

Two troubleshootings tip for the Paragon are

1. **Q:** I got the following messages when I tried to build on the Paragon:

   ```
   PGC-W-0115-Duplicate standard type (init.c: 576)
   PGC/Paragon Paragon Rel R5.0: compilation completed with warnings
   PGC-W-0115-Duplicate standard type (init.c: 576)
   PGC/Paragon Paragon Rel R5.0: compilation completed with warnings
   ```

   A: This is because the compiler doesn't handle `long long` but doesn't reject it either. It causes no harm.

2. **Q:** I get errors compiling or running Fortran programs.

   A:

   Fortran programs will need to use a *absolute* path for the '`mpif.h`' include file, due to a bug in the `if77` compiler (it searches include directories in the wrong order).

A troubleshooting tip for the older Intel i860 system is

1. **Q:** The link test fails on an Intel i860 with

```
    icc  -o overtake overtake.o test.o -L/mpich/lib/intelnx/  -lmpi  -lnode
/usr/ipsc/XDEV/i860/bin/ld860: Error: undefined symbol '_MPI_Keyval_create'
/usr/ipsc/XDEV/i860/bin/ld860: Fatal: no output file created
```

A: You are probably building `MPICH` on an old 386 running System V release 2. This version of Unix has very severe limitations on the length of filenames (more severe than we are willing to cater to). The specific problem here is that the name of the file '`mpich/src/context/keyval_create.c`' is too long for this system, and was not properly archived. You best bet is to build `MPICH` on a different, more modern system (for example, a Sun running SunOS or Solaris).

G.3 More detailed control over compiling and linking

Much of `MPICH`'s portability is handled throught the careful construction of system-dependant Makefiles by the `configure` program. This is fine for installing `MPICH`, but what can you do when you are building a new application? For simple applications, the `mpicc` and `mpif77` commands may be the simplest way to build a new application. For more complex codes, we recommend taking a sample '`Makefile.in`' file, for example, in '`mpich/examples/test/pt2pt`'. Modify those parts that are are relavent, such as the `EXECS` and specific program targets. To create a '`Makefile`', just execute

```
mpireconfig Makefile
```

(`mpireconfig` is in the same directory as `mpirun`). This generates a new '`Makefile`' from '`Makefile.in`', with the correct parameters for the `MPICH` that was installed.

References

[1] James Boyle, Ralph Butler, Terrence Disz, Barnett Glickfeld, Ewing Lusk, Ross Overbeek, James Patterson, and Rick Stevens. *Portable Programs for Parallel Processors.* Holt, Rinehart, and Winston, New York, NY, 1987.

[2] Anthony Chan, William Gropp, and Ewing Lusk. User's guide for `mpe` extensions for mpi programs. Technical Report ANL-98/xx, Argonne National Laboratory, 1998. The updated version is at `ftp://ftp.mcs.anl.gov/pub/mpi/mpeman.ps`.

[3] James Cownie and William Gropp. A standard interface for debugger access to message queue information in MPI. In Jack Dongarra, Emilio Luque, and Tomàs Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 51–58. Springer Verlag, 1999.

[4] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[5] William Gropp, David Gunter, and Valerie Taylor. FPMPI: A fine-tuning performance profiling library for MPI, November 2001. Poster presented at SC2001.

[6] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI—The Complete Reference: Volume 2, The MPI-2 Extensions.* MIT Press, Cambridge, MA, 1998.

[7] William Gropp and Ewing Lusk. A high-performance MPI implementation on a shared-memory vector supercomputer. *Parallel Computing*, 22(11):1513–1526, January 1997.

[8] William Gropp and Ewing Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. *IJSA*, 11(2):103–114, Summer 1997.

[9] William Gropp and Ewing Lusk. Installation and user's guide for `mpich`, a portable implementation of MPI. Technical Report ANL-01/x, Argonne National Laboratory, 2001. The updated version is at `ftp://ftp.mcs.anl.gov/pub/mpi/mpichman.ps`.

[10] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI Message-Passing Interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[11] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface,* 2nd edition. MIT Press, Cambridge, MA, 1999.

[12] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface.* MIT Press, Cambridge, MA, 1999.

[13] William D. Gropp and Ewing Lusk. Reproducible measurements of MPI performance characteristics. In Jack Dongarra, Emilio Luque, and Tomàs Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 11–18. Springer Verlag, 1999.

[14] William D. Gropp and Barry Smith. Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, Argonne, IL, March 1993.

[15] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414, 1994.

[16] Peter S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufman, 1997.

[17] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI—The Complete Reference: Volume 1, The MPI Core,* 2nd edition. MIT Press, Cambridge, MA, 1998.

[18] TotalView Multiprocess Debugger/Analyzer, 2000. http://www.etnus.com/Products/TotalView.