ANL/MCS-TM-ANL-96/5 Rev D

Installation Guide to `mpich`, a Portable Implementation of MPI Version 1.2.2

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 specification for a wide variety of parallel computing environments, including workstation clusters and massively parallel processors (MPPs). `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 guide explains how to compile, test, and install `mpich` and its related tools.

This document describes how to obtain and install `mpich` [11], the portable implementation of the MPI Message-Passing Standard. Details on using the `mpich` implementation are presented in a separate *User's Guide* for `mpich` [8]. Version 1.2.2 of `mpich` is primarily a bug fix and increased portability release, particularly for Linux-based clusters.

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.

Features that were new in 1.2.1 included:

- Improved support for assorted Fortran and Fortran 90 compilers. In particular, a single version of `mpich` can now be built to use several different Fortran compilers; see the installation manual (in doc/install.ps.gz) for details.
- Using a C compiler for MPI programs that use `mpich` that is different from the one that `mpich` was built with is also easier now; see the installation manual.
- Known problems and bugs with this release are documented in the file '`mpich/KnownBugs`'.
- There is an FAQ at `http://www.mcs.anl.gov/mpi/mpich/faq.html` . See this if you get "permission denied", "connection reset by peer", or "poll: protocol failure in circuit setup" when trying to run `mpich`.
- There is a paper on jumpshot available at `ftp://ftp.mcs.anl.gov/pub/mpi/jumpshot.ps.gz` . A paper on MPD is available at `ftp://ftp.mcs.anl.gov/pub/mpd.ps.gz`.

1 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.

1. If you have `gunzip`, get '`mpich.tar.gz`'; otherwise, get '`mpich.tar.Z`' from `http://www.mcs.anl.gov/mpi/mpich/download.html` or by anonymous `ftp` from `ftp.mcs.anl.gov` in the directory `pub/mpi`. (If that file is too big, try getting the pieces from `pub/mpi/mpisplit` and `cat`ing them together.)

2. `gunzip -c mpich.tar.gz | tar xovf -`
   or `tar zxvf mpich.tar.gz` (if using the GNU `tar`)
   or `zcat mpich.tar.Z | tar xovf -` (if `gzip` is unavailable)

3. `cd mpich-1.2.2`

4. `./configure`
   This will attempt to choose an appropriate default architecture and device for you. If the defaults are not what you want, see Section 4. Even better is to pick a directory to install `mpich` into and to configure `mpich` with that directory. For example:

   ```
   ./configure -prefix=/usr/local/mpich-1.2.2
   ```

5. `make >& make.log`
   (in C-shell syntax). This will take a while; depending on the load on your system and on your file server, it may take anywhere from 10 minutes to an hour or more.

6. (Optional) On workstation networks, or to run on a single workstation, edit the file '`mpich/util/machines/machines.xxx`' (where `xxx` is `mpich`'s name for your machine's architecture; you will recognize it) to reflect your local host names for your workstations. If you want to, you can skip this step because by default, five copies of the machine you have built `mpich` on will be there to begin with. On parallel machines, this step is not needed. See the '`README`' file in the '`mpich/util/machines`' directory for a description of the format.

7. (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.

8. (Optional) Put the distribution through its complete acceptance test (See Section 7 for how to do this).

9. (Optional) Build the rest of the `mpich` environment: For the `ch_p4` device, use of the secure server (see Section 6.1.3) can speed job startup. The secure server is built as part of `mpich`.

   The `nupshot` program is a faster version of `upshot`, but requires version 3.6 of the `tk` source code. If you have this package, you can build `nupshot` with

```
make nupshot
```

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

    ```
    make install
    ```

    or

    ```
    bin/mpiinstall
    ```

    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`.

11. (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/mpi`' (or wherever you have installed it). See Section 13 for commands they can use. They can also copy the '`Makefile`' in '`/usr/local/mpi/examples`' and adapt it for their own use.

In the following sections we go through these steps in more detail, and describe other aspects of the `mpich` distribution you might want to explore.

The companion *User's Guide* [8], available in compressed postscript in the '`doc`' subdirectory, gives more information on building and running MPI programs with `mpich`. Both the *Installation Guide* and the *User's Guide* are also available on the Web at `http://www.mcs.anl.gov/mpi/mpich/docs.html`.

2 Obtaining and Unpacking the Distribution

`mpich` can be obtained by anonymous `ftp` from the site `ftp.mcs.anl.gov`. Go to the directory `pub/mpi` and `get` the file `mpich.tar.gz`. This file name is a link to the most recent verstion of `mpich`. Currently it is about nine Megabytes in size. First, choose a directory into which to unpack the tar file. We recommend either using a local (not an NFS) file system; this will speed the process of building `mpich`. The file is a `gzipped` tar file, so it may be unpacked with

```
gunzip -c mpich.tar.gz | tar xovf -
```

or, when using the GNU tar (e.g., under Linux),

```
tar zxvf mpich.tar.gz
```

If you do not have `gunzip`, but do have `uncompress`, then you must get '`mpich.tar.Z`' instead, and use either

```
zcat mpich.tar.Z | tar xovf -
```

or

```
uncompress mpich.tar.Z
tar xvf mpich.tar
```

This will create a single directory called `mpich`, containing in various subdirectories the entire distribution, including all of the source code, some documentation (including this *Guide*), `man` pages, the `mpich` environment described in Section 13, and example programs. In particular, you should see the following files and directories:

`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 *Installation Guide* and the *User's Guide*.

`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 10.2).

`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.

If you have problems, check the `mpich` home page on the Web at `http://www.mcs.anl.gov/mpi/mpich` . This page has pointers to lists of known bugs and patchfiles. If you don't find what you need here, send mail to `mpi-bugs@mcs.anl.gov`.

3 Documentation

This distribution of `mpich` comes with complete `man` pages for the MPI routines and the MPE extensions. The command `mpiman` in '`mpich/bin`' is a good interface to the man pages.[1] The '`mpich/www`' directory contains HTML versions of the man pages for MPI and MPE. The '`mpich/doc`' directory contains this *Installation Guide* and also the *User's Guide*.

4 Configuring `mpich`

The next step is to configure `mpich` for your particular computing environment. `Mpich` can be built for a variety of parallel computers and also for networks of workstations. Parallel computers supported include the IBM SP (using various communication options), the Intel Paragon and IPSC860, HP Exemplar, NEC SX-4, and IBM, SGI, HP, and Sun Multiprocessors. Workstations supported are the Sun4 family (both SunOS and Solaris), Hewlett-Packard, Compaq 3000 and Alpha, IBM RS/6000 family, and SGI. Also supported are Intel x86-based PC clones running the Linux or FreeBSD operating systems. Previous versions of `mpich` supported the Kendall Square KSR-1 and KSR-2, the Meiko CS-2, Thinking Machines CM-5, and nCube. New ports are always pending.

Configuration of `mpich` is done with the `configure` script contained in the top-level directory. This script is automatically generated by the Gnu `autoconf` program (version 1.6, not version 2) from the file `configure.in`, but you do not need to have `autoconf` yourself.

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 A. The most important options are

[1] The `mpiman` command is created by the configure process described later.

--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.

--with-comm=name Select a communication option for the device. Currently only for the ch_p4 device; the values shared (for SMP nodes) and bproc (for Scyld) are supported.

--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).

-automountfix=program This is sometimes necessary for systems with automounter problems (see Section 6.1.2).

-rsh=commandname Set the name of the program used to start remote processes. Only the ch_p4 device uses this, and if no name is provided, configure will attempt to determine the appropropriate program.

In addition, configure makes use of environment variables such as MAKE, CC, F77, 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.

Mpich is implemented using an abstract device specification (ADI), described in [6]. In some environments, this abstract device is configured to be the native communication subsystem of the machine. This is done with the --with-device argument to configure. For the many other environments, a generic communication device is constructed using p4 [1, 2] and that is used as the instantiation of the ADI. In these cases, use ch_p4 as the device.

The ARCH_TYPE specifies what kind of processor the compilations will take place on. Valid ones are listed above. For the IBM SP, the architecture type is rs6000. If not given, configure will attempt to determine the type.

Some machines have multiple communication options, which are specified with the comm argument. Currently, the ch_p4 device makes use of this. By selecting -comm=shared, a version of the ch_p4 device that permits the use of both shared memory and IP/TCP is built. This is particularly useful on clusters of symmetric multiprocessors.

A new device, globus2 [3, 5], based on the Globus run-time system [4] is currently available. Like the ch_p4 device, the Globus2 device is able to use multimethod communication on some platforms. For example, the IBM SP can communicate via IBM's MPI or TCP depending on which node it is communicating with. See `http://www.globus.org` for more details.

Some sample invocations of `configure` are shown below. In most cases, the `mpe` libraries are also built. (See Section 9 for more information about installing MPE and the *User's Guide for* `mpich` for more information on using the features in MPE.) To build without the `mpe` libraries, configure with `--without-mpe`. In many cases, the detailed invocations below are the defaults, which you would get if you invoked `configure` with no arguments. That is, a good general strategy is to first try

```
./configure
```

If that doesn't work, look over the following for an environment similar to yours.

4.1 Workstations in General

While the default options are often adequate, the recommendations of this section may help `mpich` make better use of the specific facilities provided by these systems.

`Mpich` can be run on a heterogeneous network of workstations of various kinds. For simple collections of workstations, the `mpirun` command can be used; more complex collections of heterogeneous machines require a `p4` "procgroup file" (for the `ch_p4` device) or a "RSL file" (for the `globus2` device). The format of the "procgroup" file is described in Section 6.1. The format of "RSL files" can be found in the Globus documentation found at `http://www.globus.org` under the Resource Specification Language.

The `ch_p4` device is most easily used when all workstations share a common file system. MPI executables should reside in the shared file system. However, a shared file system is not necessary. By using the "procgroup" file, the location of the executable on each workstation can be specified as a different location.

Compaq Alpha If all of your workstations are from Compaq, you may want to use Compaq's own MPI. If you are using `mpich`, in order to get the full advantages of ANSI C, you may need to add `-cflags="-std"`. For strict ANSI C, use `-cflags="-std1"`.

IBM RS6000 In order to get the full advantages of ANSI C, you may need to add `-cflags="-qlanglvl=ansi"`. Currently, `mpich` has not been tested with 64 bit mode on RS6000 workstations; previous versions of `mpich`, when using the `ch_mpl` or `ch_p4` devices, have had problems with 64-bit AIX (`mpich` works with other 64-bit operating systems[2].)

SGI `configure`

Some SGI systems support both 32 and 64 bit pointers (addresses). `Mpich` uses the architecture `IRIX` to refer to 32 bit systems and `IRIX64` for 64 bit systems. `Mpich` will attempt to detect the appropriate architecture automatically, but you can force a choice by specifying the architecture with the configure options

[2]This will be fixed in a subsequent release of `mpich`, subject to the availability of a system on which to test.

--with-arch=IRIX, --with-arch=IRIXN32, --with-arch=IRIX32, or --with-arch=IRIX64. The last three of these correspond to the SGI compiler options -n32, -32, and -64. Make sure to attach the flags to the compilers and linker using environment variables as follows:

```
setenv CC "cc -64"
setenv FC "f77 -64"
configure ...
```

4.2 Workstation Networks with the ch_p4 device

Many users of mpich will be using a Beowulf cluster, typically running Linux (Windows NT users should consult the installations instructions for the NT version of mpich). The ch_p4 device is one of two devices that are appropriate for Beowulf and other clusters; the other is the ch_p4mpd device described in Section 4.3.

Linux The ch_p4 device will be chosen by default. Using shared memory with -comm=shared is supported as of mpich version 1.2.0 through Unix System V IPC's. Use of mmap is not supported, as Linux does not support the use of MAP_SHARED with MAP_ANONYMOUS. Because the System V IPC's cannot (by design) reliably be freed by an application when it is done with them, you may want to use the cleanipcs command provided with mpich.

FreeBSD For a network of PC's running the FreeBSD version of Unix:

```
configure --with-device=ch_p4 --with-arch=freebsd
```

SGI multiprocessor (such as an Onyx or Origin 2000), using the shared memory for fast message-passing

```
configure --with-device=ch_p4 -comm=shared
```

Use --with-arch=IRIX to force 32 bit pointers and --with-arch=IRIX64 to force 64 bit pointers.

Sun SunOS `configure --with-device=ch_p4 --with-arch=sun4`

Mpich now requires a compiler that supports ANSI C prototypes. The old SunOS bundled C compiler does not support prototypes. If you need a compiler that supports prototypes, we recommend the GNU C compiler (gcc).

Sun Solaris `configure --with-device=ch_p4 --with-arch=solaris`

Compaq Alpha `configure --with-device=ch_p4 --with-arch=alpha`

Fujitsu For a network of Fujitsu M780s running UXP/M, the following options have been tested:

```
setenv FC frt
configure --with-arch=UXPM --with-device=ch_p4 \
    -fflags="-Oe,-Uep -Eml -Aabe" \
    -with-mpe -mpedbg -prefix=/usr/local/mpi \
    -tcldir=/usr/local -tkdir=/usr/local -wish=/usr/local/bin/wish
```

HP HPUX For a network of HP's, including the `mpe` library but leaving out of it the MPE X graphics routines:

```
configure --with-device=ch_p4 -arch=hpux --with-mpe -no_mpegraphics
```

4.3 Workstation Networks with the `ch_p4mpd` device

4.4 Computational Grids with the `globus2` device

Before configuring for the `globus2` device, a version of Globus must already be installed[3]. You will need to know the directory where Globus is installed (e.g., `/usr/local/globus`). Set the environment `GLOBUS_INSTALL_PATH` to that directory, for example,

```
setenv GLOBUS_INSTALL_PATH /usr/local/globus
```

When configuring for the `globus2` device, you may specify one of the Globus *flavors* (e.g., mpi, debug or nodebug, threads, 32- or 64-bit, etc.). To see the complete list of *all* Globus flavors (not all may be installed on your machine) use

```
$GLOBUS_INSTALL_PATH/bin/globus-development-path -help
```

The flavors that are available to you (i.e., installed on your machine) are enumerated as directories in `$GLOBUS_INSTALL_PATH/development`. For example, Globus installation on a Solaris workstation might have the following flavors:

```
sparc-sun-solaris2.7_nothreads_standard_debug/
sparc-sun-solaris2.7_pthreads_standard_debug/
sparc-sun-solaris2.7_solaristhreads_standard_debug/
```

There are two ways to configure for the `globus2` device. Each method selects one of the Globus flavor directories in `$GLOBUS_INSTALL_PATH/development`. The first method is to specify the flavor directory *explicitly*, for example (all on one line):

```
configure --with-device=globus2:-dir=$GLOBUS_INSTALL_PATH/development/sparc-sun-solaris2.7_nothreads_standard_debug
```

Optionally, you may specify the flavor directory *implicitly*,

```
configure --with-device=globus2:-flavor=nothreads,debug
```

Finally, you may simply choose the default flavor (returned by `$GLOBUS_INSTALL_PATH/bin/globus-development-path`)

[3]See `http://www.globus.org` for instructions regarding acquiring and installing Globus.

```
configure --with-device=globus2
```

You must specify `-mpi` to enable vendor-supplied MPI communication for intra-machine messaging. In other words, when configuring on machines that provide vendor implementations of the MPI standard, you must specify `-mpi` for optimal performance. Failing to specify `-mpi` will result in TCP intra-machine communication.

Selecting `-debug` can be helpful during debugging, but can slow down performance. `-nodebug` should be used for debugged production code.

In general, `-nothreads` should be used (the Globus2 device is not multithreaded). You should select a threaded flavor only if you intend to link your MPI application with other modules that require a threaded version of Globus (e.g., you have written a library that uses Nexus which requires threaded handlers). You should *not* select a threaded version of Globus simply because your MPI application is multithreaded.

When Globus was installed, a special '`Makefile`' was automatically generated just for `mpich`. The `mpich` configure uses that file when configuring for the `globus2` device. That special '`Makefile`' contains virtually all the information `mpich` configure needs (include directory paths, special libaries, the names of C and Fortran compiler and linkers, etc.).

4.5 Massively Parallel Processors and Large SMPs

Cray multiprocessor (not a CRAY T3D but, for example, a 4 processor Cray YMP or C90)

```
configure --with-device=ch_p4 --with-arch=CRAY
```

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
```

IBM SP (using the high-performance switch for communication)

```
configure --with-device=ch_mpl --with-arch=rs6000
```

Note that this requires support for the IBM MPL message-passing library. Some recent versions of the IBM SP software may not include support for this older library. In that case, you must use the IBM implementation of MPI.

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.

NEC SX-4 For an NEC SX-4 shared-memory vector multiprocessor, use

```
configure --with-device=ch_lfshmem
```

to get the lock-free shared-memory device described in [9]. Note that this device requires special assembly-language code and/or compiler options in order to operate reliably.

4.6 Building a production mpich

By default, configure sets up mpich to be compiled without optimization and with additional code to help in identifying problems and behaviour of the mpich implementation. Once mpich passes the tests (see Section 7), you may wish to rebuild mpich without the debugging code. This will produce significantly smaller libraries and slightly faster code. To do this, add the options

```
-opt=-O --disable-devdebug
```

to the configure line, and re-run configure and make. You may also include multiple optimization options by inclosing them in quotes:

```
-opt="-O -qarch=pwr2"
```

Be very careful when using high levels of compiler optimization when compiling the ch_shmem, ch_lfshmem, or ch_p4 (when supporting shared memory) devices.

4.7 Preparing mpich for TotalView debugging

In order to examine message queues with TotalView, you must configure mpich with the --enable-debug flag. If and only if you are debugging mpich itself, be sure to configure with the -opt=-g flag as well. (This is not required to debug user code with TotalView and see MPI queues.) Neither of these is the default. Of course, your application program must be compiled and linked with -g to get the most benefit from the debugger.

4.8 Fortran

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 4.8.6 discusses how to support multiple Fortran compilers (e.g., g77 and pgf77) with a single mpich installation.

4.8.1 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.

4.8.2 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.

4.8.3 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 `MPI_OFFSET_KIND` values). 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/f77-nof90
make
make install

setenv FC f90
setenv F90 f90
configure -prefix=/usr/local/mpich-1.2/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 4.8.6.

4.8.4 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` (a script for compiling and linking MPI programs) respectively. 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.

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`.

4.8.5 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.

4.8.6 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 F77 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 F77 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`).

4.9 Special issues for heterogeneous networks and the ch_p4 device

When building `mpich` for a heterogeneous collection of workstations, you may want to configure with the option `-no_short_longs`. This indicates to `mpich` that it should not provide support for the C type `long double`. This can improve performance between systems that have the same datatype lengths for all other types (some Intel x86 machines have 12 byte (80 bits) long doubles; many other systems use either 8 or 16 byte long doubles).

4.10 Setting up rsh

If you are using `rsh` with the `ch_p4` device, you may need to set up your machine to allow the use of `rsh`. You should do this only if you are a system administrator and understand what you are doing *or* you are using an isolated network. For example, if you are using Linux on a collection of computers at your home or at work, and these machines are *not* connected to a larger network, you should follow these directrions. If any of the machines are connected to another network, talk to the administrator of the network about the policy for using `rsh`. Alternately, consider using `ssh` (Section 4.11) or the secure server (Section 6.1.3).

The following explains how to setup a single machine so that it can use `rsh` to itself to start processes. To setup `rsh`, you need to ensure that you have a file '/etc/hosts.equiv' that contains at least

```
localhost
your_machine_name
```

where `your_machine_name` is the name that you've given your machine in '/etc/hosts'.

You may also need to ensure that the files '/etc/hosts.allow' and '/etc/hosts.deny' are empty.

When using a machine that is not networked, for example, a laptop while travelling, you may need to change your network settings. Under some versions of Linux, use the `netcfg` and set `Hostname` to `localhost` and `Domain` to `localdomain`.

4.11 Configuring with ssh

The normal process startup mechanism for the ch_p4 device on networks is rsh. Use of rsh requires that certain permission be set up on the participating machines. On some networks it is undesirable to set permissions that way. The simplest alternative to the use of rsh is ssh (the secure shell). It can be used for secure distributed computing. It requires some setup, described here, but then usage is quite simple.

Here is a set of steps that need to be done before ssh will work properly with mpich.

1. Make sure ssh is installed on your network (which ssh). If it isn't, you can get ssh from http://www.ssh.fi/sshprotocols2/index.html.

2. Create your authentication key.

   ```
   ssh-keygen
   ```

   This will generate a private/public key pair. The private key will be saved in

   ```
   ~/.ssh/identity
   ```

   and the public key will be saved in

   ```
   ~/.ssh/identity.pub
   ```

3. Authorize Access. Place your public key in your ~/.ssh/authorized_keys file. All keys listed in that file are allowed access.

   ```
   cp ~/.ssh/identity.pub ~/.ssh/authorized_keys
   ```

   If the machine you are connecting to does not share a common file system, then ~/.ssh/identity.pub should be copied over to the ~/.ssh/authorized_keys file of the machine you will be connecting to. ssh will insist that authorized_keys have its permissions set so that it is not group writable, so do

   ```
   chmod go-rwx ~/.ssh/authorized_keys
   ```

   This step avoids the need to enter your password each time you want to run a secure shell command.

4. In order to avoid typing in your pass phrase each time ssh is invoked, an ssh-agent needs to be created and your pass phrase added.

   ```
   ssh-agent $SHELL
   ssh-add
   ```

5. Configure with -rsh=ssh, so that the ch_p4 device will use ssh instead of rsh:

   ```
   configure -rsh=ssh
   ```

In case of trouble:

- Make sure that the hosts listed in your 'util/machines/machine.xxxx' are also listed in the '/etc/ssh_known_hosts' file on your network or your '~/.ssh/known_hosts' file in your home directory.
- It is important that /tmp has permissions set to 377, with root as owner and group 0.
- openssh has a -v flag which is very useful for tracking down handshaking problems.

4.12 mpich and 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. However, you may run into some problems with signals. Many thread packages make use of signals such as SIGUSR1 and/or SIGUSR2. By default, the ch_p4 device uses SIGUSR1. If you are using mpich with a thread package that uses SIGUSR1, you will need to reconfigure, adding the argument -listenersig=SIGNAL NAME to the --with-device=ch_p4 line and rebuild mpich. For example,

```
./configure --with-device=ch_p4:-listenersig=SIGUSR2
make
```

4.13 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.

5 Compiling 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 14 to see if there is anything said there about your problem. If not, follow the directions in Section 14.1 for submitting a bug report. To simplify checking for problems, it is a good idea to use

```
make >& 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".

5.1 C++

The C++ support in mpich has been provided by 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 mpi2c++@mpi.nd.edu. Bug reports should also be sent to mpi-bugs@mcs.anl.gov.

5.2 Building multiple devices or architectures

When building more than one version of mpich, for example, to support two different devices or several different architectures, it is important to build each one by configuring with a unique prefix and installing the built mpich before building the next version. For example, to build both a ch_p4 and a ch_shmem version for a collection of Solaris workstations, the following commands should be used:

```
./configure --with-device=ch_p4 -prefix=/usr/local/mpich-1.2.2/solaris/ch_p4
make >& make.log
make install
./configure --with-device=ch_p4 -prefix=/usr/local/mpich-1.2.2/solaris/ch_shmem
make >& make.log
make install
```

This assumes that mpich is to be installed into '/usr/local/mpich-1.2.2', and that the 'make.log' files are checked to ensure that the creation of the libraries succeeeded.

Versions of mpich before 1.2.0 placed the device- and architecture-specific files into directories defined by mpich. With version 1.2.0, mpich follows (almost) the GNU approach to installation. In particular, you can override the choices of most of the directories with standard `configure` options. For example, to change the location of the libraries to '`/usr/local/lib`', add the configure option `-libdir=/usr/local/lib`.

6 Running an MPI Program

In order to make running programs on parallel machines nearly as portable as writing them, the environment distributed with `mpich` contains a script for doing so. It is the `mpirun` command, found in the `mpich/bin` directory, which you might want to add it to your path, with (assuming your shell is the C shell)

```
set path=($path /home/me/mpich/bin)
```

More details on `mpirun` can be found in Section 13.2. If you are going to run on a network of workstations, you will need a `machines.xxxx` file in '`mpich/util/machines`'; see Section 6.1 for details. Systems that use various kinds of filesystem automounters may need to make small changes to these programs; these are detailed in Section 6.1.2.

Some simple MPI programs are in the directory `mpich/examples/basic` and contain a C and a Fortran program for estimating π. So change to that directory and do (assuming that you have added the directory containing `mpirun` to your path)

```
make cpi
mpirun -np 4 cpi
```

to build and run the C version, and

```
make fpi
mpirun -np 4 fpi
```

to build and run the Fortran version. At this point, you have minimally tested your installation of `mpich`. You might also want to check out the performance of MPI on your system. You can do a crude check by running the program `systest`, also found in the `examples/basic` directory. To try it, do:

```
make systest
mpirun -np 2 systest
```

For a more precise benchmark, see Section 12.

6.1 Special Considerations for Running on a Network of Workstations

To run on a network of workstations, you must specify in some way the host names of the machines that you want to run on. This can be done in several ways. These are described in detail in the *Users Guide*. We give a shorter version here.

6.1.1 Using the ch_p4 device

The easiest way is to edit the file '`mpich/util/machines/machines.xxxx`', to contain names of machines of architecture `xxxx`. The `xxxx` matches the `arch` given when `mpich` was configured. Then whenever `mpirun` is executed, the required number of hosts will be selcted from this file for the run. (There is no fancy scheduling; the hosts are selected starting from the top). To run all your MPI processes on a single workstation, just make all the lines in the file the same. A sample '`machines.solaris`' file might look like:

```
mercury
venus
earth
mars
earth
mars
```

The names should be provided in the same format as is output by the `hostname` command. For example, if the result of `hostname` on `earth` is `earth.my.edu` (and similarly for the other names), then the machines file should be

```
mercury.my.edu
venus.my.edu
earth.my.edu
mars.my.edu
earth.my.edu
mars.my.edu
```

For nodes that contain multiple processors, indicate the number of processors by following the name with a colon and the number of processors. For example, if `mars` in the previous example had two processors, then the machines file should be

```
mercury
venus
earth
mars:2
earth
mars:2
```

6.1.2 Dealing with automounters

Automounters are programs that dynamically make file systems available when needed. While this is very convenient, many automounters are unable to recognize the file system names that the automounter itself generates.[4] For example, if a user accesses a file '`/home/me`', the automounter may discover that it needs to mount this file system, and does so in '`/tmp_mnt/home/me`'. Unfortunately, if the automounter on a different system is

[4] Yes, this is nonsensical.

presented with '`/tmp_mnt/home/me`' instead of '`/home/me`', it may not be able to find the file system. This would not be such a problem if commands like `pwd` returned '`/home/me`' instead of '`/tmp_mnt/home/me`'; unfortunately, it is all too easy to get a path that the automounter should, but does not, recognize.

To deal with this problem, `configure` allows you to specify a filter program when you configure with the option `-automountfix=PROGRAM`, where `PROGRAM` is a filter that reads a file path from standard input, makes any changes necessary, and writes the output to standard output. `mpirun` uses this program to help it find necessary files. By default, the value of `PROGRAM` is

```
sed -e s@/tmp_mnt/@/@g
```

This uses the `sed` command to strip the string `/tmp_mnt` from the file name. Simple `sed` scripts like this may be used as long as they do not involve quotes (single or double) or use `%` (these will interfere with the shell commands in configure that do the replacements). If you need more complex processing, use a separate shell script or program.

As another example, some systems will generate paths like

```
/a/thishost/root/home/username/....
```

which are valid only on the machine `thishost`, but also have paths of the form

```
/u/home/username/....
```

that are valid everywhere. For this case, the configure option

```
-automountfix='sed -e s@/a/.\*/home@/u/home@g'
```

will make sure that `mpirun` gets the proper filename.

6.1.3 Faster job startup for the ch_p4 device

When using the `ch_p4` device, it is possible to speedup the process of starting jobs by using the *secure server*. The secure server is a program that runs on the machines listed in the '`machines.xxxx`' file (where `xxxx` is the name of the machine architecture) and that allows programs to start faster. There are two ways to install this program: so that only one user may use it and so all users may use it. No special privileges are required to install the secure server for a single user's use.

To use the secure server, follow these steps:

1. Choose a *port*. This is a number that you will use to identify the secure server (different port numbers may be used to allow multiple secure servers to operate). A good choice is a number over 1000. If you pick a number that is already being used, the server will exit, and you'll have to pick another number. On many systems, you can use the `rpcinfo` command to find out which ports are in use (or reserved). For example, to find the ports in use on host `mysun`, try

```
rpcinfo -p mysun
```

2. Start the secure server. The script '`sbin/chp4_servs`'

   ```
   sbin/chp4_servs -port=n -arch=$ARCH
   ```

   can be used to start the secure servers. This makes use of the remote shell command (`rsh`, `remsh`, or `ssh`) to start the servers; if you cannot use the remote shell command, you will need to log into each system on which you want to start the secure server and start the server manually. The command to start an individual server using port `2345` is

   ```
   serv_p4 -o -p 2345 &
   ```

   For example, if you had chosen a port number of 2345 and were using `Solaris`, then you would give the command

   ```
   sbin/chp4_servs -port=2345 --with-arch=solaris
   ```

   The server will keep a log of its activities in a file with the name '`Secure_Server.Log.xxxx`' in the current directory, where '`xxxx`' is the process id of the process that started the server (note that the server may be running as a child of that initial process).

3. To make use of the secure servers using the `ch_p4` device, you must inform `mpirun` of the port number. You can do this in two ways. The first is to give the `-p4ssport n` option to `mpirun`. For example, if the port is `2345` and you want to run `cpi` on four processors, use

   ```
   mpirun -np 4 -p4ssport 2345 cpi
   ```

   The other way to inform `mpirun` of the secure server is to use the environment variables `MPI_USEP4SSPORT` and `MPI_P4SSPORT`. In the C-shell, you can set these with

   ```
   setenv MPI_USEP4SSPORT yes
   setenv MPI_P4SSPORT 2345
   ```

   The value of `MPI_P4SSPORT` must be the port with which you started the secure servers. When these environment variables are set, no extra options are needed with `mpirun`.

Note that when `mpich` is installed, the secure server and the startup commands are copied into the '`bin`' directory so that users may start their own copies of the server. This is discussed in the *Users Guide*.

6.1.4 Stopping the P4 servers

To stop the servers, their processes must be killed. This is easily done with the Scalable Unix Tools [7] with the command

```
ptfps -all -tn serv_p4 -and -o $LOGNAME -kill INT
```

Alternately, you can log into each system and execute something like

```
ps auxww | egrep "$LOGNAME.*serv_p4"
```

if using a BSD-style `ps`, or

```
ps -flu $LOGNAME | egrep 'serv_p4'
```

if using a System V-style `ps`. The System V style will work only if the command name is short; the System V `ps` only gives you the first 80 characters of the command name, and if it was started with a long (but valid) directory path, the name of the command may have been lost.

An alternative approach is discussed in Section 6.1.5.

6.1.5 Managing the servers

An experimental `perl5` program is provided to help you manage the p4 secure servers. This program is `chkserv`, and is installed in the '`sbin`' directory. You can use this program to check that your servers are running, start up new servers, or stop servers that are running.

Before using this script, you may need to edit it; check that it has appropriate values for `serv_p4`, `portnum`, and `machinelist`; you may also need to set the first line to your version of `perl5`.

To check on the status of your servers, use

```
chkserv -port 2345
```

To restart any servers that have stopped, use

```
chkserv -port 2345 -restart
```

This does not restart servers that are already running; you can use this as a `cron` job every morning to make sure that your servers are running. Note that this uses the same remote shell command that `configure` found; if you can't use that remote shell command to start the process on the remote systems, you'll need to restart the servers by hand. In that case, you can use the output from `chkserv -port 1234` to see which servers need to be restarted.

```
chkserv -port 2345 -kill
```

This contacts all running servers and tells them to exit. It does not use rsh, and can be used on any system.

This software is experimental. If you have comments or suggestions, please send them mpi-bugs@mcs.anl.gov.

6.2 Using the MPD System Daemons with the ch_p4mpd device

The new MPD system, together with its advantages in speed of startup and management of stdio a is described in detail in the companion document to this one, the *User's Guide for MPICH*. Here we briefly discuss the installation process.

6.2.1 Installation

To build mpich with the ch_p4mpd device, configure mpich with

```
configure --with-device=ch_p4mpd -prefix=<installdir> <other options>
```

It is particularly important to specifiy an install directory with the prefix argment (unless you want to use the default installation directory, which is /usr/local), since the ch_p4mpd device must be installed before use.

If you intend to run the MPD daemons as root, then you must configure with --enable-root as well. Then it will be possible for multiple users to use the same set of MPD daemons to start jobs.

After configuration, the usual

```
make
make install
```

will install mpich and the MPD executables in the <installdir>/bin directory, which should be added to your path.

6.2.2 Starting and Managing the MPD Daemons

Running MPI programs with the ch_p4mpd device assumes that the mpd daemon is running on each machine in your cluster. In this section we describe how to start and manage these daemons. The mpd and related executables are built when you build and install MPICH after configuring with

```
--with-device=ch_p4mpd -prefix=<prefix directory> <other options>
```

and are found in <prefix-directory>/bin, which you should ensure is in your path. A set of MPD daemons can be started with the command

```
mpichboot <file> <num>
```

where file is the name of a file containing the host names of your cluster and num is the number of daemons you want to start. The startup script uses rsh to start the daemons, but if it is more convenient, they can be started in other ways. The first one can be started with mpd -t. The first daemon, started in this way, will print out the port it is listening on for new mpds to connect to it. Each subsequent mpd is given a host and port to connect to. The mpichboot script automates this process. At any time you can see what mpds are running by using mpdtrace.

An mpd is identified by its host and a port. A number of commands are used to manage the ring of mpds:

mpdhelp prints this information

mpdcleanup deletes Unix socket files '/tmp/mpd.*' if necessary.

mpdtrace causes each mpd in the ring to respond with a message identifying itself and its neighbors.

mpdringtest count sends a message around the ring "count" times and times it

mpdshutdown mpd_id shuts down the specified mpd; mpd_id is specified as host_portnum.

mpdallexit causes all mpds to exit gracefully.

mpdlistjobs lists active jobs managed by mpds in ring.

mpdkilljob job_id aborts the specified job.

Several options control the behavior of the daemons, allowing them to be run either by individual users or by root without conflicts. The current set of command-line options comprises the following:

-h <host to connect to>

-p <port to connect to>

-c allow console (the default)

-n don't allow console

-d <debug (0 or 1)>

-w <working directory>

-l <listener port>

-b background; daemonize

-e don't let this mpd start processes, unless root

-t echo listener port at startup

The -n option allows multiple mpds to be run on a single host by disabling the console on the second and subsequent daemons.

6.3 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[5]. 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
```

6.4 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
   ```

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

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.

7 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
```

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`.

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-1.2.2`', then you should configure with the option `-prefix=/usr/local/mpich-1.2.2`. 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-1.2.2/solaris/ch_p4`, you can later add `-prefix=/usr/local/mpich-1.2.2/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-1.2.2/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-1.2.2
```

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 1.2.2, the installation commands to install into '`/usr/local`' are

```
make install PREFIX=/usr/local/mpi-1.2.2
rm /usr/local/mpi
ln -s /usr/local/mpi-1.2.2 /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`' for architecture `solaris` and device `ch_p4`, execute

```
cd examples/test
./configure -mpichpath=/usr/local/mpich-1.2.2/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.

8.1 User commands

The commands mpirun, mpicc, mpif77, mpiCC, 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-1.2.2/solaris/ch_p4 and -prefix=/usr/local/mpich-1.2.2/solaris/ch_shmem respectively.

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-1.2.2').

8.2.1 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.

8.2.2 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.

9 The MPE Library

The mpe library can be configured and installed as an extension of your current MPI implementation, or automatically during mpich's configure and make process. The only requirement is that you configure with a specific MPI implementation switch. Currently, we have configured the `mpe` library to work with `mpich`, LAM/MPI, SGI's MPI, IBM's MPI, and CRAY's MPI (not thoroughly tested).

9.1 Configure Options

There are 3 types of configure options or switches:

1. MPI implementation switch (mandatory)
2. Generic configure flags (mandatory/optional)
3. User option switches/flags (optional)

By typing

```
./configure --help
```

in the top-level `mpe` directory, a list of flags/switches can be viewed.

Specific MPI implementation switches:

--with-mpich=DIR Specifies the top-level directory where `mpich` (version 1.0.13 or later) was installed

--with-mpichdev=subdir Specifies the subdirectory of which architecture/device you wish to use

--with-lam=DIR Specifies the top-level directory where LAM'S MPI was installed

--with-sp Specifies use of the native IBM POE/MPI implementation

--with-sgi Specifies use of the native SGI MPI implementation

--with-sgi64 Specifies use of the native SGI MPI implementation, forcing it to compile in 64 bit mode

--with-cray Specifies use of the native Cray MPI implementation

If you would like to configure `mpe` with a MPI implementation not listed here, you might want to look at how the '`configure.in`' in the `mpe` directory determines which compilers, libraries, etc. to use for the above MPI implementations. You can then add your own MPI implementation section to '`configure.in`'. Make sure you type

```
autoconf
```

to create a new configure script (The `mpe` configure script was created using `autoconf 2.13` (patched)). You will also need to add an MPI implementation switch. This is achieved through the macro `AC_ARG_WITH`.

Another option is to use the generic MPI switches which lets you specify your own MPI include and library directories.

Generic MPI implementation switches:

--with-mpiinc=MPI_INC Specifes the MPI include directory—for example, `-I/pkgs/MPI/include`

--with-mpilibs=MPI_LIBS Specifies MPI Profiling and MPI libraries—for example, `-L$MPI_LIB_DIR -lfmpich -lpmpich -lmpich`

The following is not a complete list but some of the more common Generic Flags:

-prefix=DIR Specifies the destination install directory for the `mpeinstall` script. If configuring with `mpich`, it must be the same install directory as the one given as an option to `mpich`'s configure or the `mpiinstall` script in the '`mpich/util`' directory. If omitted, and `-prefix` was given as an option to the `mpich` configure, then this directory will automatically be configured. The `mpeinstall` script installs into `DIR` only the required libraries and include files, and a small subset of the examples. (See Section 9.4)

-libdir=DIR Specifies the top-level directory where the `mpe` libraries will be installed. If this directory does not exist, it will be created. This flag is mandatory when not using `mpich` and irrelevant when using `mpich` (`-libdir` is replaced by `-mpichdev`).

-bindir=DIR This is only relevant if you will be installing `jumpshot` along with the `mpe` library. This directory will be passed to `jumpshot`'s configure and will be where `jumpshot`'s executable will be installed. If configuring with `mpich`, this is automatically configured for you. If not, and this flag is omitted, this directory will be '`libdir/../bin`'. If this directory does not exist, it will be created.

--x-includes=DIR This is an optional flag which specifies that X include files are in `DIR`. If omitted, the `mpe` configure will attempt to locate them.

--x-libraries=DIR This is an optional flag which specifies that X library files are in `DIR`. If omitted, the `mpe` configure will attempt to locate them.

User Option Switches:

--enable-echo This switch will turn on strong echoing. The default is enable=no.

--enable-mpe_graphics This switch will allow the `mpe` graphics routines to be built. If disabled, the `mpe` routines that utilize the X11 graphics will not be built. This is appropriate for systems that either do not have the X11 include files or that do not support X11 graphics. The default is enable=yes.

--enable-f77 This switch will allow the compilation of routines that require a Fortran compiler. If configuring with `mpich`, the configure in the top-level `mpich` directory will choose the appropriate value for you. However, it can also be overridden. The default is enable=yes.

--enable-debug This switch turns on the debugging and diagnostic message flags in MPE and SLOG-API code. The default is enable=no.

--enable-jumpshot This switch will allow the configuration of the graphical tool `jumpshot`. The default for this option is enable=yes. If this option is enabled, and you are not configuring with `mpich`, you will also need to supply the directory path where `jumpshot` has already been installed (`--with-jumpshot_home`).

-enable-buildingmpi This switch indicates that MPE is being built as part of a larger build of `mpich`. This turns off tests for the existence of MPI libraries. This switch is not mandatory.

--with-tcldir=TCL_DIR This switch specifies that `tcl` is located in `TCL_DIR`. This can only be version 7 and `TCL_DIR` must have '`lib/libtcl.a`' and '`include/tcl.h`'. These files are only used for `nupshot`. If this switch is omitted, the configure in the `mpe` directory will attempt to locate these files.

--with-tkdir=TK_DIR This switch specifies that `tk` is located in `TK_DIR`. This can only be version 3 if you want to use `nupshot` and `TK_DIR` must have '`lib/libtcl.a`' and '`include/tk.h`'. This may be the same as `TCL_DIR`. If this switch is omitted, the configure in the `mpe` directory will attempt to locate these files.

--with-wishloc=WISHLOC This switch specifies the name of the `tcl/tk` wish executable. If this switch is omitted, the configure in the `mpe` directory will attempt to locate a version. This is used only for `nupshot` and `upshot`. Note: Because `tcl` and `tk` keep changing in incompatible ways, we will eventually be dropping support for any tool that uses `tcl/tk`. The newest version of `upshot`, `jumpshot`, is written in `Java`.

--with-jumpshot_home=JUMP_DIR This switch specifies the path of the top-level directory where `jumpshot` is installed. When configuring with `mpich`, this option is automatically configured by default. However, it can be overriden. If not configuring with `mpich`, then you need to specify the `JUMP_DIR` in order to configure `jumpshot` along with the `mpe` library.

--with-jumpshot_opts=JUMP_OPTS This switch allows you to pass specific options to `jumpshot`'s configure. Unfortunately, because of the way `autoconf 2` processes multiple arguments with '`AC_ARG_WITH`', only 1 option may be passed to `jumpshot`'s configure. If more options are required, then configure `jumpshot` separately (See Section 9.2). To view the `jumpshot` options, go to `jumpshot`'s top-level directory and type `./configure --help` or read the INSTALL in that directory.

--with-slog_home=SLOG_HOME This switch specifies the path of the top-level directory where SLOG_API is installed. When configuring SLOG_API with `mpich`, the option is automatically configured by default. However, it can be overridden here.

--with-flib_path_leader=FLIB_PATH_LEADER This switch shows how to specify a Fortran library path. It is configured by default when configuring with `mpich`.

--with-f77_extra_flag=F77_EXTRA_FLAG This switch is used for Fortran flags that are to be used for compiling but not linking. Currently, this is used for the Absoft compiler -f option. If configuring with `mpich`, this will be determined for you.

--with-cflags=MPE_CFLAGS This is an optional switch for the user to supply extra CFLAGS to the CC compiler.

--with-fflags=MPE_FFLAGS This is an optional switch for the user to supply extra FFLAGS to the Fortran compiler.

9.2 MPE Installation Instructions

As noted in the previous section, the `mpe` library can be installed as part of the `mpich` configure and make process or as an extension of an existing MPI implementation. This section describes the instructions and examples for each type of installation.

9.2.1 Configure the mpe library as part of the `mpich` configure and make process

In this `mpe` installation, no switches or flags are required. The configure in the top-level `mpich` directory will gather the necessary information and pass it to the configures in the `mpe` and `jumpshot` directories. If no switches and flags are given, then the `mpe` library and the graphical tool `jumpshot` will be automatically configured. However, the user can choose to override this by configuring `mpich` with the following options:

```
-mpe_opts=MPE_OPTS
-jumpshot_opts=JUMP_OPTS
```

where MPE_OPTS is one or more of the choices in Section 9.1, and JUMP_OPTS is one of the options in Section 10.2.1. Multiple uses of `-mpe_opts` is allowed to specify several options for the MPE configure.

Example 1: Configure `mpich` with the `mpe` library and `jumpshot`

In the top-level mpich directory,

```
./configure <mpich options>
make
```

Example 2: Configure `mpich` with `tcldir` and `tkdir` given as options to the `mpe` configure

In the top-level mpich directory,

```
./configure <mpich options> \
        -mpe_opts=--with-tcldir=<path of tcldir> \
        -mpe_opts=--with-tkdir=<path of tkdir>
make
```

Example 3: Configure `mpich`, the `mpe` library, and `jumpshot` with an install directory

In the top-level mpich directory,

```
./configure <mpich options> -prefix=<install directory>
make
```

This is useful if you wish to install `mpich`, the `mpe` library, and `jumpshot` in a public place so that others may use it. To install all 3 packages into the install directory, type

```
make install
```

in the top-level mpich directory.

Example 4: Configure `mpich` with the `mpe` library and without `jumpshot`

In the top-level mpich directory,

```
./configure <mpich options> -mpe_opts=--enable-jumpshot=no
make
```

The `jumpshot` configure is invoked through the `mpe` configure. Thus, the way in which to disable the configuration of `jumpshot` is through a configure option to the `mpe` configure. Refer to section 10.2.2 for instructions on how to install `jumpshot` separately.

Example 5: Configure `mpich` without the `mpe` library and `jumpshot`

In the top-level mpich directory,

```
./configure <mpich options> -nompe
make
```

It should be noted here that after `mpich` is configured, it is possible to configure the `mpe` library and `jumpshot` without reconfiguring `mpich`. Or, if `mpich` needs to be reconfigured, there is often no need to reconfigure the `mpe` library or `jumpshot`.

9.2.2 Configure the mpe library as part of an existing MPI implementation

In this `mpe` installation, a specific MPI implementation switch is necessary. Also, if the MPI implementation is not `mpich`, then the generic flag `-libdir` is mandatory.

Example 1: Configure `mpe` with SGI's MPI and without `jumpshot`

In the top-level mpe directory,

```
./configure --with-sgi -libdir=<directory path of libdir>
          or
./configure --with-sgi64 -libdir=<directory path of libdir>
make
```

By not specifying `--with-jumpshot_home` (and not using `mpich`), `jumpshot` does not get configured.

Example 2: Configure `mpe` with IBM's MPI and `jumpshot`

In the top-level mpe directory,

```
./configure --with-sp -libdir=<directory path of libdir> \
                --with-jumpshot_home=<directory path of jumpshot>
make
```

By not specifying `-bindir=DIR`, the `jumpshot` executable will be located in '`libdir/../bin`'. If specification of a particular bin directory is desired, then configure as follows:

In the top-level mpe directory,

```
./configure --with-sp -libdir=<directory path of libdir> \
                --with-jumpshot_home=<directory path of jumpshot> \
                -bindir=<directory path of bindir>
make
```

Example 3: Configure `mpe` with an existing `mpich` implementation and with `jumpshot`

In the top-level mpe directory,

```
./configure --with-mpich=<directory path of MPICH> \
                --with-mpichdev=<library subdirectory for MPICH>
make
```

If your `mpich` implementation has a '`mpich/jumpshot`' subdirectory, there is no need to configure with the option `--with-jumpshot_home`. If not, then this is a necessary configure option.

Example 4: Configure `mpe` with SGI's MPI and pass options to `jumpshot`'s configure

In the top-level mpe directory,

```
./configure --with-sgi -libdir=<directory path of libdir> \
                --with-jumpshot_home=<directory path of jumpshot> \
                --with-jumpshot_opts=<jumpshot option>
make
```

9.3 Example MPE Programs

As previously noted, the `mpe` library is composed of 3 different profiling libraries. Each MPI implementation requires a slightly different way in which to link with these libraries. During configure, the link path and appropriate libraries are determined and assigned to variables. These variables are substituted in the Makefile in the '`mpe/contrib/test`' directory. The following is a list of these variables:

- LOG_LIB = link path needed to link with the logging library
- TRACE_LIB = link path needed to link with the tracing library
- ANIM_LIB = link path needed to link with the animation library
- FLIB_PATH = link path needed to link fortran programs with the logging library

During make, a small C program 'cpi' (in the 'mpe/contrib/test' directory) will be linked with each of the above libraries. In the output from make, a message will be writted regarding the success of each attempted link test. Also, in the 'mpe/contrib/test' directory a small Fortran program 'fpi' will be linked with the logging library using the environment variable FLIB_PATH. The success of this link test will also be included in the make output. If the link tests were successful, then these library paths should be used for your programs as well.

9.4 mpeinstall

A 'mpeinstall' script is created during configuration. If configuring with mpich, then the 'mpiinstall' script invokes the 'mpeinstall' script. However, 'mpeinstall' can also be used by itself. This is only optional and is of use only if you wish to install the mpe library in a public place so that others may use it. Installation will consist of include, lib, bin, and example subdirectories. If jumpshot was configured, the 'mpeinstall' script will place a jumpshot executable in the bin directory.

10 Visualizing Program Behavior

The mpich distribution contains several programs for visualizing log files produced by mpich applications or the mpich library itself. The oldest included in mpich is called upshot, which is a pure tcl/tk application[6]. Its successor is called nupshot, which is faster in displaying the output of large log files since part of it is written in C. A new program, usable but still under development, is a Java version of upshot, which we call jumpshot. Upshot and nupshot use an ASCII log file format we call alog. Jumpshot uses a new, binary format called CLOG. Mpich produces CLOG files by default; to get ALOG files set the environment variable MPE_LOG_FORMAT to ALOG. A new, scalable logfile format called SLOG is also supported by mpich. Only the program jumpshot can view SLOG files.

10.1 Upshot

Another program in the 'examples/basic' or 'mpe/contrib/test' subdirectory is cpilog. This program uses some of the routines from the MPE library. If you make it and run it, it will produce a simple log file that can be viewed with the program analysis tool upshot or Jumpshot. Upshot will be installed as part of the mpich installation as long as the -nompe option is not passed to the mpich configure, and the configure in the 'mpe' subdirectory is able to locate tk and wish (these can be passed as options to the mpe configure; see Section 9.2). If the mpe library is being built with a MPI implementation other than mpich, the mpe configure will still need to locate tk and wish.

To use upshot to view a log file, do

```
make cpilog
mpirun -np 4 cpilog
```

[6] An even earlier version of upshot was implemented using Xlib and the Athena widgets

```
upshot cpilog.log
```

10.2 Jumpshot

`Jumpshot` is distributed with 2 script files, '`jumpshot.in`' (for use with Java 1.1.*) and '`jumpshot12.in`' (for use with Java 1.2.*), which are located in the '`mpe/viewers/jumpshot-2/bin`' subdirectory. The purpose of these script files is to set the `CLASSPATH` variable before invoking `jumpshot`. In order to set the `CLASSPATH` varible, the variables `JAVA_HOME` and `JUMPSHOT_HOME` must be determined. The role of configure in the '`jumpshot`' directory is to determine these variables, substitute them into the appropriate `.in` file, create the executable `jumpshot` or `jumpshot12`, and place it in the appropriate bin directory. After setting your path to include this directory, you should be able to invoke `jumpshot` from any directory by typing

```
jumpshot
```

or

```
jumpshot <name of clog file>
```

10.2.1 Configure Options

`Jumpshot` can be configured in 3 ways:

As part of the `mpich` configure. The configure in the `mpich` directory can invoke the configure in the '`mpe`' subdirectory which can invoke the configure in the '`jumpshot`' subdirectory. This is the default way to configure `mpich` but can be overridden by configuring `mpich` with the `-nompe` option.

As part of the `mpe` configure. The configure in the `mpe` directory can invoke the configure in the `jumpshot` directory. This is the default if the `mpe` library was configured with `mpich`. Otherwise, the configure option `--with-jumpshot_home=JUMP_HOME` must be given.

By itself. `Jumpshot` can be configured by typing `configure <configure options>` in the `jumpshot` directory.

By typing

```
./configure --help
```

in the top-level '`jumpshot`' directory, a list of flags/switches can be viewed.

Mandatory configure flags/switches:

--with-bindir=DIR Specifies the directory where `jumpshot`'s executable will be located. Without this flag, `jumpshot` can not be configured. If configuring with `mpich` or with just the `mpe` library. this will be automatically determined by default (but can be overridden).

--with-jumpshot_home=JUMP_HOME Specifies the path of the top-level directory where `jumpshot` is installed. Without this directory, `jumpshot` can not be configured. When configuring with `mpich`, this will be automatically determined by default (but can be overriden).

Optional configure flags/switches:

--enable-echo This switch will turn on strong echoing. The default is no echo.

--with-java=DIR Specifies the path of Java's top-level directory. If omitted, configure will attempt to locate java.

--with-java_version=VERSION Specifies the version of Java. If omitted, configure will attempt to determine your java's version.

10.2.2 Installation Instructions

As noted above, `jumpshot` can be installed as part of `mpich`'s configure, `mpe's` configure, or by itself. Below is the instructions and examples for each type of installation.

Configure jumpshot as part of the `mpich` configure. In this `jumpshot` installation, no switches and flags are required. The configures in the `mpich` and `mpe` directories will locate the necessary information and pass it to the configure in the `jumpshot` directory. If no options are given to the configure in the `mpich` directory, `jumpshot` will be configured with the default values. The user can choose to override this by configuring `mpich` with the following options:

```
-mpe_opts=MPE_OPTS
-jumpshot_opts=JUMP_OPTS
```

where `JUMP_OPTS` is one of the choices in Section 10.2.1 (multiple uses of `-jumpshot_opts` is not allowed to specify more than one option for `jumpshot`'s configure). See Section 9.1 for `MPE_OPTS`.

Example 1: Configure `mpich` with the `mpe` library and `jumpshot`

In the top-level mpich directory,

```
./configure <mpich configure options>
make
```

Example 2: Configure `mpich` without the `mpe` library and with `jumpshot`

Currently if `mpich` is configured without the `mpe` library, `jumpshot` does not get configured (since the `mpe` configure invokes `jumpshot`'s configure). The following are the steps needed to configure `mpich` and `jumpshot` without the `mpe` library.

In the top-level mpich directory,

```
./configure <mpich configure options> -nompe
make
```

In the jumpshot subdirectory,

```
configure --with-bindir=<directory path of mpich/bin> \
  --with-jumpshot_home=<directory path of jumpshot>
```

Example 3: Configure `mpich` with the `mpe` library and with java's directory path given as an option to `jumpshot`'s configure

In the top-level mpich directory,

```
./configure <mpich configure options> \
  -jumpshot_opts=--with-jumpshot_opts=--with-java=<directory path of java>
make
```

It is necessary to include `--with-jumpshot_opts` since `mpich`'s configure will strip off `-jumpshot_opts` and `mpe's` configure will strip off `--with-jumpshot_opts` and pass the option to `jumpshot`'s configure.

Example 4: Configure `mpich` with the `mpe` library and override the option `jumpshot_home` in `jumpshot`'s configure

In the top-level mpich directory,

```
./configure <mpich configure options> \
   -jumpshot_opts=--with-jumpshot_opts=--with-jumpshot_home=DIR
make
```

Configure jumpshot as part of the mpe configure

Example 1: Configure the `mpe` library with `jumpshot` and `mpich` (already configured and installed)

In the top-level mpe directory,

```
./configure --with-mpich=<top-level directory of MPICH> \
            --with-mpichdev=<library subdirectory of MPICH>
make
```

Example 2: Configure the `mpe` library with `jumpshot` and SGI's MPI

In the top-level mpe directory,

```
./configure --with-sgi -libdir=<directory path of libdir> \
            --with-jumpshot_home=<directory path of jumpshot>
make
```

Example 3: Configure the `mpe` library without `jumpshot` and IBM's MPI

In the top-level mpe directory,

```
./configure --with-sp -libdir=<directory path of libdir> \
            --enable-jumpshot=no
make
```

Example 4: Configure the `mpe` library with `java` given as an option to `jumpshot`'s configure

In the top-level mpe directory,

```
./configure <MPI implementation switch> \
      --with-jumpshot_opts=--with-java=<directory path of java>
make
```

Configure jumpshot by itself

Example 1: Configure `jumpshot`

In the top-level jumpshot directory,

```
./configure -bindir=<directory path of bindir> \
                --with-jumpshot_home=<directory path of jumpshot>
```

Example 2: Configure `jumpshot` with strong echoing

In the top-level jumpshot directory,

```
./configure -bindir=<directory path of bindir> \
            --with-jumpshot_home=<directory path of jumpshot> \
            --enable-echo=yes
```

The log file produced by `cpilog` is not very interesting, since `cpi` is such a simple program. Many interesting logfiles can be found in the '`profiling/upshot/logfiles`' subdirectory, or '`jumpshot/lib/logfiles`' subdirectory. The file `cpilog.c` demonstrates how to instrument your own code for producing such logs. The *User's Guide* [8] describes how to link with a version of `mpich` that produces them automatically. For a short description of the programs in the '`examples/basic`' directory, see the '`README`' file there.

10.2.3 Building and Using Jumpshot-3

The corresponding visualization tools for slog files is called jumpshot-3 which is located in `mpich/jumpshot-3`.

Building jumpshot-3:

cd to '`$(MPICH)/jumpshot-3`' and type `configure` at the command line. If either configure returns an error that it could not find a valid version of Java or you would like jumpshot-3 to be configured with a particular version of Java. Do

```
./configure --with-java=/homes/chan/java/jdk117_v3
```

and then type

```
make
```

cd to '`$(MPICH)/jumpshot-3/bin`' to see if the executables `jumpshot` and `slog_print` are there. `slog_print` is the script to run the SLOG Java API to print the information in the logfiles. `jumpshot` is the script to run the jumpshot-3 visualization tool to display slogfile.

If you have any questions, send them to `mpi-bugs@mcs.anl.gov`.

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 a file '`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'.

12 Benchmarking mpich

The mpich/examples/perftest directory contains a sophisticated tool for measuring latency and bandwidth for mpich on your system. To run it, first make sure that mpich was configured with the -mpe option. Then go to mpich/examples/perftest 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'
```

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*

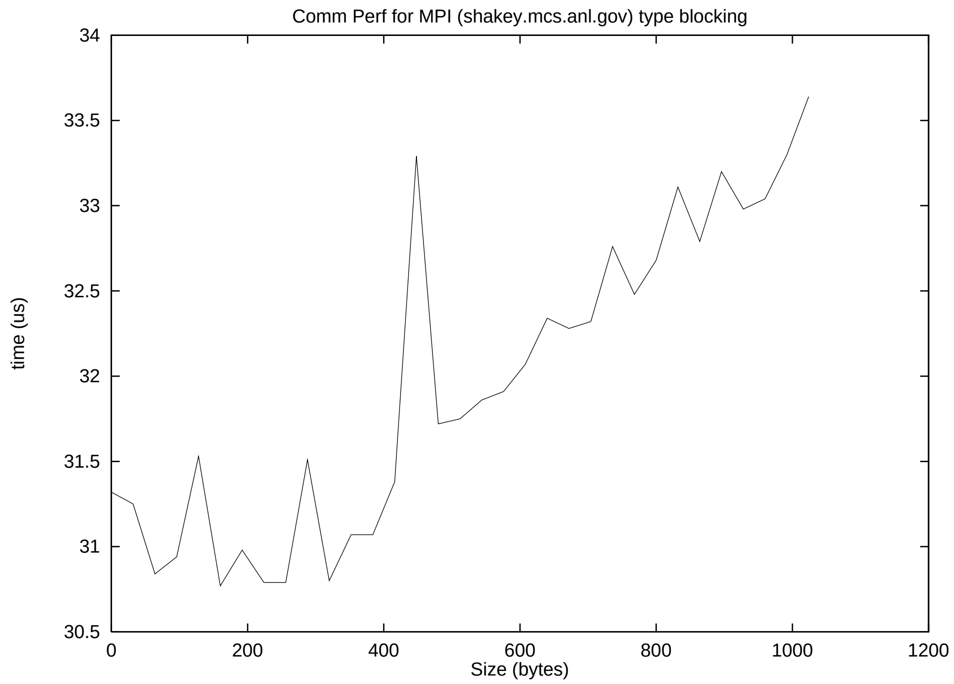

Figure 1: Sample output from `mpptest`

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 [10] discusses these issues at more length.

13 The `mpich` Programming Environment

`Mpich` comes with a number of tools to aid in building, running, and debugging MPI programs. These are described in more detail in the *User's Guide to* `mpich`; a short overview is presented here.

13.1 Introduction

The MPI standard specifies nothing outside of MPI programs, not even how they will be started. `mpich` supplies a number useful tools for managing MPI programs, including

1. `mpirun`, a portable startup command, so that MPI programs can be started the same way in many different environments,
2. `mpicc` and `mpif77`, scripts to compile and link MPI programs in C and Fortran.
3. `mpiCC` and `mpif90`, scripts to compile and link C++ and Fortran 90 programs.

4. `mpe`, a library of useful routines that work will with MPI. Curently this library includes both routines for producing log files of time-stamped events and a simple parallel X graphics library, routines for providing a sequential section code, and routines to start a debugger when errors occur.

5. A set of predefined profiling libraries. The MPI Standard specifies a mechanism whereby the user may "wrap" any collection of MPI functions with code of his own, without accessing the MPI implementation source code. We supply tools for constructing such a profiling version of the MPI library with a minimum of effort, as well as three preconstructed sets of wrappers, for accumulating time spent in MPI routines, for preparing log files, and for program animation.

6. `upshot`, a tool for examining log files produced by the `mpe` logging functions or by the automatic logging in the logging profiling library.

7. `jumpshot`, a Java version of `upshot` and `nupshot`.

13.2 `mpirun`, a Portable Startup Script

Each parallel computing environment provides some mechanism for starting parallel programs. Unfortunately, these mechanisms are very different from one another. In an effort to make this aspect of parallel programming portable as well, `mpich` contains a script called `mpirun`. This is script is partially customized during the configuration process when `mpich` is built. Therefore the actual "source" for `mpirun` is (for most devices) in the file '`mpirun.in`' in the '`mpich/util`' directory; some devices also have additional files in their source directories (e.g., '`mpid/ch_p4`'). The most common invocation of `mpirun` just specifies the number of processes and the program to run:

```
mpirun -np 4 cpi
```

the complete list of options for `mpirun` is obtained by running

```
mpirun -help
```

More details on using `mpirun` may be found in the Users Guide for `mpich`.

13.3 Commands for compiling and linking programs

The `mpich` implementation provides two commands for compiling and linking C, C++, Fortran-77, and Fortran-90 programs. They also have a simple interface to the profiling and visualization libraries described in [12] through these command-line options:

-mpilog Build version that generates MPE log files.

-mpitrace Build version that generates traces.

-mpianim Build version that generates real-time animation.

-show Show the commands that would be used without actually running them.

Use these commands just like the usual compilers. For example,

```
mpicc -c foo.c
mpiCC -c foo.C
mpif77 -c foo.f
mpif90 -c foo.f90
```

and

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

Note that for Fortran 90, different systems may require different suffixes. For example, AIX systems do not support `f90` as a file suffix for Fortran 90 programs.

Commands for the linker may include additional libraries. For example, to use some routines from the MPE library, enter

```
mpicc -o foo foo.o -lmpe
```

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

```
mpicc -o foo foo.c
mpiCC -o foo foo.C
mpif77 -o foo foo.f
mpif90 -o foo foo.f90
```

may not work on some systems.

More information on using these commands may be found in the Users Guide to `mpich`.

14 Problems

This section describes some commonly encountered problems and their solutions. It also describes machine-dependent considerations. You should also check the Users Guide, where problems related to compiling, linking, and running MPI programs (as opposed to building the `mpich` implementation) are discussed.

14.1 Submitting bug reports

Any problem that you can't solve by checking this section should be sent to `mpi-bugs@mcs.anl.gov`. Please include:

- The version of `mpich` (e.g., 1.2.2)
- 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
  ```

- If the problem is with a script like configure or mpirun, run the script with the `-echo` argument (e.g., `mpirun -echo -np 4 a.out`)
- If you are using a network of workstations, also send the output of `sbin/tstmachines`. The program `tstmachines` is discussed in the *Users Guide to* `mpich`.

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 workarounds. 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.

14.2 Problems configuring

14.2.1 General

1. **Q:** When trying to run `configure`, I get error messages like

   ```
   ./configure: syntax error at line 20: '(' unexpected
   ```

   A: You have an obsolete version of the Bourne shell (`sh`). `mpich` requires that the `sh` shell support shell procedures; this has been standard in most Bourne shells for years. To fix this, you might consider (a) getting an update from your vendor or (b) installing one of the many publically available sh-shell replacements.

2. **Q:** The configure reports the compiler as being broken, but there is no problem with the compiler (it runs the test that supposedly failed without trouble).

   A: You may be using the Bash shell ('`/bin/bash`') as a replacement for the Bourne shell ('`/bin/sh`'). We have reports that, at least under Linux, Bash does not properly handle return codes in expressions. One fix is to use a different shell, such as '`/bin/ash`', on those systems.

   This won't work on some Linux systems (*every* shell is broken). We have reports that the following will work:

(a) In 'configure', change `trap 'rm -f confdefs*' 0` to `trap 'rm -f confdefs*' 1`

(b) After configure finishes, remove the file 'confdefs.h' manually.

3. **Q:** configure reports errors of the form

   ```
   checking gnumake... 1: Bad file number
   ```

   A: Some older versions of the `bash` shell do not handle output redirection correctly. Either upgrade your version of `bash` or run configure under another shell (such as `/bin/sh`). Make sure that the version of `sh` that you use is not an alias for `bash`. `configure` trys to detect this situation and will normally issue an error message.

4. **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.

14.2.2 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 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`'.

14.3 Problems building mpich

14.3.1 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/them/burgess/mpich/IRIX32/ch_p4/bin/mpicc \
                                         -o overtake overtake.o test.o
 ld: WARNING 126: The archive \
    /homes/them/burgess/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.

14.3.2 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: 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.

2. **Q:** When building the `ch_p4` device, I get errors of the form

```
making p4 in directory lib
        make  libp4.a
        cc -I../include -I../../../../include   -c p4_globals.c
        cc -I../include -I../../../../include   -c p4_MD.c
        cc -I../include -I../../../../include   -c p4_error.c
cc-142 cc: WARNING File = p4_error.c, Line = 152
  The number of old style and prototype parameters does not agree.
cc-142 cc: WARNING File = p4_error.c, Line = 162
  The number of old style and prototype parameters does not agree.
cc-142 cc: WARNING File = p4_error.c, Line = 169
  The number of old style and prototype parameters does not agree.
cc-142 cc: WARNING File = p4_error.c, Line = 174
  The number of old style and prototype parameters does not agree.
```

A: These have to do with declarations for a signal handler, and can be ignored. Specifically, P4 is using the `SIG_IGN` (ignore signal) and `SIG_DFL` (default behavior) which are defined in '`/usr/include/signal.h`'.

14.3.3 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).

On HP-UX version 10 systems, you can try configuring with `-fflags=+U77` and rebuilding MPICH. This is now the default.

`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`.

14.3.4 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.

14.3.5 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
```

14.3.6 IBM SP

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

```
mpCC  -o overtake overtake.o test.o \
      -L/usr/local/src/Mpi/1.2.0/lib/rs6000/ch_eui   -lmpich
ld: 0711-317 ERROR: Undefined symbol: .mp_main
ld: 0711-317 ERROR: Undefined symbol: .mp_environ
ld: 0711-317 ERROR: Undefined symbol: .mpc_bsend
...
```

   A: Your IBM implementation does not seem to contain the MPL routines that MPICH uses to implement MPI. Your system may contain the IBM version of MPI; you should use that instead.

14.3.7 Compaq ULTRIX

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.

14.4 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.

14.4.1 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`, not yet supported in `mpich`). 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.

   One system known to use `SIGALRM` is the IBM MPL/POE (device `ch_mpl`) software for using the High Performance Switch in the IBM SP parallel computers. Users must *not* use `SIGALRM` on this system.

A Configure Usage

The command `configure -help` will print out

```
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-c++ ] [--disable-c++]
            [--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

You can use --without-<featurename> to turn off a feature (except for
device).
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.

If '-with-cross=file is given, 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.

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, on FreeBSD systems, -make=gnumake may be required because makes derived from BSD 4.4 do not support the include operation (instead using the form .include, unlike all other makes); this is used in the wrappergen utility.

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 '-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.

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.

Sample Configure Usage:

To make for running on sun4's running SunOS with ch_p4 as the device, and with the installation directory equal to the current directory:

```
./configure --with-device=ch_p4 --with-arch=sun4
```

```
  make

Known devices are
        ch_nx     (native Intel NX calls),
        ch_mpl    (native IBM EUI or MPL calls),
        ch_p4     (p4)
        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)
        sun4      (SUN OS 4.x)
        solaris   (Solaris)
        solaris86 (Solaris on Intel platforms)
        hpux      (HP UX)
        sppux     (SPP UX)
        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)
        IRIX      (synonym for sgi)
        IRIX32    (IRIX with 32bit objects -32)
        IRIXN32   (IRIX with -n32)
        IRIX64    (IRIX with 64bit objects)
        alpha     (DEC alpha)
        intelnx   (Intel i860 or Intel Delta)
        paragon   (Intel Paragon)
        tflops    (Intel TFLOPS)
        meiko     (Meiko CS2)
        CRAY      (CRAY XMP, YMP, C90, J90, T90)
        cray_t3d  (CRAY T3D)
        freebsd   (PC clones running FreeBSD)
        netbsd    (PC clones running NetBSD)
        LINUX     (PC clones running LINUX)
        LINUX_ALPHA (Linux on Alpha processors)
        ksr       (Kendall Square KSR1 and KSR2)
        EWS_UX_V  (NEC EWS4800/360AD Series workstation.  Untested.)
        UXPM      (UXP/M. Untested.)
        uxpv      (uxp/v. Untested.)
        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)

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.

Others may be recognized.
```

B 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.

B.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.

B.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).

B.3 `mpireconfig`, a way to create Makefiles

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.

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. More recent extensive contributions have been made by Debbie Swider and (for `Jumpshot`) Omer Zaki. Anthony Chan has helped with SLOG and `Jumpshot-3`. David Ashton, who has developed the Windows NT version of `mpich`, 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.

References

[1] Ralph Butler and Ewing Lusk. User's guide to the p4 parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, October 1992.

[2] Ralph Butler and Ewing Lusk. Monitors, messages, and clusters: The p4 parallel programming system. *Parallel Computing*, 20:547–564, April 1994. (Also Argonne National Laboratory Mathematics and Computer Science Division preprint P362-0493).

[3] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. A wide-area implementation of the Message Passing Interface. *Parallel Computing*, 24(11), 1998.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, 11(2):115–128, 1997.

[5] Ian Foster and Nicholas T. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of SC98*. IEEE, November 1999. `http://www.supercomp.org/sc98`.

[6] William Gropp and Ewing Lusk. An abstract device definition to support the implementation of a high-level message-passing interface. Technical Report MCS-P342-1193, Argonne National Laboratory, 1993.

[7] William Gropp and Ewing Lusk. Scalable Unix tools on parallel processors. In *Proceedings of the Scalable High Performance Computing Conference*, pages 56–62. IEEE, 1994.

[8] William Gropp and Ewing Lusk. User's guide for `mpich`, a portable implementation of MPI. Technical Report ANL-96/6, Argonne National Laboratory, 1996. The updated version is at `ftp://ftp.mcs.anl.gov/pub/mpi/userguide.ps`.

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

[10] William Gropp and Ewing Lusk. Reproducible measurements of MPI performance characteristics. Technical Report ANL/MCS-P755-0699, Mathematics and Computer Science Division, Argonne National Laboratory, June 1999.

[11] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message-passing interface standard. *Parallel Computing*, 22:789–828, 1996.

[12] Edward Karrels and Ewing Lusk. Performance analysis of MPI programs. In Jack Dongarra and Bernard Tourancheau, editors, *Proceedings of the Workshop on Environments and Tools For Parallel Scientific Computing*. SIAM Publications, 1994.