ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL-96/5

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

by

William Gropp and Ewing Lusk

Mathematics and Computer Science Division

July 1996

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

Installation Guide to mpich, a Portable Implementation of MPI

by

William Gropp and Ewing Lusk

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. `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 is a document in progress: hopefully helpful, but hardly whole. Please send suggestions for improvements (or impairments) to `mpi-bugs@mcs.anl.gov`. Details on using the MPICH implementation are presented in a separate users guide for `mpich`.

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` by anonymous `ftp` from `info.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 | tar xovf -` or `zcat mpich.tar.Z | tar xovf -`
3. `cd mpich`
4. `configure -arch=sun4 -device=ch_p4` (for example)
5. `make >& make.log` (in C-shell syntax). Depending on the load on your system and on your file server, this may take anywhere from 10 minutes to an hour or more.
6. To run workstation networks or on a single workstation, edit the file `mpich/util/machines/machines.sample` to reflect your local host names for the machines of the architecture given with `-arch=` above. 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. Build and run a simple test program:

```
cd examples/basic
make cpi
mpirun -np 4 cpi
```

8. `make PREFIX=/usr/local/mpi install` to install MPICH into `/usr/local/mpi`.
9. (Optional) Build the rest of the `mpich` environment: For the `ch_p4` device, use of the secure server (see Section 6.1.2) can speed job startup; you can build it with

   ```
   make serv_p4
   ```

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

   ```
   make nupshot
   ```

At this point you have run an MPI program on your system. In the following sections we go through these steps in more detail and describe other aspects of the `mpich` distribution you might wish to explore.

The companion users guide [5] gives more information on building and running MPI programs with `mpich`.

2 Obtaining and Unpacking the Distribution

`mpich` can be obtained by anonymous `ftp` from the site `info.mcs.anl.gov`. Go to the directory `pub/mpi` and `get` the file `mpich.tar.Z`. This file name is a link to the most recent verstion of `mpich`. Currently it is about 4 megabytes in size. The file is a compressed tar file, so it may unpacked with

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

If your system does not have `zcat`, you must uncompress it in a separate step:

```
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 11, 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`.

README Basic information and instructions for configuring.

aclocal.m4 Used for building `configure` from `configure.in`; not needed for most installations.

bin Home for executable files like `mpirun` and `mpiman`.

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 documentation, including this 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 libraries, both user and system.

installtest A place to test the installation script.

lib The machine-dependent libraries, after they are built. Subdirectories are maintained for each version of the system that is built, so this same tree can be host for multiple installations. Machine-dependent scripts such as `mpirun` and `mpireconfig` are also kept here.

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.

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

profiling The profiling subsystem, including a system for automatically generating the "wrappers" for the MPI profiling interface.

ptx_ifile A file needed by the Sequent Symmetry running the PTX operating system.

ref Postscript versions of the reference manuals.

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.

If you have problems, check the MPICH home page on the Web at http://www.mcs.anl.gov/Projects/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, the MPE extensions, and the ADI (Abstract Device Interface) routines. The command mpiman in mpich/bin is a good interface to the man pages.[1] The mpich/ref directory contains printable versions of the manuals for MPI and the ADI, in compressed PostScript form.

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 SP1 and SP2 (using various communication options), the TMC CM-5, the Intel Paragon, IPSC860, and Touchstone Delta, the Ncube Ncube2, the Meiko CS-2, the Kendall Square KSR-1 and KSR-2, Convex Exemplar, and IBM, SGI and Sun multiprocessors. Workstations supported are the Sun4 family, Hewlett-Packard, DEC 3000 and Alpha, IBM RS/6000 family, and SGI. Also supported are Intel x86-based PC clones running the LINUX or FreeBSD operating systems. 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 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 the following.

```
Configuring with args -usage
Configuring MPICH Version 1.0.13.
Usage: /home/MPI/mpich/configure -arch=ARCH_TYPE -comm=COMM_TYPE -device=DEVICE
                   [-prefix=INSTALL_DIR]  [-c++[=C++_COMPILER]]
                   [-cc=C_COMPILER] [-fc=FORTRAN_COMPILER]
                   [-clinker=C_LINKER] [-flinker=FORTRAN_LINKER]
                   [-mpe] [-nompe] [-nof77] [-f90nag] [-opt=OPTFLAGS]
                   [-make=MAKEPGM]
                   [-cflags=CFLAGS] [-fflags=FFLAGS]
                   [-optcc=C_OPTFLAGS] [-optf77=F77_OPTFLAGS]
                   [-lib=LIBRARY] [-mpilibname=MPINAME]
                   [-no_mpegraphics] [-no_short_longs] [-memdebug]
                   [-x11_lib=X11LIB] [-x11_inc=X11INC]
                   [-mpedbg] [-nompedbg] [-cross] [-devdebug] [-nodevdebug]
                   [-debug] [-nodebug]
                   [-var_pkt] [-pkt_size=LENGTH] [-adi_collective]
                   [-adi_coll_world]
                   [-wish=WISH] [-tcldir=TCLDIR] [-tkdir=TKDIR]
```

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

```
                   [-fortnames=FORTRANNAMES]
                   [-ar_nolocal] [-automountfix=AUTOMOUNTFIX]
                   [-noranlib] [-rsh=RSHCOMMAND] [-rshnol]
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)
   C++_COMPILER = default is to use g++ (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
   WISH         = Name of tcl/tk wish executable.  Configure will attempt
                  to find a version of wish for you, but if there is
                  no wish in your path or you need to use a different version,
                  use this option.  Used only for the display tools
                  (nupshot and upshot).  tk 3.x required for nupshot;
                  tk 3.x or 4.x for upshot.
   TCLDIR       = Directory containing tcl.  Must have lib/libtcl.a and
                  include/tcl.h .  Used only for nupshot.
   TKDIR        = Directory containing tk 3.3, 3.4, 3.5, or 3.6.  Must have
                  lib/libtk.a and include/tk.h .  Used only for nupshot.
                  May be the same as TCLDIR.
   FORTRANNAMES = Form of the Fortran names.  See below.
   X11LIB       = Full path name for libX11.a
   X11INC       = Full path name for X11.h
   AUTOMOUNTFIX = Command to fix automounters
   RSHCOMMAND   = Command to use for remote shell
   MPILIBNAME   = Name to use instead of mpi in the name of the MPI
                  library.  If set, libMPILIBNAME will be used instead
                  or libmpi.  This can be used on systems with
                  several different MPI implementations.
```

One and only one 'arch', 'comm', and 'prefix' argument should be provided. 'arch' MUST be specified before 'comm'.

If '-c++' is included as an option, then the C++ interface is also built. By default, g++ is used as the c++ compiler. THIS IS CURRENTLY UNSUPPORTED.

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 '-cross' is given, configure assumes that you are cross-compiling. If it is not given, configure expects to be able to run programs. Even if '-cross' is not selected, configure will try to determine if you are cross-compiling; this switch is needed only on systems where attempting to run a cross-compiled program causes the configure script to hang.

If '-mpe' is included as an option, then the MPE 'helper' libraries will also be built. '-nompe' causes the MPE libraries to not be built. The default is '-mpe'. If '-no_mpegraphics' is used, then the MPE routines that make use of X11 graphics will NOT be built; this is appropriate for systems that do not have the X11 include files or that do not support X11 graphics (some message-passing systems cannot interoperate with X11). The options -x11_inc and -x11_lib may be used to specify the locations of the X11 include files and libraries in the event that configure cannot find them (they should both be specified in that case).

The option '-mpedbg' enables the '-mpedbg' command line switch in MPI programs. When used with an MPI program, the default error handler (i.e., MPI_COMM_WORLD's error handler) tries to start xterm's running dbx for each process that detects an error. This option is intended primarily for workstation environments but should work on some MPPs (such as IBM SP2).

The option '-nof77' prevents the compilation of routines that require a Fortran compiler. If this option is selected, you may not use the Fortran interface to MPI.

The option '-f90nag' allows you to use the NAG Fortran 90 instead of Fortran 77. This is a preliminary version and is based on the version for NeXTs.

The option '-opt' allows you to specify 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

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 of bugs in the system make.

The option '-no_short_longs' may be used to suppress support for ANSI C types 'long long int' and 'long double' when they are the same size as 'long' and 'double' respectively. Some systems allow these long ANSI 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).

The option '-fortnames=FORTRANNAMES' allows you to specify the form of the

Fortran names. This is used primarily to generate names with and without trailing underscores for those systems that support both. Possible values are

FORTRANNAMES value	if Fortran MPI_SEND looks like
DOUBLEUNDERSCORE	mpi_send__
UNDERSCORE	mpi_send_
CAPS	MPI_SEND
NOUNDERSCORE	mpi_send

This option should normally NOT be used; configure determines what the Fortran compiler generates. This can be used to override that choice.

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 occurred 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).

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., '-device=ch_p4' or '-device=ch_tcp').

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 '-nodevdebug' disables the debugging code in the MPI ADI code. This should be used only when you are sure that everything is working correctly. (This option is also present to remind benchmarkers that the low-level code by default may contain debugging code.) Note also that some of the device code (in mpid/*) has had the debugging code removed from the source code. '-devdebug' turns on the debugging code. '-nodevdebug' is the default.

The option '-var_pkt' allows you to set the message size at which MPICH changes from its short to long message protocol.

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. The option '-adi_coll_world' asks the ADI to try to use any collective operations that are supported only on the MPI_COMM_WORLD communicator (and any communicator with a similiar group). This is also untested.

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 -device=ch_p4 -arch=sun4
  make
```

```
Known devices are chameleon,
        ch_nx     (native Intel NX calls),
        ch_mpl    (native IBM EUI or MPL calls),
        ch_nc     (native nCUBE calls, requires -arch=ncube),
        ch_cmmd   (native TMC CM-5 CMMD calls),
        ch_p4     (p4)
        ch_nexus  (Nexus)
        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),
        meiko     (for Meiko CS2, using elan tport library), and
        nx        (for Intel Paragon),
        t3d          (for the Cray T3D, using Cray shmem library).
```

```
Known architectures include (case is important)
        sun4      (SUN OS 4.x)
        solaris   (Solaris)
        solaris86 (Solaris on Intel platforms)
        hpux      (HP 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)
        IRIX64    (IRIX with 64bit objects)
        alpha     (DEC alpha)
        intelnx   (Intel i860 or Intel Delta)
        paragon   (Intel Paragon)
        meiko     (Meiko CS2)
        CRAY      (CRAY XMP, YMP, C90, J90, T90)
        cray_t3d  (CRAY T3D)
        freebsd   (PC clones running FreeBSD)
        LINUX     (PC clones running LINUX)
        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 (-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.
```

Normally, you should use `configure` with as few arguments as you can. For example, setting the C compiler with `-cc=xxx` may require also setting `-cflags=yyy`; configure will (usually) choose both the compiler and flags appropriately.

`mpich` is implemented using an abstract device specification (ADI), described in [3]. In some environments, this abstract device is configured to be the native communication subsystem of the machine. This is done with the `device` argument to `configure`. For the rest of the 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.

A new device named `ch_nexus` and based on the Nexus run-time system has been made available recently. Like `p4`, Nexus is able to use multimethod communication on many platforms. For example, the IBM SPx binary can communicate via MPL or TCP depending on which node it is communicating with. See http://www.mcs.anl.gov/people/geisler/projects/newmpi.html for more details.

The `ARCH_TYPE` specifies what kind of processor the compilations will take place on. Valid ones are listed above. For the IBM SP1 and SP2, the architecture type is `rs6000`. If the type is not given, `configure` will attempt to determine it.

Some machines have multiple communication options, which are specified with the `comm` argument. Currently, only the `ch_p4` device makes use of this. By selecting `-comm=shared`, a `p4` device that permits the use of both shared memory and IP/TCP is built. This is particularly useful on clusters of symmetric multiprocessors.

Some sample invocations of `configure` are shown below. In most cases, you may wish to add the argument `-mpe` to the `configure` command; this makes the MPE extensions available to your users.

First, for massively parallel processors (MPPs) and multiprocessors:

Convex Exemplar For a Convex Exemplar, please get the official version from Convex/HP. This is based on mpich, but has been tuned for better performance on the Exemplar. If for some reason you wish to use the shared-memory version of `mpich` on the Convex, use

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

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

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

Cray t3d (assuming you are logged into the YMP front end).

```
configure -device=t3d -arch=cray_t3d
```

Intel Paragon `configure -device=nx -arch=paragon`

IBM SP2 (using the high-performance switch for communication)

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

To use this, you must have POE installed.

Meiko CS-2 `configure -device=ch_meiko -arch=meiko`

SGI multiprocessors `configure -device=ch_shmem`

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

For networks of workstations (can interoperate with other types of workstations),

DEC Alpha 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 advantages of ANSI C, you may need to add `-cflags="-qlanglvl=ansi"`. The mpich code uses `__STDC__` to check for the presence of ANSI C features; the IBM RS/6000 compilers do *not* define this by default.

SGI `configure -device=ch_p4`

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. You can use `-arch=IRIX` or `-arch=IRIX64` to force a particular system. 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`.

SGI multiprocessor (such as an Onyx, Challenge, or Power Challenge), using the shared memory for fast message passing

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

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

Sun SunOS including the `mpe` libraries (See the users guide [5]:

```
configure -device=ch_p4 -arch=sun4 -mpe
```

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

DEC Alpha `configure -device=ch_p4 -arch=alpha`

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

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

Nexus device For a machine using the Nexus device, change the device in the above examples to `ch_nexus`. The Nexus device requires an ANSI C compiler, because the Nexus header files use the function prototypes. If the default compiler does not support function prototypes (assuming ansicc is your compiler), add `-cc=ansicc` to the configure command line. Remember to point this at the correct compiler for your message-passing system (e.g., `mpcc` for the SP or `icc` for the Paragon).

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

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

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

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

`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 "startup file" (for the `ch_nexus` device). The format of these files is described in Section 6.1.

4.1 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 behavior 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 -nodevdebug
```

to the `configure` line, and rerun `make`. You may also include multiple optimization options by enclosing them in quotes:

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

4.2 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 `-nof77`. In this case, only the C programs will be built and tested.

4.3 Special Issues for Heterogeneous Networks

When building `mpich` for a heterogeneous collection of workstations, you may wish 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 long doubles; many other systems use either 8- or 16-byte long doubles).

5 Compiling `mpich`

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

```
make
```

This will clean all the directories of previous object files (if any), compile both profiling and nonprofiling versions of the source code, 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 13 to see whether anything is said there about your problem. If not, follow the directions in Section 13.1 for submitting a bug report. To simplify checking for problems, use

```
make >& make.log &
```

Specific (nondefault) targets can also be made. See the `Makefile` to see what they are.

After this `make` is run, the size of the distribution will be about 20 megabytes (depending on the particular machine it is being compiled for), before any of the examples or the extensive test library is built. The `Makefile`s are built for the various example subdirectories, but the example programs have to be made by hand.

5.1 Getting tcl, tk, and wish

The software packages tcl, tk, and wish are available by anonymous ftp from `ftp.smli.com` in the directory `/pub/tcl`. They are needed only for the `upshot` and `nupshot` programs; you do not need them in order to install MPI.

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 tk4.0.

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).

5.2 Building Multiple Devices or Architectures

When building more than one version of `mpich`, it is important to use the versions of the programs `mpirun`, etc., from the library directory, not `mpich/bin`.

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. Several of the examples directories already have symbolic links to this command, but eventually you might wish 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 11.2. 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 6.1.1.

Some simple MPI programs will have been built during the compilation process. They are in the directory mpich/examples/basic and contain a C and a Fortran program for estimating π. Change to that directory, and do

```
    mpirun -np 4 cpi
```

to run the C version, and

```
    mpirun -np 4 fpi
```

to run the Fortran version. At this point, you have minimally tested your installation of mpich. You might also wish to check 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 10.

Another program in the examples/basic subdirectory is cpilog. This program uses some of the routines from the MPE library; you must have configured with the -mpe option. If you make it and run it, it will produce a simple log file that can be viewed with the program analysis tool upshot. To do so, you may need to build upshot with

```
make upshot
```

in the top level mpich directory. Note that upshot requires the tk shell wish. To use upshot to view a log file, do

```
    make cpilog
    mpirun -np 4 cpilog
    upshot cpilog.log
```

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.

The file `cpilog.c` demonstrates how to instrument your own code for producing such logs. The users guide [5] 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. The logging routines are part of `mpe`, so be sure that your configuration has been done with the `-mpe` option.

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 wish to run on. This process can be done in several ways. These are described in detail in the users guide. We give a shorter version here.

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.sun4` file might look like

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

To run the test suite in `examples/test`, you need a machines file with at least five lines in it. This is for homogeneous networks. Heterogeneous networks are discussed in the users guide.

6.1.1 Dealing with Automounters

Automounters are programs that dynamically make file systems available when needed. While very convenient, many automounters are unable to recognize the file system names that the automounter itself generates. 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 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. 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.

6.1.2 Faster Job Startup

When using the `ch_p4` or `ch_nexus` devices, it is possible to speed 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.ARCH` file 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 will 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. If using the `ch_p4` device, build the secure server. From the top-level directory, do

   ```
     make serv_p4
   ```

   At the end of this step, the executable for the secure server is in the same directory as the MPI libraries.

3. Start the secure server. The script `bin/chp4_servs`

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

can be used to start the secure servers. This makes use of the remote shell command (rsh or remsh) 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 sun4s, you would give the command

```
  bin/chp4_servs -port=2345 -arch=sun4
```

The server will keep a log of its activities in a file with the name P4Server.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).

4. 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 wish 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.

5. If using the ch_nexus device, find the Nexus secure server in the Nexus directory, for example, /usr/local/nexus/bin/sserver.

6. Start the Nexus secure server on each machine. The command to start an individual server using port 2345 is

   ```
   ssserver -d -p 2345 &
   ```

7. The ch_nexus device requires that you record the port numbers in a resource database (.rdb) file. The format of the file is

   ```
       <host> ss_port=<port #>
   ```

   The -nexusdb flag should be used to tell mpirun the name of the file:

   ```
       mpirun -nexusdb ports program
   ```

Note that when mpich is installed, the secure server and the startup commands are copied into the library directory so that users may start their own copies of the server. This is discussed in the users guide.

6.1.3 Stopping the Servers

To stop the servers, their processes must be killed. Stopping is easily done with the Scalable Unix Tools [4] with the command

```
pfps -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'
```

and then use the `kill` command on the resulting process number (users of System V-style `ps` commands will have to figure out what their particular form of `ps` needs and adjust the `egrep` command accordingly).

An alternative approach is discussed in Section 6.1.4.

6.1.4 Managing the Servers

An experimental `perl5` program is provided to help you manage the `p4` secure servers. This program is `chkserv`, and is in the `util` 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 *must* edit it. It has sample values for the fields that it will use. In particular, you should set `serv_p4`, `portnum`, and `machinelist` appropriately; 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 `rsh` to start the process on the remote systems; if you can't use `rsh`, 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 (it contacts the server and tells it to exit).

This software is experimental. If you have comments or suggestions, please send them to `mpibugs@mcs.anl.gov`.

6.2 Special Considerations for Running with Shared Memory

When using the `ch_shmem` or `ch_lfshmem` devices with System V shared memory, processes that exit abnormally (e.g., with a segmentation violation) may leave System V semaphores or shared-memory segments allocated.[2] Since there are 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
```

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/examples/test` directory will cause these programs to be compiled, linked, and executed, and their output to be compared with the expected output. Linking all these test programs takes up considerable space, so you might wish 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 your disk space is limited, consider either running with

```
make testing TESTARGS=-small
```

or going to each directory and executing

```
./runtests -small
```

With the `-small` switch, each executable is built, run, and deleted before the next test program is built.

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`.

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

8 Installing mpich for Others to Use

This step is optional. Once you have tested all parts of the MPI distribution (including the tools, particularly upshot and/or nupshot), you can install mpich into a publicly available directory. To install the libraries and include files in a publicly available place, choose a directory, such as /usr/local/mpi, change to the top-level mpich directory, and do

```
    make install PREFIX=/usr/local/mpi
```

The man pages will have been copied with the installation, so you might wish to add /usr/local/mpi/man to the default system MANPATH. The man pages can be conveniently browsed with the mpiman command, found in the mpich/bin 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 1.0.13, the installation commands to install into /usr/local are

```
make install PREFIX=/usr/local/mpi-1.0.13
rm /usr/local/mpi
ln -s /usr/local/mpi-1.0.13 /usr/local/mpi
```

The script util/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 -cpnuphost to install the copy in mpich/profiling/nupshot. Normally, mpiinstall builds a new version of nupshot to ensure 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.

If you are supporting multiple devices on a single platform (for example, ch_p4 and ch_shmem), you should build one and install it in the regular way, then build the second and install it with mpiinstall -libonly. In this case, make sure that your users use the programs in the library directory rather than the bin directory.

8.1 User Commands

The commands mpirun, mpicc, mpif77, mpiman, and mpireconfig 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. For convenience, these are in both mpich/bin and mpich/lib/<arch>/<device>; if there is any chance that multiple architectures or devices are used, the second form should be used.

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/mpi`).

8.2.1 Man Pages

A complete set of Unix man pages for the `mpich` implementation are in `mpich/man`. `man/man3` contains the MPI routines; `man/man4` contains the MPE routines and `mpirun`, and `man/man5` contains the MPID routines (these are for the low-level part of the `mpich` implementation, are are not of interest to users). The command `mpich/bin/mpiman` is a script that runs `xman` on these man pages.

8.2.2 Web Versions of Man Pages

Web (HTML) versions are available from ftp://info.mcs.anl.gov/pub/mpi/manwww.tar.Z. They are available at http://www.mcs.anl.gov/mpi/www. A sample Web page is shown below and is also available in `mpich/util/mpichsite.html`

```
<TITLE>Using MPICH at Argonne</TITLE>

<H1>Site-specific information on the MPICH implementation of MPI</H1>

<H2>Location of libraries</H2>
The MPICH implementation is located in <TT>/usr/local/mpi/lib</TT>; the
libraries are in
<DL>
<DT> <TT>lib/sun4/ch_p4</TT>
<DD> for Sun4

<DT> <TT>lib/rs6000/ch_mpl</TT>
<DD> for the SPx

<DT> <TT>lib/IRIX/ch_shmem</TT>
<DD> for the SGI system
</DL>

<H2>Location of programs</H2>
Programs are located in the same directory as the libraries

<H2>Documentation</H2>
The command <TT>/usr/local/mpi/bin/mpiman</TT> provides man pages with xman.
The <A HREF="http://www.mcs.anl.gov/mpi/www/index.html">man pages</A>
are also available.  The
<A HREF="http://www.mcs.anl.gov/mpi/mpi-report-1.1/mpi-report.html">MPI
Standard Version 1.1</A> is available in hypertext form.
```

```
<H2>Examples</H2>
A simple example in C and Fortran is in <TT>/usr/local/mpi/examples</TT>.
More examples may be found in the MPICH source tree, located at
<TT>/home/MPI/mpich/examples</TT>.
```

8.2.3 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`. Users may be interested in some of the examples that are in the source tree, also in the `examples` directory.

9 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 multilanguage support. This multilanguage support is often called NLS, for National Language Support. `mpich` comes with error messages in 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 an 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 multibyte 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 multibyte 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/lib/rs6000`.

10 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'`.

The programs `mpptest` and `goptest` have a wide variety of capabilities; the option `-help` will list them. For example, `mpptest` can automatically pick message lengths to discover any sudden changes in behavior and can investigate the ability to overlap communication with computation. These programs are written using MPI and may be used with *any* MPI implementation, not just `mpich`.

11 The `mpich` Programming Environment

11.1 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`, easy ways to compile and link MPI programs.
3. `mpireconfig`, a way to create `Makefiles` from `Makefile.in` templates

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. nupshot, an experimental, faster version of upshot.

11.2 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 script is partially customized during the configuration process when mpich is built. Therefore the actual "source" for mpirun is in the file mpirun.sh.in in the mpich/bin directory. 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
```

This is the result:

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

  mpirun_options:
    -arch <architecture>
            specify the architecture (must have matching machines.<arch>
            file in /Net/antireo/antireo9/MPI/mpich/bin/machines) if using the
execer
    -h      This help
    -machine <machine name>
            use startup procedure for <machine name>
            Currently supported:
              chameleon
              meiko
              paragon
```

```
          p4
          sp1
          ibmspx
          anlspx
          ksr
          sgi_mp
          ipsc860
          inteldelta
          cray_t3d
          execer
          smp
          symm_ptx

-machinefile <machine-file name>
        Take the list of possible machines to run on from the
        file <machine-file name>.  This is a list of all available
        machines; use -np <np> to request a specific number of machines.
-np <np>
        specify the number of processors to run on
-nolocal
        don't run on the local machine (only works for
        p4 and ch_p4 jobs)
-stdin filename
        Use filename as the standard input for the program.  This
        is needed for programs that must be run as batch jobs, such
        as some IBM SP systems and Intel Paragons using NQS (see
        -paragontype below).
-t      Testing - do not actually run, just print what would be
        executed
-v      Verbose - throw in some comments
-dbx    Start the first process under dbx where possible
-gdb    Start the first process under gdb where possible
         (on the Meiko, selecting either -dbx or -gdb starts prun
         under totalview instead)
-xxgdb  Start the first process under xxgdb where possible (-xdbx
        does not work)
-tv     Start under totalview

  Special Options for NEC - CENJU-3:

-batch  Excecute program as a batch job (using cjbr)

-stdout filename
        Use filename as the standard output for the program.
-stderr filename
        Use filename as the standard error  for the program.

  Special Options for Nexus device:
```

```
-nexuspg filename
        Use the given Nexus startup file instead of creating one.
        Overrides -np and -nolocal, selects -leave_pg.

-nexusdb filename
        Use the given Nexus resource database.

Special Options for Workstation Clusters:

-e      Use execer to start the program on workstation
        clusters
-pg     Use a procgroup file to start the p4 programs, not execer
        (default)
-leave_pg
        Don't delete the P4 procgroup file after running
-p4pg filename
        Use the given p4 procgroup file instead of creating one.
        Overrides -np and -nolocal, selects -leave_pg.
-tcppg filename
        Use the given tcp procgroup file instead of creating one.
        Overrides -np and -nolocal, selects -leave_pg.
-p4ssport num
        Use the p4 secure server with port number num to start the
        programs.  If num is 0, use the value of the
        environment variable MPI_P4SSPORT.  Using the server can
        speed up process startup.  If MPI_USEP4SSPORT as well as
        MPI_P4SSPORT are set, then that has the effect of giving
        mpirun the -p4ssport 0 parameters.

Special Options for Batch Environments:

-mvhome Move the executable to the home directory.  This
        is needed when all file systems are not cross-mounted
        Currently only used by anlspx
-mvback files
        Move the indicated files back to the current directory.
        Needed only when using -mvhome; has no effect otherwise.
-maxtime min
        Maximum job run time in minutes.  Currently used only
        by anlspx.  Default value is 15 minutes.
-nopoll Do not use a polling-mode communication.
        Available only on IBM SPx.
-mem value
       This is the per node memory request (in Mbytes).  Needed for some
       CM-5s. ( Default 32. )

-cpu time
```

```
            This is the the hard cpu limit used for some CM-5s in
            minutes. (Default  minutes.)

    Special Options for IBM SP2:

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

    Special Options for Intel Paragon:

    -paragontype name
            Selects one of default, mkpart, NQS, depending on how you want
            to submit jobs to a Paragon.

    -paragonname name
            Remote shells to name to run the job (using the -sz method) on
            a Paragon.

    -paragonpn name
            Name of partition to run on in a Paragon (using the -pn name
            command-line argument)

On exit, mpirun returns a status of zero unless mpirun detected a problem, in
which case it returns a non-zero status (currently, all are one, but this
may change in the future).

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

    /home/MPI/mpich/util/mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program

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

    /home/MPI/mpich/util/mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program.%a

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

/home/MPI/mpich/util/mpirun -arch sun4 -np 2 -arch rs6000 -np 3 /tmp/me/%a/program

It is important to specify the architecture with -arch BEFORE specifying
the number of processors.  Also, the FIRST -arch command must refer to the
processor on which the job will be started.  Specifically, if -nolocal is
```

```
NOT specified, then the first -arch must refer to the processor from which
mpirun is running.
```

For backward compatibility with earlier versions of `mpirun`, each of these arguments can also be used with the prefix `mr_`, as in

```
    mpirun -mr_np 4 myprog
```

11.3 The mpicc and mpif77 Commands

The `mpich` implementation provides two commands for compiling and linking C and Fortran programs. You may use these commands *instead of* the `Makefile.in` versions, particularly for programs contained in a small number of files. In addition, they have a simple interface to the profiling and visualization libraries described in [7]. This is a program to compile or link MPI programs. In addition, the following special options are supported:

-mpilog Build the version that generates MPE log files.

-mpitrace Build the version that generates traces.

-mpianim Build the version that generates real-time animation.

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

Use this just like the usual C or Fortran compiler, for example,

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

and

```
    mpicc -o foo foo.o
    mpif77 -o foo foo.o
```

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
    mpif77 -o foo foo.f
```

may not work on some systems, and is not recommended.

These commands are set up for a specific architecture and `mpich` device and are located in the directory that contains the MPI libraries. For example, if the architecture is `sun4` and the device is `ch_p4`, these commands may be found in `/usr/local/mpi/lib/sun4/ch_p4` (assuming that `mpich` is installed in `/usr/local/mpi`).

11.4 `mpireconfig`, a Way to Create Makefiles

Much of `mpich`'s portability is handled throught the careful construction of system-dependent 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 relevant, 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.

11.5 `nupshot`, a Way to View Logfiles

`Nupshot` is a newer version of `upshot` [6] and can display log files created by using either the `-mpilog` option to `mpicc` or `mpif77` or through use of the MPE logfile facilities.

12 Automatic Report Generation

We are working on a prototype system for automatically producing a report describing the installation of MPI on the system you are using. This is done by going to the top-level directory and doing

```
    configure ... >& config.log
    doc/port
```

Warning: this rebuilds the system, since one of the things the report will contain is the set of machine-specific parameters used and the total time it take to build it. The reason for directing the configure output to a file is so that the document generator can include comments on the configuration itself, such as calling out any problems that `configure` noticed.

If the system that you are running on contains all the necessary components for producing the report, it will appear in the file `doc/doc1.tex`. This is a LaTeX file and should be processed in the `doc` directory:

```
cd doc
latex doc1
bibtex
latex doc1
dvips doc1
```

The graphs of performance require `gnuplot` and are generated as Postscript files.

13 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.

13.1 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., 1.0.13)
- 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 `bin/tstmachines`. The program `tstmachines` is discussed in the users guide.

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 problems are peculiar to specific environments and give suggested work-arounds. Each section is organized in question-and-answer format, with questions that relate to more than one environment (workstation, operating system, etc.) first, followed by questions that are specific to a particular environment. Problems with workstation clusters are collected together as well.

13.2 Problems Configuring

13.2.1 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 publicly 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 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` tries to detect this situation and will normally issue an error message.

13.2.2 LINUX

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

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

   A: This is probably caused by a problem in the Fortran compiler in 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 nonzero 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. 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 nonzero 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
   ```

13.3 Problems Building `mpich`

13.3.1 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 ensure 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, a large number of systems have ridiculously small `/tmp` partitions, and making any use of `/tmp` is 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`.

13.3.2 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 whether `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.

13.3.3 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 `-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).

13.3.4 Intel i860

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).

13.3.5 Intel Paragon

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 int` but doesn't reject it either. It causes no harm.

13.3.6 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 -jinstead 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 `Makefiles`). Your only option is to not use the `-O3` option.

13.3.7 LINUX

1. **Q:** The link test failed on LINUX 3.0 with

   ```
   ...
   cc  -o overtake overtake.o test.o -L/usr/local/mpich/lib/LINUX/ch_p4
   -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`.

13.3.8 IBM SP2

1. **Q:** Linking fails on an IBM SP2 for the `ch_mpl` device.

   A: You may have a version of the IBM MPL/POE software that already includes MPI. If that is the case, we recommend that you use that version. If you need to use MPICH, configure with `-mpilibname=mpich` and rebuild MPICH.

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

   ```
           mpCC  -o overtake overtake.o test.o -L/usr/local/src/Mpi/1.0.11/
           lib/rs6000/ch_eui   -lmpi
   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.

   We have been discussing with IBM a way to provide `mpich` on SP systems that also have the IBM MPI; unfortunately, we have not been able to get the information that we need.

   In the interim, you can do the following:

   Change the use of a library path (`-L/usr/local/...`), followed by `-lmpi`, with an explicit file name. For example, instead of

   ```
   -L/usr/local/src/Mpi/1.0.11/lib/rs6000/ch_eui -lmpi
   ```

   do

   ```
   /usr/local/src/Mpi/1.0.11/lib/rs6000/ch_eui/libmpi.a
   ```

   This change needs to be made in the `Makefiles` and in the scripts such as `mpicc`, `mpif77`, and `mpiCC`.

13.3.9 DEC 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 DEC ULTRIX systems. You may be able to work around this with

   ```
   setenv PROG_ENV SYSTEM_FIVE
   ```

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

13.4 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.

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 definition. 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 `SIGALAM` is the IBM MPL/POE (device `ch_mpl`) software for using the High Performance Switch in the IBM SPx parallel computers.

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`.

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 MCS-P362-0493).

[3] William Gropp and Ewing Lusk. An abstract device definition to support the implementation of a high-level message-passing interface. Preprint MCS-P342-1193, Mathematics and Computer Science Division, Argonne National Laboratory, 1993.

[4] 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.

[5] William Gropp and Ewing Lusk. User's guide for `mpich`, a portable implementation of MPI. Technical Report ANL-96/6, Argonne National Laboratory, 1996.

[6] Virginia Herrarte and Ewing Lusk. Studying parallel program behavior with `upshot`. Technical Report ANL–91/15, Argonne National Laboratory, Argonne, 1991.

[7] 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*, pages 195–200. SIAM Publications, 1994.

Distribution for ANL-96/5

Internal:

J. M. Beumer (10)
F. Y. Fradin
W. D. Gropp (10)
E. L. Lusk (10)
G. W. Pieper
R. L. Stevens
C. L. Wilkinson
TIS File

External:

DOE-OSTI, for distribution per UC-405 (52)
ANL-E Library
ANL-W Library
Manager, Chicago Operations Office, DOE
Mathematics and Computer Science Division Review Committee:
- F. Berman, University of California at LaJolla
- G. Cybenko, Dartmouth College
- T. DuPont, The University of Chicago
- J. G. Glimm, State University of New York at Stony Brook
- M. T. Heath, University of Illinois, Urbana
- E. F. Infante, University of Minnesota
- K. Kunen, University of Wisconsin at Madison
- R. E. O'Malley, University of Washington
- L. R. Petzold, University of Minnesota

F. Howes, Dept. of Energy - Office of Computational and Technology Research
D. Nelson, Dept. of Energy - Office of Computational and Technology Research