MPICH2 Installer's Guide*

Version 1.4

Mathematics and Computer Science Division

Argonne National Laboratory

Pavan Balaji
Darius Buntinas
Ralph Butler
Anthony Chan
David Goodell
William Gropp
Jayesh Krishna
Rob Latham
Ewing Lusk
Guillaume Mercier
Rob Ross
Rajeev Thakur

Past Contributors:
David Ashton
Brian Toonen

June 16, 2011

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, SciDAC Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Contents

1 Introduction

This manual describes how to obtain and install MPICH2, the MPI-2 implementation from Argonne National Laboratory. (Of course, if you are reading this, chances are good that you have already obtained it and found this document, among others, in its `doc` subdirectory.) This *Guide* will explain how to install MPICH so that you and others can use it to run MPI applications. Some particular features are different if you have system administration privileges (can become "root" on a Unix system), and these are explained here. It is not necessary to have such privileges to build and install MPICH2. In the event of problems, send mail to `mpich-discuss@mcs.anl.gov`. Once MPICH2 is installed, details on how to run MPI jobs are covered in the *MPICH2 User's Guide*, found in this same `doc` subdirectory.

MPICH2 has many options. We will first go through a recommended, "standard" installation in a step-by-step fashion, and later describe alternative possibilities.

2 Quick Start

In this section we describe a "default" set of installation steps. It uses the default set of configuration options, which builds the `nemesis` communication device and the `Hydra` process manager, for languages C, C++, Fortran-77, and Fortran-90 (if those compilers exist), with compilers chosen automatically from the user's environment, without tracing and debugging options. It uses the `VPATH` feature of `make`, so that the build process can take place on a local disk for speed.

2.1 Prerequisites

For the default installation, you will need:

1. A copy of the distribution, `mpich2.tar.gz`.
2. A C compiler.
3. A Fortran-77, Fortran-90, and/or C++ compiler if you wish to write MPI programs in any of these languages.

4. Any one of a number of Unix operating systems, such as IA32-Linux. MPICH2 is most extensively tested on Linux; there remain some difficulties on systems to which we do not currently have access. Our `configure` script attempts to adapt MPICH2 to new systems.

Configure will check for these prerequisites and try to work around deficiencies if possible. (If you don't have Fortran, you will still be able to use MPICH2, just not with Fortran applications.)

2.2 From A Standing Start to Running an MPI Program

Here are the steps from obtaining MPICH2 through running your own parallel program on multiple machines.

1. Unpack the tar file.

   ```
   tar xfz mpich2.tar.gz
   ```

   If your tar doesn't accept the z option, use

   ```
   gunzip -c mpich2.tar.gz | tar xf -
   ```

   Let us assume that the directory where you do this is `/home/you/libraries`. It will now contain a subdirectory named `mpich2-1.4`.

2. Choose an installation directory (the default is `/usr/local/bin`):

   ```
   mkdir /home/you/mpich2-install
   ```

   It will be most convenient if this directory is shared by all of the machines where you intend to run processes. If not, you will have to duplicate it on the other machines after installation. Actually, if you leave out this step, the next step will create the directory for you.

3. Choose a build directory. Building will proceed *much* faster if your build directory is on a file system local to the machine on which the configuration and compilation steps are executed. It is preferable that this also be separate from the source directory, so that the source directories remain clean and can be reused to build other copies on other machines.

```
mkdir /tmp/you/mpich2-1.4
```

4. Choose any configure options. See Section 2.4.1 for a description of the most important options to consider.

5. Configure MPICH2, specifying the installation directory, and running the `configure` script in the source directory:

```
cd  /tmp/you/mpich2-1.4
/home/you/libraries/mpich2-1.4/configure \
        -prefix=/home/you/mpich2-install |& tee c.txt
```

   where the \ means that this is really one line. (On `sh` and its derivatives, use `2>&1 | tee c.txt` instead of `|& tee c.txt`). Other configure options are described below. Check the `c.txt` file to make sure everything went well. Problems should be self-explanatory, but if not, send `c.txt` to `mpich-discuss@mcs.anl.gov`. The file `config.log` is created by `configure` and contains a record of the tests that `configure` performed. It is normal for some tests recorded in `config.log` to fail.

6. Build MPICH2:

```
make |& tee m.txt       (for csh and tcsh)
  OR
make 2>&1 | tee m.txt   (for bash and sh)
```

   This step should succeed if there were no problems with the preceding step. Check file `m.txt`. If there were problems, do a `make clean` and then run make again with `VERBOSE=1`

```
make VERBOSE=1 |& tee m.txt       (for csh and tcsh)
  OR
make VERBOSE=1 2>&1 | tee m.txt   (for bash and sh)
```

   and then send `m.txt` and `c.txt` to `mpich-discuss@mcs.anl.gov`.

7. Install the MPICH2 commands:

```
make install |& tee mi.txt
```

This step collects all required executables and scripts in the `bin` subdirectory of the directory specified by the prefix argument to configure.

(For users who want an install directory structure compliant to GNU coding standards (i.e., documentation files go to `${datarootdir}/doc/${PACKAGE}`, architecture independent read-only files go to `${datadir}/${PACKAGE}`), replace `make install` by

```
make install PACKAGE=mpich2-<versrion>
```

and corresponding installcheck step should be

```
make installcheck PACKAGE=mpich2-<version>
```

Setting PACKAGE in `make install` or `make installcheck` step is optional and unnecessary for typical MPI users.)

8. Add the `bin` subdirectory of the installation directory to your path:

   ```
   setenv PATH /home/you/mpich2-install/bin:$PATH
   ```

   for `csh` and `tcsh`, or

   ```
   export PATH=/home/you/mpich2-install/bin:$PATH
   ```

   for `bash` and `sh`. Check that everything is in order at this point by doing

   ```
   which mpicc
   which mpiexec
   ```

   All should refer to the commands in the `bin` subdirectory of your install directory. It is at this point that you will need to duplicate this directory on your other machines if it is not in a shared file system such as NFS.

9. Check that you can reach these machines with `ssh` or `rsh` without entering a password. You can test by doing

   ```
   ssh othermachine date
   ```

or

```
rsh othermachine date
```

If you cannot get this to work without entering a password, you will need to configure `ssh` or `rsh` so that this can be done.

10. Test the setup you just created:

    ```
    mpiexec -f machinefile -n <number> hostname
    ```

    The machinefile contains the list of hosts you want to run the executable on.

    ```
    % cat machinefile
       host1     # Run 1 process on host1
       host2:4   # Run 4 processes on host2
       host3:2   # Run 2 processes on host3
       host4:1   # Run 1 process on host4
    ```

11. Now we will run an MPI job, using the `mpiexec` command as specified in the MPI-2 standard.

    As part of the build process for MPICH2, a simple program to compute the value of π by numerical integration is created in the `mpich2-1.4/examples` directory. If the current directory is the top level MPICH2 build directory, then you can run this program with

    ```
    mpiexec -n 5 -f machinefile ./examples/cpi
    ```

    The `cpi` example will tell you which hosts it is running on.

    There are many options for `mpiexec`, by which multiple executables can be run, hosts can be specified, separate command-line arguments and environment variables can be passed to different processes, and working directories and search paths for executables can be specified. Do

    ```
    mpiexec --help
    ```

    for details. A typical example is:

```
mpiexec -f machinefile -n 1 ./master : -n 19 ./slave
```

to ensure that the process with rank 0 runs on your workstation.

The arguments between ':'s in this syntax are called "argument sets," since they apply to a set of processes. More argments are described in the *User's Guide*.

If you have completed all of the above steps, you have successfully installed MPICH2 and run an MPI example.

2.3 Compiler Optimization Levels

MPICH2 can be configured with two sets of compiler flags: `CFLAGS`, `CXXFLAGS`, `FFLAGS`, `FCFLAGS` (abbreviated as `xFLAGS`) and `MPICH2LIB_CFLAGS`, `MPICH2LIB_CXXFLAGS`, `MPICH2LIB_FFLAGS`, `MPICH2LIB_FCFLAGS` (abbreviated as `MPICH2LIB_xFLAGS`) for compilation; `LDFLAGS` and `MPICH2LIB_LDFLAGS` for linking. All these flags can be set as part of configure command or through environment variables. (`CPPFLAGS` stands for C preprocessor flags, which should NOT be set)

Both `xFLAGS` and `MPICH2LIB_xFLAGS` affect the compilation of the MPICH2 libraries. However, only `xFLAGS` is appended to MPI wrapper scripts, `mpicc` and friends.

MPICH2 libraries are built with default compiler optimization, -O2, which can be modified by –enable-fast configure option. For instance, –disable-fast disables the default optimization option. –enable-fast=O¡n¿ sets default compiler optimization as -O¡n¿. For more details of –enable-fast, see the output of "configure –help". Any other complicated optimization flags for MPICH2 libraries have to be set throught `MPICH2LIB_xFLAGS`. `CFLAGS` and friends are empty by default.

For example, to build a "production" MPICH2 environment with -O3 for all language bindings, one can simply do

```
./configure --enable-fast=O3
```

or

```
./configure --disable-fast MPICH2LIB_CFLAGS=-O3 \
```

```
MPICH2LIB_FFLAGS=-O3 \
MPICH2LIB_CXXFLAGS=-O3 \
MPICH2LIB_FCFLAGS=-O3
```

This will cause the MPICH2 libraries to be built with -O3, and -O3 will not be included in the mpicc and other MPI wrapper script.

There are certain compiler flags that should not be used with MPICH2's configure, e.g. gcc's -Werror which would confuse configure and cause certain configure tests to fail to detect the correct system features. To use -Werror in building MPICH2 libraries, you can pass the compiler flags during the make step through the Makefile variable, `MPICH2_MAKE_CFLAGS` as follows:

```
make VERBOSE=1 MPICH2_MAKE_CFLAGS="-Wall -Werror"
```

(assume CC is set to gcc). The content of `MPICH2_MAKE_CFLAGS` is appended to the `CFLAGS` in almost all Makefiles.

2.4 Common Non-Default Configuration Options

A list of `configure` options is found in Section 10. Here we comment on some of them.

2.4.1 The Most Important Configure Options

–**prefix** Set the installation directories for MPICH2.

–**enable-debuginfo** Provide access to the message queues for debuggers such as Totalview.

–**enable-g** Build MPICH2 with various debugging options. This is of interest primarily to MPICH2 developers. The options

```
--enable-g=dbg,mem,log
```

are recommended in that case.

–**enable-fast** Configure MPICH2 for fastest performance at the expense of error reporting and other program development aids. This is recommended only for getting the best performance out of proven production applications, and for benchmarking.

–**enable-shared** Build MPICH2 with shared libraries. MPICH2 will try to automatically detect the type of shared library support required. For cases where you want to manually control this, the following options can be used.

`--enable-sharedlibs=gcc` for standard gcc on Linux

`--enable-sharedlibs=osx-gcc` for Mac OS X or

`--enable-sharedlibs=solaris-cc` for cc on Solaris

–**with-pm** Select the process manager. The default is `hydra`; also useful are `gforker` and `remshell`. You can build with all three process managers by specifying

```
--with-pm=hydra:gforker:remshell
```

–**without-mpe** Configure MPICH2 without the MPE package of program development tools (including the Jumpshot performance viewer)

–**with-java** Set the location of Java installation. This option is necessary only if the default Java installation in your PATH does not contain a valid Java installation for Jumpshot, e.g.

```
--with-java=/opt/jdk1.6.0
```

2.4.2 Using the Absoft Fortran compilers with MPICH2

For best results, it is important to force the Absoft Fortran compilers to make all routine names monocase. In addition, if lower case is chosen (this will match common use by many programs), you must also tell the the Absoft compiles to append an underscore to global names in order to access routines such as `getarg` (`getarg` is not used by MPICH2 but is used in some of the tests and is often used in application programs). We recommend configuring MPICH2 with the following options

```
setenv F77 f77
setenv FFLAGS "-f -N15"
setenv FCFLAGS "-YALL_NAMES=LCS -YEXT_SFX=_"

./configure ....
```

2.5 Shared Libraries

Shared libraries are currently only supported for gcc (and gcc-like compilers) on Linux and Mac and for cc on Solaris. To have shared libraries created when MPICH2 is built, specify the following when MPICH2 is configured:

```
configure --enable-shared
```

For users who wish to manually control the linker parameters, this can be done using:

```
configure --enable-sharedlibs=gcc          (on Linux)
configure --enable-sharedlibs=osx-gcc      (on Mac OS X)
configure --enable-sharedlibs=solaris-cc   (on Solaris)
```

2.6 What to Tell the Users

Now that MPICH2 has been installed, the users have to be informed of how to use it. Part of this is covered in the *User's Guide*. Other things users need to know are covered here.

3 Migrating from MPICH1

MPICH2 is an all-new rewrite of MPICH1. Although the basic steps for installation have remained the same (`configure`, `make`, `make install`), a number of things have changed. In this section we attempt to point out what you may be used to in MPICH1 that are now different in MPICH2.

3.1 Configure Options

The arguments to `configure` are different in MPICH1 and MPICH2; the `Installer's Guide` discusses `configure`. In particular, the newer `configure` in MPICH2 does not support the `-cc=<compiler-name>` (or `-fc`, `-c++`, or `-f90`) options. Instead, many of the items that could be specified in the command line to configure in MPICH1 must now be set by defining an environment variable. E.g., while MPICH1 allowed

```
./configure -cc=pgcc
```

MPICH2 requires

```
setenv CC pgcc
```

(or `export CC=pgcc` for `ksh` or `CC=pgcc ; export CC` for strict `sh`) before `./configure`. Basically, every option to the MPICH-1 configure that does not start with `--enable` or `--with` is not available as a configure option in MPICH2. Instead, environment variables must be used. This is consistent (and required) for use of version 2 GNU `autoconf`.

3.2 Other Differences

Other differences between MPICH1 and MPICH2 include the handling of process managers and the choice of communication device.

For example, the new process managers have a new format and slightly different semantics for the `-machinefile` option. Assume that you type this data into a file named `machfile`:

```
bp400:2
bp401:2
bp402:2
bp403:2
```

If you then run a parallel job with this machinefile, you would expect ranks 0 and 1 to run on bp400 because it says to run 2 processes there before going on to bp401. Ranks 2 and 3 would run on bp401, and rank 4 on bp402, e.g.:

```
mpiexec -l -machinefile machfile -n 5 hostname
```

produces:

```
0: bp400
1: bp400
2: bp401
3: bp401
4: bp402
```

4 Choosing the Communication Device

MPICH2 is designed to be build with many different communication devices, allowing an implementation to be tuned for different communication fabrics. A simple communication device, known as "ch3" (for the third version of the "channel" interface) is provided with MPICH2 and is the default choice.

The ch3 device itself supports a variety of communication methods. These are specified by providing the name of the method after a colon in the `--with-device` configure option. For example, `--with-device=ch3:sock` selects the (older) socket-base communication method. The supported methods include:

ch3:nemesis This method is our new, high performance method. It has been made the default communication channel starting the 1.1 release of MPICH2. It uses shared-memory to send messages between processes on the same node and the network for processes between nodes. Currently sockets and Myrinet-MX are supported networks. It supports `MPI_THREAD_MULTIPLE` and other levels of thread safety.

ch3:sock This method uses sockets for all communications between processes. It supports `MPI_THREAD_MULTIPLE` and other levels of thread safety.

Most installations should use the default `ch3:nemesis` method for best performance. For platforms that are not supported by nemesis, the `ch3:sock` method is suggested.

5 Installing and Managing Process Managers

MPICH2 has been designed to work with multiple process managers; that is, although you can start MPICH2 jobs with `mpiexec`, there are different mechanisms by which your processes are started. An interface (called PMI) isolates the MPICH2 library code from the process manager. Currently three process managers are distributed with MPICH2

hydra This is the default process manager tha natively uses the existing daemons on the system such as ssh, slurm, pbs.

smpd This one can be used for both Linux and Windows. It is the only process manager that supports the Windows version of MPICH2.

gforker This is a simple process manager that creates all processes on a single machine. It is useful for both debugging and on shared memory multiprocessors.

5.1 hydra

`hydra` is the default process manager that launches processes using the native daemons present on the system such as ssh, slurm, pbs, etc. To configure with the `hydra` process manager, use

```
configure --with-pm=hydra ...
```

5.2 SMPD

5.2.1 Configuration

You may add the following configure options, `--with-pm=smpd`, to build and install the smpd process manager. The process manager, smpd, will be installed to the bin sub-directory of the installation directory of your choice specified by the `--prefix` option.

smpd process managers run on each node as stand-alone daemons and need to be running on all nodes that will participate in MPI jobs. smpd process managers are not connected to each other and rely on a known port to communicate with each other. Note: If you want multiple users to use the same nodes they must each configure their smpds to use a unique port per user.

smpd uses a configuration file to store settings. The default location is `~/.smpd`. This file must not be readable by anyone other than the owner and contains at least one required option - the access passphrase. This is stored in the configuration file as `phrase=<phrase>`. Access to running smpds is authenticated using this passphrase and it must not be your user password.

5.2.2 Usage and administration

Users will start the smpd daemons before launching mpi jobs. To get an smpd running on a node, execute

```
smpd -s
```

Executing this for the first time will prompt the user to create a `~/.smpd` configuration file and passphrase if one does not already exist.

Then users can use `mpiexec` to launch MPI jobs.

All options to `smpd`:

`smpd -s`
: Start the smpd service/daemon for the current user. You can add `-p <port>` to specify the port to listen on. All smpds must use the same port and if you don't use the default then you will have to add `-p <port>` to mpiexec or add the `port=<port>` to the `.smpd` configuration file.

`smpd -r`
: Start the smpd service/daemon in root/multi-user mode. This is not yet implemented.

`smpd -shutdown [host]`
: Shutdown the smpd on the local host or specified host. Warning: this will cause the smpd to exit and no mpiexec or smpd commands can be issued to the host until smpd is started again.

5.3 gforker

`gforker` is a simple process manager that runs all processes on a single node; it's version of `mpiexec` uses the system `fork` and `exec` calls to create the new processes. To configure with the `gforker` process manager, use

```
configure --with-pm=gforker ...
```

6 Testing

Once MPICH2 has been installed, you can test it by running some of the example programs in the `examples` directory. A more thorough test can be run with the command `make testing`. This will produce a summary on standard output, along with an XML version of the test results in `mpich2/test/mpi`. In addition, running `make testing` from the top-level (`mpich2`) directory will run tests of the commands, such as `mpicc` and `mpiexec`, that are included with MPICH2.

Other MPI test suites are available from `http://www.mcs.anl.gov/mpi/mpi-test/tsuite.html`. As part of the MPICH2 development, we run the MPICH1, MPICH2, C++, and Intel test suites every night and post the results on `http://www.mcs.anl.gov/mpi/mpich1/micronotes/mpich2-status/`. Other tests are run on an occasional basis.

7 Benchmarking

There are many benchmarking programs for MPI implementations. Three that we use are `mpptest` (`http://www.mcs.anl.gov/mpi/mpptest`), `netpipe` (`http://www.scl.ameslab.gov/netpipe`), and `SkaMPI` (`http://liinwww.ira.uka.de/~skampi`). Each of these has different strengths and weaknesses and reveals different properties of the MPI implementation.

In addition, the MPICH2 test suite contains a few programs to test for performance artifacts in the directory `test/mpi/perf`. An example of a performance artifact is markedly different performance for the same operation when performed in two different ways. For example, using an MPI datatype for a non-contiguous transfer should not be much slower than packing the data into a contiguous buffer, sending it as a contiguous buffer, and then unpacking it into the destination buffer. An example of this from the MPI-1 standard illustrates the use of MPI datatypes to transpose a matrix "on the fly," and one test in `test/mpi/perf` checks that the MPI implementation performs well in this case.

8 MPE

MPICH2 comes with the same MPE (Multi-Processing Environment) tools that are included with MPICH1. These include several trace libraries for recording the execution of MPI programs and the Jumpshot and SLOG tools for performance visualization. The MPE tools are built and installed by default and should be available without requiring any additional steps. The installation of MPE is documented in `mpich2/src/mpe2/INSTALL` and the usage of MPE is documented in `mpich2/src/mpe2/README` and MPICH2 user's guide.

9 Windows Version

9.1 Binary distribution

The Windows binary distribution uses the Microsoft Installer. Download and execute `mpich2-1.x.xxx.msi` to install the binary distribution. The default installation path is `C:\Program Files\MPICH2`. You must have administrator privileges to install `mpich2-1.x.xxx.msi`. The installer installs a Windows service to launch MPICH applications and only administrators may install services. This process manager is called smpd.exe. Access to the process manager is passphrase protected. The installer asks for this passphrase. Do not use your user password. The same passphrase must be installed on all nodes that will participate in a single MPI job.

Under the installation directory are three sub-directories: `include`, `bin`, and `lib`. The `include` and `lib` directories contain the header files and libraries necessary to compile MPI applications. The `bin` directory contains the process manager, `smpd.exe`, and the the MPI job launcher, `mpiexec.exe`. The dlls that implement MPICH2 are copied to the Windows system32 directory.

You can install MPICH in unattended mode by executing

```
msiexec /q /I mpich2-1.x.xxx.msi
```

The smpd process manager for Windows runs as a service that can launch jobs for multiple users. It does not need to be started like the unix version

does. The service is automatically started when it is installed and when the machine reboots. smpd for Windows has additional options:

`smpd -start`
: Start the Windows `smpd` service.

`smpd -stop`
: Stop the Windows `smpd` service.

`smpd -install`
: Install the `smpd` service.

`smpd -remove`
: Remove the `smpd` service.

`smpd -register_spn`
: Register the Service Principal Name with the domain controller. This command enables passwordless authentication using kerberos. It must be run on each node individualy by a domain administrator.

9.2 Source distribution

In order to build MPICH2 from the source distribution under Windows, you must have MS Developer Studio .NET 2003 or later, perl and optionally Intel Fortran 8 or later.

- Download `mpich2-1.x.y.tar.gz` and unzip it.
- Bring up a Visual Studio Command prompt with the compiler environment variables set.
- Run `winconfigure.wsf`. If you don't have a Fortran compiler add the `--remove-fortran` option to `winconfigure` to remove all the Fortran projects and dependencies. Execute `winconfigure.wsf /?` to see all available options.
- open `mpich2\mpich2.sln`
- build the ch3sockRelease mpich2 solution
- build the ch3sockRelease mpich2s project

- build the Release mpich2 solution
- build the fortRelease mpich2 solution
- build the gfortRelease mpich2 solution
- build the sfortRelease mpich2 solution
- build the channel of your choice. The options are `sock` and `nemesis`. If you plan on launching more processes than you have processors you should use the default `sock` channel. The `nemesis` channel uses a polling progress engine that can perform poorly when multiple processes compete for individual processors.

9.3 `cygwin`

MPICH2 can also be built under `cygwin` using the source distribution and the Unix commands described in previous sections. This will not build the same libraries as described in this section. It will build a "Unix" distribution that runs under `cygwin`.

9.4 `Vista/Windows 7/Windows Server 2008 Users`

To install MPICH version `1.3` or later please follow the instructions provided in 9.1. Please follow the steps below to install older versions of MPICH on Windows flavors with User Account Control.

1. Right-click on the command prompt icon and choose "Run as administrator"
2. You can install MPICH from the administrator command prompt by executing

```
msiexec /I mpich2-1.x.xxx.msi
```

10 All Configure Options

To get the latest list of all the configure options recognized by the top-level configure, use:

```
configure --help
```

Not all of these options may be fully supported yet.

Notes on the configure options. The `--with-htmldir` and `--with-docdir` options specify the directories into which the documentation will be installed by `make install`.