MPICH Installer's Guide*

Version 3.2.1

Mathematics and Computer Science Division
Argonne National Laboratory

Abdelhalim Amer Pavan Balaji Wesley Bland
William Gropp Yanfei Guo Rob Latham Huiwei Lu
Lena Oden Antonio J. Peña Ken Raffenetti
Sangmin Seo Min Si Rajeev Thakur Junchao Zhang
Xin Zhao

May 25, 2022

*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 **1**

2 Quick Start **1**

2.1 Prerequisites 1

2.2 From A Standing Start to Running an MPI Program 2

2.3 Selecting the Compilers 6

2.4 Compiler Optimization Levels 7

2.5 Common Non-Default Configuration Options 8

2.5.1 The Most Important Configure Options 8

2.5.2 Using the Absoft Fortran compilers with MPICH . . . 9

2.6 Shared Libraries 9

2.7 What to Tell the Users 9

3 Migrating from MPICH1 **10**

3.1 Configure Options 10

3.2 Other Differences 10

4 Choosing the Communication Device **11**

5 Installing and Managing Process Managers **12**

5.1 hydra 12

5.2 gforker 13

6 Testing **13**

7 Benchmarking **13**

8 All Configure Options **14**

1 Introduction

This manual describes how to obtain and install MPICH, the MPI 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 MPICH. In the event of problems, send mail to `discuss@mpich.org`. Once MPICH is installed, details on how to run MPI jobs are covered in the *MPICH User's Guide*, found in this same `doc` subdirectory.

MPICH 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, `mpich.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. MPICH 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 MPICH 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 MPICH, just not with Fortran applications.)

2.2 From A Standing Start to Running an MPI Program

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

1. Unpack the tar file.

   ```
   tar xfz mpich.tar.gz
   ```

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

   ```
   gunzip -c mpich.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 `mpich-3.2.1`.

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

   ```
   mkdir /home/you/mpich-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/mpich-3.2.1
```

4. Choose any configure options. See Section 2.5.1 for a description of the most important options to consider.

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

```
cd  /tmp/you/mpich-3.2.1
/home/you/libraries/mpich-3.2.1/configure \
        -prefix=/home/you/mpich-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 `discuss@mpich.org`. 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 MPICH:

```
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 `discuss@mpich.org`.

7. Install the MPICH 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=mpich-<versrion>
```

and corresponding installcheck step should be

```
make installcheck PACKAGE=mpich-<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/mpich-install/bin:$PATH
   ```

   for `csh` and `tcsh`, or

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

   for `bash`, and

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

   for `sh` (the `bash` syntax may work for more recent implementations of `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.

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 standard.

    As part of the build process for MPICH, a simple program to compute the value of π by numerical integration is created in the `mpich-3.2.1/examples` directory. If the current directory is the top level MPICH 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 MPICH and run an MPI example.

2.3 Selecting the Compilers

The MPICH configure step will attempt to find the C, C++, and Fortran compilers for you, but if you either want to override the default or need to specify a compiler that configure doesn't recognize, you can specify them on the command line using these variables

CC The C compiler.

CXX The C++ compiler. Use `--disable-cxx` if you do not want to build the MPI C++ interface

F77 The Fortran 77 compiler (for the original MPI Fortran bindings). Use `--disable-f77` if you do not want to build either the Fortran 77 or Fortran 90 MPI interfaces

FC The Fortran 90 (or later) compiler. Use `--disable-fc` if you do not want to build the Fortran 90 MPI interfaces. Note that in previous versions of MPICH, the variable name was `F90`. As Fortran has had 3 major releases since Fortran 90 (95, 2003, and 2008), most tools, including those built with GNU autotools, have or are changing to use `FC` instead of `F90`.

For example, to select the Intel compilers instead of the GNU compilers on a system with both, use

```
./configure CC=icc CXX=icpc F77=ifort FC=ifort ...
```

Note the use of the same Fortran compiler, `ifort`, for both Fortran 77 and Fortran 90; this is an increasingly common choice.

2.4 Compiler Optimization Levels

MPICH can be configured with two sets of compiler flags: `CFLAGS`, `CXXFLAGS`, `FFLAGS`, `FCFLAGS` (abbreviated as `xFLAGS`) and `MPICHLIB_CFLAGS`, `MPICHLIB_CXXFLAGS`, `MPICHLIB_FFLAGS`, `MPICHLIB_FCFLAGS` (abbreviated as `MPICHLIB_xFLAGS`) for compilation; `LDFLAGS` and `MPICHLIB_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 `MPICHLIB_xFLAGS` affect the compilation of the MPICH libraries. However, only `xFLAGS` is appended to MPI wrapper scripts, `mpicc` and friends.

MPICH 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>` (note that this assumes that the compiler accepts this format). For more details of `--enable-fast`, see the output of `configure --help`. Any other complicated optimization flags for MPICH libraries have to be set throught `MPICHLIB_xFLAGS`. `CFLAGS` and friends are empty by default.

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

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

or

```
./configure --enable-fast=all MPICHLIB_CFLAGS=-O3 \
                              MPICHLIB_FFLAGS=-O3 \
                              MPICHLIB_CXXFLAGS=-O3 \
                              MPICHLIB_FCFLAGS=-O3
```

This will cause the MPICH 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 MPICH'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 MPICH libraries, you can pass the compiler flags during the make step through the Makefile variable, `MPICH_MAKE_CFLAGS` as follows:

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

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

2.5 Common Non-Default Configuration Options

A brief discussion of some of the `configure` options is found in Section 8. Here we comment on some of the most commonly used options.

2.5.1 The Most Important Configure Options

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

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

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

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

are recommended in that case.

–**enable-fast** Configure MPICH 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 MPICH with shared libraries. MPICH will try to automatically detect the type of shared library support required. See Section 2.6 for more details.

–**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
```

–**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
```

2.5.2 Using the Absoft Fortran compilers with MPICH

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 MPICH but is used in some of the tests and is often used in application programs). We recommend configuring MPICH with the following options

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

./configure ....
```

2.6 Shared Libraries

To have shared libraries created when MPICH is built, specify the following when MPICH is configured:

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

2.7 What to Tell the Users

Now that MPICH 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

MPICH 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 MPICH.

3.1 Configure Options

The arguments to `configure` are different in MPICH1 and MPICH; the `Installer's Guide` discusses `configure`. In particular, the newer `configure` in MPICH 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
```

MPICH requires

```
./configure CC=pgcc
```

Basically, every option to the MPICH-1 configure that does not start with `--enable` or `--with` is not available as a configure option in MPICH. 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 MPICH 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

MPICH 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 MPICH 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. Methods supported by the MPICH group 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 MPICH. 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.

MPICH is designed to efficiently support all types of systems. The `ch3:nemesis` device is the primary focus of the MPICH group, but other research groups and computer vendors can and have developed both their own ch3 "channels" as well as complete communication "devices" in place of ch3.

5 Installing and Managing Process Managers

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

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

gforker This is a simple process manager that creates all processes on a single machine. It is useful both for debugging and for running 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 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 MPICH 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 `mpich/test/mpi`. In addition, running `make testing` from the top-level (`mpich`) directory will run tests of the commands, such as `mpicc` and `mpiexec`, that are included with MPICH.

Other MPI test suites are available from `http://www.mcs.anl.gov/mpi/mpi-test/tsuite.html`. As part of the MPICH development, we run the MPICH1, MPICH, C++, and Intel test suites every night and post the results on `http://www.mpich.org/static/cron/tests/`. 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 MPICH 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 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`.