MPICH User's Guide*

Version 3.1rc1

Mathematics and Computer Science Division

Argonne National Laboratory

Pavan Balaji
Wesley Bland
William Gropp
Rob Latham
Antonio Peña
Rajeev Thakur

Past Contributors:
David Ashton
Darius Buntinas
Ralph Butler
Anthony Chan
James Dinan
David Goodell
Jayesh Krishna
Ewing Lusk
Guillaume Mercier
Rob Ross
Brian Toonen

November 6, 2013

*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 assumes that MPICH has already been installed. For instructions on how to install MPICH, see the *MPICH Installer's Guide*, or the `README` in the top-level MPICH directory. This manual explains how to compile, link, and run MPI applications, and use certain tools that come with MPICH. This is a preliminary version and some sections are not complete yet. However, there should be enough here to get you started with MPICH.

2 Getting Started with MPICH

MPICH is a high-performance and widely portable implementation of the MPI Standard, designed to implement all of MPI-1, MPI-2, and MPI-3 (including dynamic process management, one-sided operations, parallel I/O, and other extensions). The *MPICH Installer's Guide* provides some information on MPICH with respect to configuring and installing it. Details on compiling, linking, and running MPI programs are described below.

2.1 Default Runtime Environment

MPICH provides a separation of process management and communication. The default runtime environment in MPICH is called Hydra. Other process managers are also available.

2.2 Starting Parallel Jobs

MPICH implements `mpiexec` and all of its standard arguments, together with some extensions. See Section 5.1 for standard arguments to `mpiexec` and various subsections of Section 5 for extensions particular to various process management systems.

2.3 Command-Line Arguments in Fortran

MPICH1 (more precisely MPICH1's `mpirun`) required access to command line arguments in all application programs, including Fortran ones, and

MPICH1's `configure` devoted some effort to finding the libraries that contained the right versions of `iargc` and `getarg` and including those libraries with which the `mpif77` script linked MPI programs. Since MPICH does not require access to command line arguments to applications, these functions are optional, and `configure` does nothing special with them. If you need them in your applications, you will have to ensure that they are available in the Fortran environment you are using.

3 Quick Start

To use MPICH, you will have to know the directory where MPICH has been installed. (Either you installed it there yourself, or your systems administrator has installed it. One place to look in this case might be `/usr/local`. If MPICH has not yet been installed, see the *MPICH Installer's Guide*.) We suggest that you put the `bin` subdirectory of that directory into your path. This will give you access to assorted MPICH commands to compile, link, and run your programs conveniently. Other commands in this directory manage parts of the run-time environment and execute tools.

One of the first commands you might run is `mpichversion` to find out the exact version and configuration of MPICH you are working with. Some of the material in this manual depends on just what version of MPICH you are using and how it was configured at installation time.

You should now be able to run an MPI program. Let us assume that the directory where MPICH has been installed is `/home/you/mpich-installed`, and that you have added that directory to your path, using

```
setenv PATH /home/you/mpich-installed/bin:$PATH
```

for `tcsh` and `csh`, or

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

for `bash` or `sh`. Then to run an MPI program, albeit only on one machine, you can do:

```
cd  /home/you/mpich-installed/examples
mpiexec -n 3 ./cpi
```

Details for these commands are provided below, but if you can successfully execute them here, then you have a correctly installed MPICH and have run an MPI program.

4 Compiling and Linking

A convenient way to compile and link your program is by using scripts that use the same compiler that MPICH was built with. These are `mpicc`, `mpicxx`, `mpif77`, and `mpif90`, for C, C++, Fortran 77, and Fortran 90 programs, respectively. If any of these commands are missing, it means that MPICH was configured without support for that particular language.

4.1 Special Issues for C++

Some users may get error messages such as

```
SEEK_SET is #defined but must not be for the C++ binding of MPI
```

The problem is that both `stdio.h` and the MPI C++ interface use `SEEK_SET`, `SEEK_CUR`, and `SEEK_END`. This is really a bug in the MPI standard. You can try adding

```
#undef SEEK_SET
#undef SEEK_END
#undef SEEK_CUR
```

before `mpi.h` is included, or add the definition

```
-DMPICH_IGNORE_CXX_SEEK
```

to the command line (this will cause the MPI versions of `SEEK_SET` etc. to be skipped).

4.2 Special Issues for Fortran

MPICH provides two kinds of support for Fortran programs. For Fortran 77 programmers, the file `mpif.h` provides the definitions of the MPI constants

such as MPI_COMM_WORLD. Fortran 90 programmers should use the MPI module instead; this provides all of the definitions as well as interface definitions for many of the MPI functions. However, this MPI module does not provide full Fortran 90 support; in particular, interfaces for the routines, such as MPI_Send, that take "choice" arguments are not provided.

5 Running Programs with mpiexec

The MPI Standard describes mpiexec as a suggested way to run MPI programs. MPICH implements the mpiexec standard, and also provides some extensions.

5.1 Standard mpiexec

Here we describe the standard mpiexec arguments from the MPI Standard [1]. The simplest form of a command to start an MPI job is

```
mpiexec -f machinefile -n 32 a.out
```

to start the executable a.out with 32 processes (providing an MPI_COMM_WORLD of size 32 inside the MPI application). Other options are supported, for search paths for executables, working directories, and even a more general way of specifying a number of processes. Multiple sets of processes can be run with different executables and different values for their arguments, with ":" separating the sets of processes, as in:

```
mpiexec -f machinefile -n 1 ./master : -n 32 ./slave
```

It is also possible to start a one process MPI job (with a MPI_COMM_WORLD whose size is equal to 1), without using mpiexec. This process will become an MPI process when it calls MPI_Init, and it may then call other MPI functions. Currently, MPICH does not fully support calling the dynamic process routines from the MPI standard (e.g., MPI_Comm_spawn or MPI_Comm_accept) from processes that are not started with mpiexec.

5.2 Extensions for All Process Management Environments

Some `mpiexec` arguments are specific to particular communication subsystems ("devices") or process management environments ("process managers"). Our intention is to make all arguments as uniform as possible across devices and process managers. For the time being we will document these separately.

5.3 `mpiexec` Extensions for the Hydra Process Manager

MPICH provides a number of process management systems. Hydra is the default process manager in MPICH. More details on Hydra and its extensions to mpiexec can be found at `http://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager`

5.4 Extensions for SMPD Process Management Environment

SMPD is an alternate process manager that runs on both Unix and Windows. It can launch jobs across both platforms if the binary formats match (big/little endianness and size of C types– `int`, `long`, `void*`, etc).

5.4.1 `mpiexec` arguments for SMPD

`mpiexec` for smpd accepts the standard MPI `mpiexec` options. Execute

```
mpiexec
```

or

```
mpiexec -help2
```

to print the usage options. Typical usage:

```
mpiexec -n 10 myapp.exe
```

All options to `mpiexec`:

`-n x`

`-np x`
launch `x` processes

`-localonly x`

`-np x -localonly`
launch `x` processes on the local machine

`-machinefile filename`
use a file to list the names of machines to launch on

`-host hostname`
launch on the specified host.

`-hosts n host1 host2 ... hostn`

`-hosts n host1 m1 host2 m2 ... hostn mn`
launch on the specified hosts. In the second version the number of processes = m1 + m2 + ... + mn

`-dir drive:\my\working\directory`

`-wdir /my/working/directory`
launch processes with the specified working directory. (`-dir` and `-wdir` are equivalent)

`-env var val`
set environment variable before launching the processes

`-exitcodes`
print the process exit codes when each process exits.

`-noprompt`
prevent `mpiexec` from prompting for user credentials. Instead errors will be printed and `mpiexec` will exit.

`-localroot`
launch the root process directly from mpiexec if the host is local. (This allows the root process to create windows and be debugged.)

`-port port`

`-p port`
specify the port that `smpd` is listening on.

`-phrase passphrase`
: specify the passphrase to authenticate connections to `smpd` with.

`-smpdfile filename`
: specify the file where the `smpd` options are stored including the passphrase. (unix only option)

`-path search_path`
: search path for executable, ; separated

`-timeout seconds`
: timeout for the job.

Windows specific options:

`-map drive:\\host\share`
: map a drive on all the nodes this mapping will be removed when the processes exit

`-logon`
: prompt for user account and password

`-pwdfile filename`
: read the account and password from the file specified.

  put the account on the first line and the password on the second

`-nopopup_debug`
: disable the system popup dialog if the process crashes

`-priority class[:level]`
: set the process startup priority class and optionally level.
  class = 0,1,2,3,4 = idle, below, normal, above, high
  level = 0,1,2,3,4,5 = idle, lowest, below, normal, above, highest
  the default is -priority 2:3

`-register`
: encrypt a user name and password to the Windows registry.

`-remove`
: delete the encrypted credentials from the Windows registry.

`-validate [-host hostname]`
: validate the encrypted credentials for the current or specified host.

`-delegate`
use passwordless delegation to launch processes.

`-impersonate`
use passwordless authentication to launch processes.

`-plaintext`
don't encrypt the data on the wire.

5.5 Extensions for the gforker Process Management Environment

`gforker` is a process management system for starting processes on a single machine, so called because the MPI processes are simply `fork`ed from the `mpiexec` process. This process manager supports programs that use `MPI_Comm_spawn` and the other dynamic process routines, but does not support the use of the dynamic process routines from programs that are not started with `mpiexec`. The `gforker` process manager is primarily intended as a debugging aid as it simplifies development and testing of MPI programs on a single node or processor.

5.5.1 `mpiexec` arguments for gforker

In addition to the standard `mpiexec` command-line arguments, the `gforker` `mpiexec` supports the following options:

`-np <num>` A synonym for the standard `-n` argument

`-env <name> <value>` Set the environment variable `<name>` to `<value>` for the processes being run by `mpiexec`.

`-envnone` Pass no environment variables (other than ones specified with other `-env` or `-genv` arguments) to the processes being run by `mpiexec`. By default, all environment variables are provided to each MPI process (rationale: principle of least surprise for the user)

`-envlist <list>` Pass the listed environment variables (names separated by commas), with their current values, to the processes being run by `mpiexec`.

`-genv <name> <value>` The

-genv options have the same meaning as their corresponding `-env` version, except they apply to all executables, not just the current executable (in the case that the colon syntax is used to specify multiple execuables).

`-genvnone` Like `-envnone`, but for all executables

`-genvlist <list>` Like `-envlist`, but for all executables

`-usize <n>` Specify the value returned for the value of the attribute `MPI_UNIVERSE_SIZE`.

`-l` Label standard out and standard error (`stdout` and `stderr`) with the rank of the process

`-maxtime <n>` Set a timelimit of `<n>` seconds.

`-exitinfo` Provide more information on the reason each process exited if there is an abnormal exit

In addition to the commandline argments, the `gforker` `mpiexec` provides a number of environment variables that can be used to control the behavior of `mpiexec`:

`MPIEXEC_TIMEOUT` Maximum running time in seconds. `mpiexec` will terminate MPI programs that take longer than the value specified by `MPIEXEC_TIMEOUT`.

`MPIEXEC_UNIVERSE_SIZE` Set the universe size

`MPIEXEC_PORT_RANGE` Set the range of ports that `mpiexec` will use in communicating with the processes that it starts. The format of this is `<low>:<high>`. For example, to specify any port between 10000 and 10100, use `10000:10100`.

`MPICH_PORT_RANGE` Has the same meaning as `MPIEXEC_PORT_RANGE` and is used if `MPIEXEC_PORT_RANGE` is not set.

`MPIEXEC_PREFIX_DEFAULT` If this environment variable is set, output to standard output is prefixed by the rank in `MPI_COMM_WORLD` of the process and output to standard error is prefixed by the rank and the text `(err)`; both are followed by an angle bracket (`>`). If this variable is not set, there is no prefix.

`MPIEXEC_PREFIX_STDOUT` Set the prefix used for lines sent to standard output. A `%d` is replaced with the rank in `MPI_COMM_WORLD`; a `%w` is replaced with an indication of which `MPI_COMM_WORLD` in MPI jobs that involve multiple `MPI_COMM_WORLD`s (e.g., ones that use `MPI_Comm_spawn` or `MPI_Comm_connect`).

`MPIEXEC_PREFIX_STDERR` Like `MPIEXEC_PREFIX_STDOUT`, but for standard error.

`MPIEXEC_STDOUTBUF` Sets the buffering mode for standard output. Valid values are `NONE` (no buffering), `LINE` (buffering by lines), and `BLOCK` (buffering by blocks of characters; the size of the block is implementation defined). The default is `NONE`.

`MPIEXEC_STDERRBUF` Like `MPIEXEC_STDOUTBUF`, but for standard error.

5.6 Restrictions of the remshell Process Management Environment

The `remshell` "process manager" provides a very simple version of `mpiexec` that makes use of the secure shell command (`ssh`) to start processes on a collection of machines. As this is intended primarily as an illustration of how to build a version of `mpiexec` that works with other process managers, it does not implement all of the features of the other `mpiexec` programs described in this document. In particular, it ignores the command line options that control the environment variables given to the MPI programs. It does support the same output labeling features provided by the `gforker` version of `mpiexec`. However, this version of `mpiexec` can be used much like the `mpirun` for the `ch_p4` device in MPICH-1 to run programs on a collection of machines that allow remote shells. A file by the name of `machines` should contain the names of machines on which processes can be run, one machine name per line. There must be enough machines listed to satisfy the requested number of processes; you can list the same machine name multiple times if necessary.

5.7 Using MPICH with SLURM and PBS

There are multiple ways of using MPICH with SLURM or PBS. Hydra provides native support for both SLURM and PBS, and is likely the easiest

way to use MPICH on these systems (see the Hydra documentation above for more details).

Alternatively, SLURM also provides compatibility with MPICH's internal process management interface. To use this, you need to configure MPICH with SLURM support, and then use the `srun` job launching utility provided by SLURM.

For PBS, MPICH jobs can be launched in two ways: (i) use Hydra's mpiexec with the appropriate options corresponding to PBS, or (ii) using the OSC mpiexec.

5.7.1 OSC mpiexec

Pete Wyckoff from the Ohio Supercomputer Center provides a alternate utility called OSC mpiexec to launch MPICH jobs on PBS systems. More information about this can be found here: `http://www.osc.edu/~pw/mpiexec`

6 Specification of Implementation Details

The MPI Standard defines a number of areas where a library is free to define its own specific behavior as long as such behavior is documented appropriately. This section provides that documentation for MPICH where necessary.

6.1 MPI Error Handlers for Communicators

In Section 8.3.1 (Error Handlers for Communicators) of the MPI-3.0 Standard [2], MPI defines an error handler callback function as

```
typedef void MPI_Comm_errhandler_function(MPI_Comm *, int *, ...);
```

Where the first argument is the communicator in use, the second argument is the error code to be returned by the MPI routine that raised the error, and the remaining arguments to be implementation specific "varargs". MPICH does not provide any arguments as part of this list. So a callback function being provided to MPICH is sufficient if the header is

```
typedef void MPI_Comm_errhandler_function(MPI_Comm *, int *);
```

7 Debugging

Debugging parallel programs is notoriously difficult. Here we describe a number of approaches, some of which depend on the exact version of MPICH you are using.

7.1 TotalView

MPICH supports use of the TotalView debugger from Etnus. If MPICH has been configured to enable debugging with TotalView then one can debug an MPI program using

```
totalview -a mpiexec -a -n 3 cpi
```

You will get a popup window from TotalView asking whether you want to start the job in a stopped state. If so, when the TotalView window appears, you may see assembly code in the source window. Click on `main` in the stack window (upper left) to see the source of the `main` function. TotalView will show that the program (all processes) are stopped in the call to `MPI_Init`.

If you have TotalView 8.1.0 or later, you can use a TotalView feature called indirect launch with MPICH. Invoke TotalView as:

```
totalview <program> -a <program args>
```

Then select the Process/Startup Parameters command. Choose the Parallel tab in the resulting dialog box and choose MPICH as the parallel system. Then set the number of tasks using the Tasks field and enter other needed mpiexec arguments into the Additional Starter Arguments field.

8 Checkpointing

MPICH supports checkpoint/rollback fault tolerance when used with the Hydra process manager. Currently only the BLCR checkpointing library

is supported. BLCR needs to be installed separately. Below we describe how to enable the feature in MPICH and how to use it. This information can also be found on the MPICH Wiki: `http://wiki.mpich.org/mpich/index.php/Checkpointing`

8.1 Configuring for checkpointing

First, you need to have BLCR version 0.8.2 installed on your machine. If it's installed in the default system location, add the following two options to your configure command:

```
--enable-checkpointing --with-hydra-ckpointlib=blcr
```

If BLCR is not installed in the default system location, you'll need to tell MPICH's configure where to find it. You might also need to set the `LD_LIBRARY_PATH` environment variable so that BLCR's shared libraries can be found. In this case add the following options to your configure command:

```
--enable-checkpointing --with-hydra-ckpointlib=blcr
--with-blcr=BLCR_INSTALL_DIR LD_LIBRARY_PATH=BLCR_INSTALL_DIR/lib
```

where `BLCR_INSTALL_DIR` is the directory where BLCR has been installed (whatever was specified in `--prefix` when BLCR was configured). Note, checkpointing is only supported with the Hydra process manager. Hyrda will used by default, unless you choose something else with the `--with-pm=` configure option.

After it's configured, compile as usual (e.g., `make; make install`).

8.2 Taking checkpoints

To use checkpointing, include the `-ckpointlib` option for `mpiexec` to specify the checkpointing library to use and `-ckpoint-prefix` to specify the directory where the checkpoint images should be written:

```
shell$ mpiexec -ckpointlib blcr \
       -ckpoint-prefix /home/buntinas/ckpts/app.ckpoint \
       -f hosts -n 4 ./app
```

While the application is running, the user can request for a checkpoint at any time by sending a `SIGUSR1` signal to `mpiexec`. You can also automatically checkpoint the application at regular intervals using the mpiexec option `-ckpoint-interval` to specify the number of seconds between checkpoints:

```
shell$ mpiexec -ckpointlib blcr \
       -ckpoint-prefix /home/buntinas/ckpts/app.ckpoint \
       -ckpoint-interval 3600 -f hosts -n 4 ./app
```

The checkpoint/restart parameters can also be controlled with the environment variables `HYDRA_CKPOINTLIB`, `HYDRA_CKPOINT_PREFIX` and `HYDRA_CKPOINT_INTERVAL`.

Each checkpoint generates one file per node. Note that checkpoints for all processes on a node will be stored in the same file. Each time a new checkpoint is taken an additional set of files are created. The files are numbered by the checkpoint number. This allows the application to be restarted from checkpoints other than the most recent. The checkpoint number can be specified with the `-ckpoint-num` parameter. To restart a process:

```
shell$ mpiexec -ckpointlib blcr \
       -ckpoint-prefix /home/buntinas/ckpts/app.ckpoint \
       -ckpoint-num 5 -f hosts -n 4
```

Note that by default, the process will be restarted from the first checkpoint, so in most cases, the checkpoint number should be specified.

9 Other Tools Provided with MPICH

MPICH also includes a test suite for MPI functionality; this suite may be found in the `mpich/test/mpi` source directory and can be run with the command `make testing`. This test suite should work with any MPI implementation, not just MPICH.

A Frequently Asked Questions

The frequently asked questions are maintained online here:`http://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions`

References

[1] Message Passing Interface Forum. MPI2: A Message Passing Interface standard. *International Journal of High Performance Computing Applications*, 12(1–2):1–299, 1998.

[2] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, version 3.0. Technical report, 2012.