Getting Started MPC
Multi-Processor Communications Library
Version 2.1

Patrick Carribault Marc Pérache Sylvain Didelot

April 12, 2011

Contents

1 Introduction

This *Getting Started* guide details the various ways to install and configure the MPC library (Multi-Processor Communications) version 2.1.0 including MPI 1.3, OpenMP 2.5 (with patched GCC) and PThread support. Section 2 describes the steps to install and setup the library. Section 3 enumerates the parallel-programming models supported in MPC. A Frequently Asked Questions (FAQ) section is also provided at this end of this guide (see section 5).

For further information on MPC internals, the reader may refer to the articles [3, 2, 1] containing more details about the MPC framework and its execution model.

2 Installation

This section takes you through a sequence of steps to get MPC up and running.

2.1 Prerequisites

The following prerequisites are required to compile MPC:

- The main archive file MPC_2.1.0.tar.gz
- A C compiler (e.g., gcc)
- A GNU C compiler
- An optional Fortran compiler if Fortran applications are to be used (e.g., g77 or gfortran)

Some additionnal extra libraries are required to compile the patched GCC and GDB. See their corresponding website and installation guide to get a list of prerequisites.

2.2 Standard Installation

These steps describe the default installation of MPC with its additionnal components (GCC and GDB).

1. Unpack the main archive (`tar` format) and go to the top-level directory:

   ```
   tar xfz MPC_2.1.0.tar.gz
   cd MPC_2.1.0
   ```

   If your tar program does not accept the 'z' option, use the following commands

   ```
   gunzip MPC_2.1.0.tar.gz
   tar xf MPC_2.1.0.tar
   cd MPC_2.1.0
   ```

2. Choose an installation directory. Using the default `/usr/local/` will overwrite legacy headers: *choose a custom install path*.

   ```
   mkdir /home/you/mpc-install
   ```

The most convenient choice is a directory shared among all the machines you want to use the library on. If such a directory is not accessible, you will have to deploy MPC on every machine after the installation.

3. Configure MPC, specifying the installation directory:

```
./configure --prefix=/home/you/mpc-install
```

4. Build MPC:

```
make
```

5. Install the MPC commands and library:

```
make install
```

6. Source the `mpcvars` script (located inside the `bin/` directory of the MPC installation) to update environment variables (e.g., `PATH` and `LD_LIBRARY_PATH`).

   For csh and tcsh:

```
source /home/you/mpc-install/bin/mpcvars.csh
```

   For bash and sh:

```
. /home/you/mpc-install/bin/mpcvars.sh
```

   Check that everything went well at this point by running

```
which mpcrun
which mpc_cc
```

7. To compile your first MPC program, you may execute the `mpc_cc` compiler:

```
mpc_cc MPC_Tests/parallel/MPC_Message_Passing/hello_world.c \
       -o hello_world
```

   This command uses the main default patched GCC to compile the code. If you want to use your favorite compiler instead (loosing some features like OpenMP support and global-variable removal), you may use the `mpc_cflags` and `mpc_ldflags` commands:

```
$CC MPC_Tests/parallel/MPC_Message_Passing/hello_world.c \
     -o hello_world ‘mpc_cflags‘ ‘mpc_ldflags‘
```

8. To execute your MPC program, use the mpcrun command:

```
mpcrun -m=ethread      -n=4 hello_world
mpcrun -m=ethread_mxn  -n=4 hello_world
mpcrun -m=pthread      -n=4 hello_world
```

   See the section *Thread types* for details on the '-m' option.

2.3 Custom Installation

The previous section described the default installation and configuration of MPC. But other alternatives are available. You can find out more details on the configuration by running:

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

The following options are related to additional libraries required to compile the MPC distribution:

- `--with-cpath=DIR1:DIR2:...`
  Add directories to the CPATH environment variable for the whole MPC distribution.
- `--with-library-path=DIR1:DIR2:...`
  Add directories to the LD_LIBRARY_PATH environment variable for the whole MPC distribution

For more information about options to MPC, run the `configure` script located in the `mpc` directory.

2.4 Known issues

2.4.1 Error related to mpfr/gmp

When building `mpc-gcc`, you may get errors related to `libmpfr` or `libgmp`. If those libraries are installed into a non standard prefix, it may be required to use the following arguments while running the `configure` script :

```
./configure --gcc-with-mpfr=YOUR_PREFIX --gcc-with-gmp=YOUR_PREFIX\
            --with-library-path=YOUR_PREFIX
```

In case of conflitcts with `libgmp`, it may be preferable to avoid the version 4.3.2 which broke its ABI due to a mistake in documentation and move to 5.0.1 or higher.

3 MPC Supported APIs

3.1 Message Passing

3.1.1 API

MPC is fully MPI-1.3 compliant and supports the `MPI_THREAD_MULTIPLE` level from standard 2. See the document *MPI: A Message-Passing Interface Standard* version 1.3 (May 2008) for more details.

3.1.2 Warning to Users

Remove global variables! In MPC, every MPI task is a thread and thus all tasks share global variables with each other.

3.1.3 Removing Global Variables

We propose several solutions to ease the removal of global variables:

1. Use the option `-Wmpc` with the patched GCC compiler to generate warnings. In this mode, the compiler will warn you about every global variable declared in the program.
2. Use the option `-fmpc-privatize` to automatically privatize the global variables. In this mode, every global variable is duplicated for every MPI task such as the code can run correctly with MPC. Note the application has to be compiled as a dynamic library for this solution to work.

3.2 OpenMP

3.2.1 API

MPC is fully OpenMP-2.5 compliant. See the document *OpenMP Application Program Interface* version 2.5 (May 2005) for more details.

3.2.2 Compiling and Running OpenMP Programs

To compile applications with OpenMP directives (C, C++ or Fortran), you have to use the default compiler coming with the MPC Distribution (see Section 2 for explanation to install MPC). This compiler is a patched version of GCC generating code for the MPC library when transforming OpenMP directives. Thus, to activate the OpenMP transformation, use the `-fopenmp` option with the compiler drivers `mpc_cc` (C), `mpc_cxx` (C++) or `mpc_f77` (Fortran).

3.2.3 Threadprivate Variables

The OpenMP standard proposes a directive to create *thread-private* variables. The MPC implementation follows the standard and supports this feature. Nevertheless, for this feature to work, the application has to be compiled as a dynamic library.

Here are a few steps to compile your program as a dynamic library.

1. change the name of the `main` of your program to `realmain`
2. compile your program to `.o` with the `-fPIC` option
   ```
   mpc_cc -fPIC -o prog.o -c prog.c
   ```
3. transform your `.o` into a dynamic library
   ```
   mpc_cc -shared -Wl,-soname,libprog.so -o libprog.so prog.o
   ```
4. compile the file `main.c` which should call the function `realmain` located in the dynamic library
   ```
   mpc_cc -o prog_exec main.c -L. -lprog
   ```
5. run your program specifying where is your dynamic library. If your program executable and the dynamic library are in the current directory, you can use the command
   ```
   LD_LIBRARY_PATH=. mpc_run ./prog_exec
   ```

A typical `main.c` file should look like this.

```
#include "mpc.h"

int realmain (int argc, char **argv) ;

int main (int argc, char **argv)
{
  int return_value = 0 ;
  return_value = realmain(argc,argv) ;
  return return_value ;
}
```

3.3 Threads

3.3.1 API

MPC provides a POSIX Thread 2003 compatible API.

3.3.2 Thread types

The main command `mpcrun` accepts the '-m' option to choose between several kind of threads. Here is a list of the current available thread types:

1. Ethread: Mx1 user level thread model.
2. Ethread_mxn: MxN user level thread model.
3. Pthread: underlying POSIX Thread library.

The article [3] contains more details on the multiple thread types and their characteristics.

3.3.3 Warning to Users

It is dangerous to mix MPC POSIX Threads and system POSIX Threads! This mix may lead to an undefined behavior.

4 Running MPC

The `mpcrun` script drives the launch of MPC programs with different types of parallelism. Its usage is defined as follows:

```
Usage mpcrun [option] [--] binary [user args]

Informations:
    --help,-h Display this help
    --show, Display command line
    --version-details, Print version of each module used
    --report, Print report
    --tmp_dir=dir, Directory to store mpc files
    --verbose,-v Verbose mode

Topology:
```

```
    --task-nb=n,-n=n Total number of tasks
    --process-nb=n,-p=n Total number of processes
    --cpu-nb=n,-c=n Number of cpus per process
    --node-nb=n,-N=n Total number of nodes
    --enable-smt Enable SMT capabilities (disabled by default)
    --share-node Restrict CPU number to share node

Multithreading:
    --multithreading=n,-m=n Define multithreading mode
        modes: pthread ethread_mxn ethread

Network:
    --network=n,-net=n Define Network mode
        modes:  none tcp ...
        modes (experimental): ...

Checkpoint/Restart and Migration:
    --checkpoint Enable checkpoint
    --migration Enable migration
    --restart Enable restart

Launcher:
    --launcher=n,-l=n Define launcher
    --opt=<options> launcher specific options
    --launch_list print available launch methods

Debugger:
    --dbg=<debugger_name> to use a debugger
```

4.1 Mono-process job

In order to run an MPC job in a single process, you should use on of the following methods (depending on the thread type you want to use).

```
mpcrun -m=ethread      -n=4 hello_world
mpcrun -m=ethread_mxn  -n=4 hello_world
mpcrun -m=pthread      -n=4 hello_world
```

4.2 Multi-process job on a single node

In order to run an MPC job in a 2-process single-node manner with the SHared Memory module enabled (*SHM*), you should use one of the following methods (depending on the thread type you want to use). Note that on a single node, even if the *TCP* module is explicitly used, MPC automatically uses the *SHM* module for all process communications.

```
mpcrun -m=ethread      -n=4 -p=2 -net=tcp hello_world
mpcrun -m=ethread_mxn  -n=4 -p=2 -net=tcp hello_world
mpcrun -m=pthread      -n=4 -p=2 -net=tcp hello_world
```

Of course, this mode supports both MPI and OpenMP standards, enabling the use of hybrid programming.

There are different implementations of inter-process communications. A call to `mpcrun --help` details all the available implementations.

4.3 Multi-process job on multiple nodes

In order to run an MPC job on 2 node with 8 processes communicating with *TCP*, you should use one of the following methods (depending on the thread type you want to use). Note that on multiple nodes, MPC automatically switches to the MPC SHared Memory module (*SHM*) when a communication between processes on the same node occurs. This behavior is available with all inter-process communication modules (*TCP* included).

```
mpcrun -m=ethread      -n=8 -p=8 -net=tcp -N=2 -l=ssh hello_world
mpcrun -m=ethread_mxn  -n=8 -p=8 -net=tcp -N=2 -l=ssh hello_world
mpcrun -m=pthread      -n=8 -p=8 -net=tcp -N=2 -l=ssh hello_world
```

Of course, this mode supports both MPI and OpenMP standards, enabling the use of hybrid programming.

There are different implementations of inter-process communications and launch methods. A call to `mpcrun --help` detail all the available implementations and launch methods.

4.3.1 Launch with ssh

In order to execute an MPC job on multile nodes using *ssh*, you need to fill the `$HOME/.mpcrun_tcp_host` file with the list of nodes to use.

5 FAQ

Q - How can I execute Fortan program on MPC ?

A - First, be sure that you don't have disabled the fortran support (`--disable-fortran` of the MPC configure). Second, rename your main fortran function by `subroutine mpc_user_main`. For example, change <<`program main_program`>> by <<`subroutine mpc_user_main`>> Now, you can execute your fortran program using the `mpcrun` command.

Q - How can I disable the MPC SHared Memory module (*SHM*) ?

A - The *SHM* module is enabled by default. To disable it, pass the argument `--disable-shm` to the MPC configure. Don't forget to recompile MPC.

Q - MPC configure gives me a "FATAL ERROR" message. What can I do ?

```
######################## FATAL ERROR ########################
MPC *WILL* overwrite the following include file(s):
include/mpi.h
include/pthread.h
include/semaphore.h
include/omp.h
```

```
Please set your --prefix correctly and run configure again.
If you know what you are doing, you can use the configure
flag --disable-prefix-check. Keep in mind that your headers
will be *DEFINITIVELY* overwritten.
Configure exiting with no Makefile generated....
############################################################
```

A - You must be careful with this error message. MPC detected that the prefix path you have given already includes header files. I.e: mpi.h, pthread.h, semaphore.h and omp.h. If you continue the MPC installation using this prefix path, these files will be *DEFINITIVELY* overwritten. As a conclusion, either you change the prefix path (recommanded version), or you pass the argument `--disable-prefix-check` to MPC configure, but your headers *WILL* be overwritten.

Q - How can I tune the MPC SHared Memory module (*SHM*) according to my needs ?

A - You need to edit the file located there : `mpc/MPC_Message_Passing/sctk_low_level_comm/sctk_shm_consts.h`. In this file, you can modify the number of cells in each queue (PTP queues, collective queues, etc...) as well as the size allocated by each cell. Don't forget to recompile MPC after it.

Q - When compiling, I have the error undefined reference to `mpc_user_main__`

A - The file containing your `main` should include `mpi.h` or `mpc.h`

6 Contacts

- CARRIBAULT Patrick patrick.carribault@cea.fr
- PÉRACHE Marc marc.perache@cea.fr
- DIDELOT Sylvain sdidelot@exascale-computing.eu
- VALAT Sébastien sebastien.valat@cea.fr

References

[1] Patrick Carribault, Marc Pérache, and Hervé Jourdren. Enabling loz-overhead hybrid MPI/OpenMP parallelism with MPC. In *To appear in International Workshop on OpenMP (IWOMP'10)*, Tsukuba, Japan, June 2010.

[2] Marc Pérache, Patrick Carribault, and Hervé Jourdren. MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption. In *16th European PVM/MPI Users' Group Meeting (EuroPVM/MPI 2009)*, Finland, September 2009.

[3] Marc Pérache, Hervé Jourdren, and Raymond Namyst. MPC: A unified parallel runtime for clusters of NUMA machines. In *Proceedings of the 14th International Euro-Par Conference (Euro-Par 2008)*, Las Palmas de Gran Canaria, Spain, August 2008.