Multi-Processor Computing framework *(MPC version 2.5.1)*

Getting Started guide

Introduction

This Getting Started guide details the various ways to install and configure the MPC library (Multi-Processor Computing) version including MPI 1.3, OpenMP 2.5 (with patched GCC) and PThread support. For further information on MPC internals, the reader may refer to documentation for more details about the MPC framework and its execution model.

Prerequisites

The following prerequisites are required to compile MPC:

- The main archive file from the Downloads page
- A *C* compiler (e.g., *gcc*)
- An optional *Fortran* compiler if Fortran applications are to be used (e.g., *g77* or *gfortran*)
- Optional *SLURM*, *HWLOC* and *OPENPA* libraries installed
- For *Infiniband* support *libibvers* is required

Standard Installation

These steps describe the default installation of MPC with its additional components (*GCC*,*GDB*, *Hydra*, *HWLOC* and *OPENPA*).

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

   ```
   $ tar xfz MPC_2.5.1.tar.gz
   $ cd MPC_2.5.1
   ```

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

   ```
   $ gunzip MPC_2.5.1.tar.gz
   $ tar xf MPC_2.5.1.tar
   $ cd MPC_2.5.1
   ```

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

   ```
   $ mkdir $HOME/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.

4. Launch the installation script for MPC and its dependencies. Specifying the installation directory:

   ```
   $ ./installmpc --prefix=$HOME/mpc-install
   ```

   Note: you can use the flag *-jX* to perform a faster installation. *X* should be a number and is the total amount of processes launched by *Make*.

5. Source the *mpcvars* script (located at the root of the MPC installation directory) to update environment variables (e.g., *PATH* and *LD_LIBRARY_PATH*).

   ```
   For *csh* and *tcsh*:

   $ source $(HOME)/mpc-install/mpcvars.csh

   For *bash* and *sh*:

   $ source $(HOME)/mpc-install/mpcvars.sh

   Check that everything went well at this point by running

   $ which mpc_cc
   $ which mpcrun

   If there was no errors, it should display the path of the mpc_cc and mpcrun scripts.
   ```

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

   ```
   $ mpc_cc mpcframework/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`
```

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

```
$ mpcrun -m=ethread      -p=1 -n=4 hello_world
$ mpcrun -m=ethread_mxn  -p=1 -n=4 hello_world
$ mpcrun -m=pthread      -p=1 -n=4 hello_world
```

See supported APIs for details on the '-m' option.

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.

Specify the PMI based launcher to be used by MPC as follows:

- **-with-hydra** *Compile MPC with the Hydra launcher (embedded in the MPC distribution). This option is enabled by default.*
- **-with-slurm[=prefix]** *Compile MPC with the SLURM launcher.*

Specify external libraries used by MPC:

- **-with-hwloc=prefix** *Specify the prefix of hwloc library
- **-with-openpa=prefix** *Specify the prefix of OpenPA library
- **-with-libxml2=prefix** *Specify the prefix of libxml2 library
- **-with-mpc-binutils=prefix** *Specify the prefix of the binutils

Specify external libraries for gcc compilation:

- **-with-gmp=prefix** *Specify the prefix of gmp library
- **-with-mpfr=prefix** *Specify the prefix of mpfr library
- **-with-mpc=prefix** *Specify the prefix of mpc multiprecision library

Specify external gcc and gdb:

- **-with-mpc-gcc=prefix** *Specify the prefix of the gcc used by MPC
- **-with-mpc-gdb=prefix** *Specify the prefix of the gdb used by MPC

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
    --verbose=n,-v,-vv,-vvv Verbose mode (level 1 to 3)
    --verbose Verbose level 1

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)
    --disable-share-node Do not restrict on CPU number to share node

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

Network:
    --network=n,-net=n Define Network mode (TCP + SHM by default)
        modes:  none ib 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
    --config=<file> Configuration file to load.
    --profiles=<p1,p2> List of profiles to enable in config.

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

Profiling (if compiled with --enable-MPC_Profiler) :
        --profiling=AA,BB,...

        example : --profiling=stdout,html,latex

        With the following output modules :
        * file : Outputs to file indented profile with time in standard units
        * file-raw : Outputs to file unindented profile in ticks (gnuplot compliant)
        * file-noindent : Same as "text" with no indentation
        * stdout : "text" to stdout
        * stdout-raw : "text_raw" to stdout
        * stdout-noindent : "text_noindent" to stdout
        * latex : Outputs profile to a latex source file
        * html : Outputs profile to an html file
```

Launcher options

Options passed to the launcher options should be compatible with the launch mode chosen during *configure*. For more information you might read the documentations of mpiexec and srun respectively for Hydra and Slurm.

- **Hydra**: If MPC is configured with Hydra, mpcrun should be used with *-l=mpiexec* argument. Note that this argument is used by default if not specified.
- **SLURM**: If MPC is configured with SLURM, mpcrun should be used with *-l=srun* argument.

Mono-process job

In order to run an MPC job in a single process with Hydra, 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
```

To use one of the above methods with SLURM, just add *-l=srun* to the command line.

Multi-process job on a single node

In order to run an MPC job with Hydra in a two-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
```

To use one of the above methods with SLURM, just add *-l=srun* to the command line.

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.

Multi-process job on multiple nodes

In order to run an MPC job on two nodes with eight 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 hello_world
$ mpcrun -m=ethread_mxn  -n=8 -p=8 -net=tcp -N=2 hello_world
```

```
$ mpcrun -m=pthread      -n=8 -p=8 -net=tcp -N=2 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. Acall to mpcrun -help detail all the available implementations and launch methods.

Launch with Hydra

In order to execute an MPC job on multile nodes using *Hydra*, you need to provide the list of nodes in a *hosts* file and set the HYDRA_HOST_FILE variable with the path to the file. You can also pass the host file as a parameter of the launcher as follow:

```
$ mpcrun -m=ethread -n=8 -p=8 -net=tcp -N=2 --opt='-f hosts' hello_world
```

See Using the Hydra Process Manager for more information about hydra hosts file.

FAQ

Q - How can I execute Fortan program on MPC ?

- 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 <> by <>. Now, you can execute your fortran program using the *mpcrun* command.

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

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

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 each modification.

Q - Why is my MPI program crashing when I use the MPC compilers?

In MPC, every MPI task is a thread and thus all tasks share global variables with each other. This sharing can cause undefined behavior, so we propose several solutions to ease the removal of global variables:

- Use the compiler 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 than in versions older than 2.4.0, the application has to be compiled as a dynamic library for this solution to work.
- 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.

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

The file containing your *main* should include *mpi.h* or *mpc.h*.

Q - Why won't HDF5 compile with MPC?

MPI-IO is not yet supported. MPC supports the MPI 1.3 standard, but MPI-IO was added in the MPI 2.0 standard.

Publications

1. S. Didelot, P. Carribault, M. Pérache, and W. Jalby, "Improving MPI Communication Overlap With Collaborative Polling," *in European MPI Users Group Meeting (EUROMPI'12)* , 2012. View PDF
2. A. Mahéo, S. Koliaï, P. Carribauilt, M. Pérache, and W. Jalby. "Adaptive OpenMP for Large NUMA Nodes," *in International Workshop on OpenMP (IWOMP'12)*, 2012. View PDF
3. M. Tchiboukdjian, P. Carribault, and M. Pérache, "Hierarchical Local Storage: Exploiting Flexible User-Data Sharing Between MPI Tasks," *in IEEE International Parallel and Distributed Processing (IPDPS'12)* , 2012. View PDF
4. P. Carribault, M. Pérache, and H. Jourdren, "Thread-Local Storage Extension to Support Thread-Based MPI/OpenMP Applications," *in International Workshop on OpenMP (IWOMP'11)* , 2011. View PDF
5. K. Pouget, M. Pérache, P. Carribault, and H. Jourdren, "User Level DB: a Debugging API for User-Level Thread

Libraries," *in Workshop on Multithreaded Architectures and Applications (MTAAP'10)* , 2010. View PDF

6. P. Carribault, M. Pérache, and H. Jourdren, "Enabling low-overhead hybrid MPI/OpenMP parallelism with MPC," *in International workshop on OpenMP (IWOMP'10)*, 2010. View PDF

7. M. Pérache, P. Carribault, and H. Jourdren, "MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption," *in 16th european PVM/MPI users group meeting (EuroPVM/MPI 2009)* , 2009. View PDF

8. M. Pérache, H. Jourdren, and R. Namyst, "MPC: a unified parallel runtime for clusters of NUMA machines," *in Proceedings of the 14th international EURO-PAR conference (EURO-PAR 2008)*, 2008. View PDF

News Article

CEA Ports the MPC Framework on Intel MIC Architure

Contacts

- CARRIBAULT Patrick patrick.carribault@cea.fr
- PÉRACHE Marc marc.perache@cea.fr
- JAEGGER Julien julien@cea.fr
- ParaTools Staff info@paratools.fr