MPI verification for MFIX

1. Introduction

This document outlines the basics steps to verify proper MPI installation prior to MFIX compilation for DMP execution. The objective is to provide a starting point for trouble shooting, and is not intended to be a comprehensive guide to MPI or Linux. Other operating systems such as Windows or MacOS might have similar procedures but this tutorial is particularly intended for Linux based operating systems.

It is assumed that the reader is familiar with basic Linux operating system commands and MPI terminology. Unfamiliar readers should take the time to learn the basic concepts used in the prerequisite section. If you have difficulties understanding these concepts, please consult with your organization. Most universities offer refresher courses in Linux and parallel programming concepts. There is also an abundance of material on the internet, or in any library that may be helpful to gain knowledge in these topics.

Some examples are given to illustrate the procedure. Linux commands must be entered at the command prompt (represented by the ">" sign, which is not part of the command itself). The output shown will likely be different on your system, due to the wide range of possible configurations. All examples shown herein are obtained with the gfortran compiler (version 4.7.2) and Open MPI project implementation of MPI-2 (http://www.open-mpi.org). The same procedure applies similarly for other Fortran compilers and the MPICH implementation (http://www.mcs.anl.gov/research/projects/mpich2). Please note that some Linux distributions come with a default MPI installation and mpif.h header file, which may cause some confusion for compiling MFIX in DMP mode. MPI library must be compiled with the same compiler that is intended for MFIX compilation (i.e., if using gfortran for MFIX compilation then have an MPI installation compiled with gcc and gfortran, alternatively if using Intel Fortran (ifort) compiler for MFIX compilation then need to have an MPI library that was compiled with Intel C/C++ and Fortran compilers).

2. Prerequisites

This section will show how to identify and set some environment variables. Properly following each step in this section should help detect and solve most common issues encountered when attempting to compile and run MFIX in DMP mode.

2.1. Fortran compiler

You should have a Fortran compiler installed on your machine. We will use the gfortran, but yours may be different. Type the following command with the argument at the command prompt:

> gfortran -v

This command will be useful both to check if the paths are set properly and also inquire about the version of the gfortran compiler. There may be a lot of information, including the version of the compiler. Below is the last line of the output, showing that version 4.7.2 is installed:

gcc version 4.7.2 (Ubuntu/Linaro 4.7.2-2ubuntu1)

NOTE: If you don't receive a response showing the compiler version or some other output indicating command not found, make sure gfortran compiler is installed on your system and your user account environment/path variables are set properly by consulting to your system administrator or using the information available on the internet for troubleshooting. It may be as simple as issuing load module gfortran if your system administrator uses modules.

2.2. Shell

You should know the Linux shell you are using, since the syntax of some commands vary from one shell to another. To inquire about your shell, type the following at the prompt:

> echo $SHELL

In this document, we will use the bash shell. The output from the previous command will return

/bin/bash

Your shell could be different on your machine (for example, the csh shell).

If you are using a different shell either find the equivalent syntax for the shell type you are using or switch to bash shell temporarily by issuing the following command:

> /usr/bin/bash

This will result in a temporary bash shell session only valid for the duration of your terminal session in that shell and usually changes the prompt symbol to indicate a different shell being used. Issuing the above "echo $SHELL" command is a good way to check the current shell type.

2.3. Environment variables

Environment variables define various file locations and other options that affect how processes run in Linux. For example, $SHELL is an environment variable. To display the value of a variable, use the "echo" command. To set the value of a variable, use the "export" command. For example, to assign the value /usr to the environment variable MPIHOME in *bash shell*, type

export MPIHOME=/usr

This will setup the location of the MPI installation. The actual path depends on your MPI implementation. If you do not know yet where MPI is installed, the actual location will be found in step 2.4. To view the value of MPIHOME environment variable, type

> echo $MPIHOME

which based on the above command will return

/usr

Notice the dollar sign in front of the variable name when using the environment variable in a Linux command such as displaying its value with echo command.

With the csh shell, the syntax to define an environment variable is different:

setenv MPIHOME /usr

Please note that the environment variables get reset every time you logon or open a new shell. In order to not having to define environment variables every time you logon or open a new terminal, it is useful to add the above lines to your default shell startup scripts, such as .bashrc for bash shell or .cshrc for csh shell or .tcshrc for tcsh. If you do not understand this, please consult your system administrator or utilize vast amount of information available on the internet.

2.4. MPI implementation

You should have either openmpi or mpich installed on your machine. If you do not know how to install MPI, please consult with your system administrator. In the next few subsections, we show how to set some of the key environment variables that are needed by MFIX in order to successfully compile with MPI.

2.4.1. MPIHOME

To verify where MPI is installed type:

> which mpif90

/usr/bin/mpif90

or alternatively you can query with “whereis” command as shown below, which can show the location of the requested entity independent of your path settings:

> whereis mpif90

mpif90: /usr/bin/mpif90.openmpi /usr/bin/mpif90 /usr/bin/X11/mpif90.openmpi /usr/bin/X11/mpif90 /usr/share/man/man1/mpif90.1.gz

This means we are using the openmpi implementation, and it is installed in /usr. Define MPIHOME with

export MPIHOME=/usr

2.4.2. Include and Library paths

Next, find some information about mpif90, which is simply a wrapper script that is linked to a compiler and sets the appropriate libraries from MPI automatically. To find the location of the include file, type:

> mpif90 -showme:incdirs

This will return the location where the MPI header file (mpif.h) for Fortran resides:

/usr/lib/openmpi/include

It will be used later when we compile MFIX in section 4. Typically, the makefile will detect the location, and no user-input should be required. You can verify that mpif.h is located in this folder by typing:

> ls /usr/lib/openmpi/include

A list of files will be displayed, including mpif.h.

To find the location of the MPI library files, type:

> mpif90 -showme:libdirs

The library path will be similar to:

/usr/lib/openmpi/lib

At this point, we will define the LD_LIBRARY_PATH environment variable:

> export LD_LIBRARY_PATH=/usr/lib/openmpi/lib:$LD_LIBRARY_PATH

> echo $LD_LIBRARY_PATH

/usr/lib/openmpi/lib:

We are actually prepending (adding at the beginning) the path to the variable. This is done so that this environment variable does not lose its old information. Note that the above variable might already be set correctly on your machine. Type "echo $LD_LIBRARY_PATH" (without the quotes) at the command line to verify the current information in this environment variable. If the correct path to MPI library is already present, then there is no need to do it again.

Note: We can also get the same information by typing:

> mpif90 –show

which returns the following on the current machine:

gfortran -I/usr/lib/openmpi/include -pthread -I/usr/lib/openmpi/lib -L/usr/lib/openmpi/lib -lmpi_f90 -lmpi_f77 -lmpi -lopen-rte -lopen-pal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl

Note that mpif90 is a wrapper, which means it is a script that runs a compiler with a set of other flags. The MPI implementation must be compatible with the Fortran compiler. Here, we are using gfortran (highlighted in blue). The include file location is highlighted in red, and the library file location is highlighted in green.

2.5. Checklist

You should now be able to do or identify the following (as determined in a previous section identified in parenthesis):

- ☐ Which Fortran compiler you are using (Section 2.1)
- ☐ What is your Linux distribution and current terminal shell type (Section 2.2)
- ☐ Setup and viewing an environment variable (Section 2.3)
- ☐ Which MPI implementation is installed on your system (Section 2.4)
- ☐ Where MPI is installed, and define MPIHOME accordingly (Section 2.4.1)
- ☐ Location of the MPI header file mpif.h (include path) (Section 2.4.2)
- ☐ MPI library path, and define LD_LIBRARY_PATH accordingly (Section 2.4.2)

NOTE: Do not go to the next section if you have difficulties in identifying any of the items in the check list above. Consult with your system administrator if you are not sure how to proceed. ***Any help request submitted to mfix-help must include all the information in this check list.***

3. Testing MPI Setup with a Simple Program

We can now test MPI with a simple Hello World example, located in ~/mfix/tools/mpi. Type:

> cd ~/mfix/tools/mpi

> mpif90 hello_world.f90 -o hello_world.exe

to compile the file. To run the program with 4 processors, enter the following:

> mpirun -np 4 hello_world.exe

This should return an output similar to the following if MPI is working properly:

```
Hello world from rank      1  of      4
Hello world from rank      3  of      4
Hello world from rank      2  of      4
Hello world from rank      0  of      4
```

Each processor displays a simple "Hello world" message, along with its rank. Please note that the order of the messages will be different on your machine, and will vary each time the program is executed.

Also be aware of the system architecture you are running. Today's multicore platforms enable 4 to 12 cores on the same processor socket, with several sockets per motherboard. Hence, running only 4 processor test might not reveal problems between sockets or nodes, which rely on the network protocol offered through MPI installation. It is highly recommended to test sufficiently high number of processors to make sure the MPI installation can communicate across sockets or nodes using the interconnect (e.g. Infiniband) or alternatively use the mpirun arguments to force execution by only 1 MPI rank per socket if you would like to test the communication over the interconnect.

NOTE: Do not go to the next section if you cannot successfully compile and run the Hello World example as the root cause of the problem is going to affect MFIX DMP runs. **Consult with your system administrator if you are not sure how to proceed.**

4. Compile MFIX for DMP execution, and run a DMP test case

4.1. Compilation

Go to the fluidbed1_dmp_test folder and invoke the makefile by typing:

> cd ~/mfix/tutorials/fluidbed1_dmp_test

> mkmfix

==========

Mode of execution:

==========

[1] Serial

[x] Parallel, Shared Memory (SMP) - Temporarily unavailable

[3] Parallel, Distributed Memory (DMP)

[x] Parallel, Hybrid (SMP+DMP) - Temporarily unavailable

Select the mode of execution [1] : 3 ← **Type 3 and Press Enter**

==========

Level of Optimization:

==========

[0] None (Debug mode)

[x] Level 1 (not available)

[x] Level 2 (not available)

[3] Level 3 (most aggressive)

Select the level of optimization [3] : ← **Press Enter**

==========

Option to re-compile source files in run directory:

==========

[1] Do not force re-compilation

[2] Force re-compilation

Select Option to re-compile source files in run directory [1] : ← **Press Enter**

Next, the makefile will verify that the mpif.h file is found (this should be consistent with what was found in section 2.4.2), and will display a list of compilers.

64 bit Linux system detected, please select compiler.

checking for mpif.h in the default directory /usr/lib/openmpi/include

mpif.h was found.

==========

MFIX Compilation directives available for following compilers:

==========

[1] GNU (gfortran) version 4.3 and above

[2] Portland Group (pgf90) version 11.7 and above

[3] Intel (ifort) version 11.1 and above

Select the compiler to compile MFIX? [1] ← **Press Enter**

Compilation and linking may take from few minutes to ten or more minutes, depending on the compiler selected and its interprocedural optimization phases. For gfortran it shouldn't take more than few minutes. At the end of the compilation, you should have the executable mfix.exe file in the run directory.

****

Compilation successful: mfix.2013-1 created

To run MFIX type (or equivalent): mpirun -np<# processors> mfix.exe

****

4.2. Additional verifications (optional)

Check the date and time stamp of the executable as a secondary check:

> ls -als mfix.exe

5200 -rwxrwxr-x 1 ubuntu ubuntu 5310086 Feb 8 11:01 mfix.exe

Verify that mpif.h is properly pointing to the correct location:

> ls -als ~/mfix/model/mpif.h

0 lrwxrwxrwx 1 ubuntu ubuntu 31 Feb 8 11:01 /home/ubuntu/mfix/model/mpif.h -> /usr/lib/openmpi/include/mpif.h

Make sure all the dynamically linked libraries necessary for the executable to run successfully can be seen with your current shell environment. To check this, issue the following command which checks for the dynamically linked libraries in the executable and shows appropriate paths if you were to launch it:

> ldd ./mfix.exe

An output similar to the following should be obtained:

linux-vdso.so.1 => (0x00007fff6b3f9000)

libmpi_f77.so.0 => /usr/lib/libmpi_f77.so.0 (0x00007f7e395f9000)

libgfortran.so.3 => /usr/lib/x86_64-linux-gnu/libgfortran.so.3 (0x00007f7e392e5000)

libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f7e38fe8000)

libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f7e38dd2000)

libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7e38a13000)

libmpi.so.0 => /usr/lib/libmpi.so.0 (0x00007f7e38762000)

libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f7e38545000)

libquadmath.so.0 => /usr/lib/x86_64-linux-gnu/libquadmath.so.0 (0x00007f7e3830f000)

/lib64/ld-linux-x86-64.so.2 (0x00007f7e39842000)

libopen-rte.so.0 => /usr/lib/libopen-rte.so.0 (0x00007f7e380c0000)

libopen-pal.so.0 => /usr/lib/libopen-pal.so.0 (0x00007f7e37e68000)

libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f7e37c64000)

libutil.so.1 => /lib/x86_64-linux-gnu/libutil.so.1 (0x00007f7e37a60000)

If the output shows dynamic libraries that are not found (as shown below), you should identify these libraries and append the appropriate path to the $LD_LIBRARY_PATH environment variable as shown earlier:

linux-vdso.so.1 => (0x00007fff6b3f9000)

libmpi_f77.so.0 => /usr/lib/libmpi_f77.so.0 (0x00007f7e395f9000)

libgfortran.so.3 => not found ←**one of the library cannot be found**

libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f7e38fe8000)

……

4.3. Run MFIX in DMP mode

The fluidbed1_dmp_test case is setup to use 4 processors (make sure NODESJ = 4 in mfix.dat). To run MFIX, type

> mpirun -np 4 mfix.exe

The simulation should start and take a few minutes to complete. Note that this test case only verifies proper MPI installation, and no speed-up should be expected with such a small test case. **If the simulation is successful, it means you are ready to run MFIX in DMP mode with larger scale simulations.**

Notes:

1) MFIX has the capability to enable each processor to write out a LOG file (casenameXXXX.LOG where XXXX represents the processor number). The default setting is to enable only root processor (processor # 0) to output the LOG file. However, sometimes it might be necessary to have all processors write the log file especially when recurring errors from one particular processor is observed during execution or restarts. To enable the LOG file output from all processors, set the following variable to TRUE in mfix.dat: ENABLE_DMP_LOG = .TRUE.
2) To submit an MFIX job on a cluster through a batch queue system, consult with you system administrator, since the procedure is highly dependent on the batch queuing software and hardware configuration of the system.