ITCS 4145/5145 Assignment 1

Exercises in compiling and running simple MPI programs

Author: B. Wilkinson. Modification date: Jan 17, 2012

This assignment is to become familiar with the installation and should be very straightforward. We will use the coit-grid cluster. The programming environment is Linux. An implementation of MPI called MPICH is installed. In MPICH, there are two commands that will be used mainly: **mpicc**, the command (a script) to compile MPI programs, and **mpiexec** the command to execute a MPI program. You may use any editor available such as **vi** or **nano** (preferred) when asked to alter files. When downloading files with a browser, be careful to make sure the file name extension and/or the contents have not been altered with HTML tags.

Preliminary tasks (5 minutes, 5%)

Task 1: Connect to coit-grid01.uncc.edu

Connect to into coit-grid01.uncc.edu using Putty (or another ssh client).

Task 2: Check MPI commands

First check the implementation and version of these commands with:

```
which mpicc
which mpiexec
```

The full paths should be returned.

Task 3: Set up a directory for the assignments

Make a directory called mpi that will be used for the MPI programs in this course, and “cd” into that directory The commands are:

```
mkdir mpi_assign
cd mpi_assign
```

All mpi commands will be issued from this directory.

Exercise 1 "Hello World" (30 minutes, 35%)

Task 1: Executing a simple Hello World program.

Copy the C program hello.c into mpi_assign. This program is given below:

```
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include "mpi.h"
```

```
main(int argc, char **argv ) {
  char message[20];
  int i,rank, size, type=99;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if(rank == 0) {
    strcpy(message, "Hello, world");
    for (i=1; i<size; i++)
      MPI_Send(message, 13, MPI_CHAR, i, type, MPI_COMM_WORLD);
  }
  else
    MPI_Recv(message, 20, MPI_CHAR, 0, type, MPI_COMM_WORLD, &status);

  printf( "Message from process =%d : %.13s\n", rank,message);
  MPI_Finalize();
}
```

The program sends the message "Hello, world" from the master process, (rank = 0) to all the slave processes (workers, rank != 0). The slave processes receive the messages and then all print the messages out to stdout,

Compilation:

Compile and execute the hello program using four processes in total. To compile the program use the script:

```
mpicc -o hello hello.c
```

which uses the gcc compiler (probably) to links in the libraries and create an executable hello, and hence all the usual flags that can be used with gcc can be used with mpicc.

Execution:

For the MPI installation on coit-grid01.uncc.edu, it is necessary to submit the application to **mpiexec** with the specific path to the executable, i.e.:

```
mpiexec -n 4 ./hello
```

where `./` denotes the path to current directory (or give its full path) otherwise, you will get a "File not found" error. The full path is `/nsf-home/username/...` .

So far, four instances of the program will execute just on coit-grid01.unc.edu. You should get the output:

```
Message from process =0 : Hello, world
Message from process =2 : Hello, world
Message from process =1 : Hello, world
Message from process =3 : Hello, world
```

The order of messages may be different; it will depend upon how the four processes are scheduled on coit-grid01.uncc.edu.

For convenience, for subsequent programs, you may use the default executable name, a.out, if you wish.

Task 2 Modification so that master prints out all messages

Modify the hello.c program so that the slave processes do not issue the print statements, but each sends a message back to the master containing their rank After the master receives each message, it print out a message containing the slave's rank. The program should produce output such as:

```
Master: Hello slaves give me your messages
Message received from process 1 : Hello back
Message received from process 2 : Hello back
Message received from process 3 : Hello back
```

Now, all print statements are issued by the master.

Clue: MPI_Comm_rank returns the processes rank using the argument rank.

Task 3: Different messages from each process

Alter the code so that each process send a different message back, e.g.:

```
Master: Hello slaves give me your messages
Message received from process 1 : Hello, I am John
Message received from process 2 : Hello, I am Mary
Message received from process 3 : Hello, I am Susan
```

Task 4: Experiments with tags

In the previous programs, the tag of 99 is used in messages. Modify the program from task 3 so that the master process sends a messages to each slave, but with the tag of 100 and the slave waits for message with a tag of 100. Confirm program works.

Repeat but make the slaves wait for tag 101, and check program hangs. Why?

Exercise 2 Using multiple computers (30 minutes, 35%)

In the previous exercise, only one computer was used, coit-grid01.unc.edu, with multiple processes time-shared on the single processor. In the following exercise, multiple computers will be used. Also the time of execution will be measured.

Task 1: Determining the available servers in the coit-grid cluster

At various times, not all servers are available. For MPI programs to operate between servers, the mpi daemons mpd must be running on each server to form a ring.

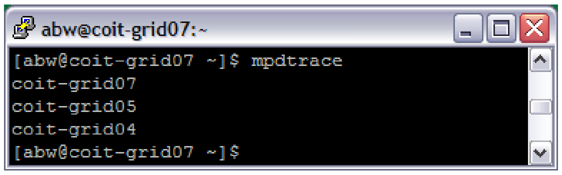
Execute the command:

```
mpdtrace
```

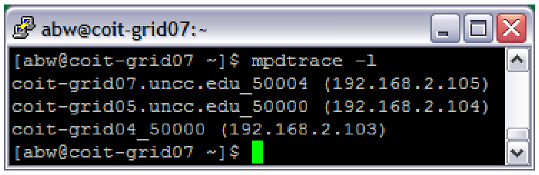
or

```
mpdtrace -l
```

which will list those servers where the mpd is running, for example:



```
[abw@coit-grid07 ~]$ mpdtrace
coit-grid07
coit-grid05
coit-grid04
[abw@coit-grid07 ~]$
```



```
[abw@coit-grid07 ~]$ mpdtrace -l
coit-grid07.uncc.edu_50004 (192.168.2.105)
coit-grid05.uncc.edu_50000 (192.168.2.104)
coit-grid04_50000 (192.168.2.103)
[abw@coit-grid07 ~]$
```

Make a note of the servers and use these servers in the following.

Task 2: Setting up the list of machines to use

The usual way MPI identifies the computers (machines) that it can use for executing a program is by listing them in a file, and using the -machinefile flag with mpirun, i.e, if the file containing the list of machines is called machines:

```
mpiexec -machinefile machines -n 4 ./a.out
```

would run `a.out` with four processes. Each processes would execute on one of the machines in the list. MPI would cycle through the list of machines giving processes to machines. (One can also specify the number of processes on a particular machine by adding that number after the machine name.)

Because of the way the cluster is set up with servers having dual internal and external Ethernet connects, each such server has local name. coit-grid01.uncc is called grid01 and so on.

Create a file called machines containing the list of machines, *using the local names*, e.g.

```
grid01:4
grid02:4
grid03:4
grid04:4
```

These domain names point to the internal IP addresses or you can use local IP addresses. Do not use external IP addresses or names on this cluster.

Each of the computers consists of two hyperthreaded Intel processors. Each can naturally support up to four processes at the same time.)

Task 3: Running a simple job

Execute the hello.c program from Exercise 1 with 4 computers and 16 processes, i.e. execute the command:

```
mpirun -machinefile machines -n 16 ./hello
```

After accepting the hosts, you should get the output such as

```
Message from process =0 : Hello, world
Message from process =3 : Hello, world
Message from process =1 : Hello, world
Message from process =2 : Hello, world
.
.
.
```

but in an indeterminate order. *Explain the output.*

Task 4 Identifying the machine names

In the previous task, we could not tell which computer is executing which process. Modify the program hello.c to also output the machine name, and generate output such as follows:

```
Message from coit-grid04.unc.edu process =0 : Hello, world
Message from coit-grid02.unc.edu process =3 : Hello, world
Message from coit-grid01.unc.edu process =1 : Hello, world
Message from coit-grid03.unc.edu process =2 : Hello, world
.
.
.
```

A C statement to obtain the hostname is `gethostname(&myname,80)` where `myname` is declared as `char myname[80];`

Task 5: Measuring time of execution

Add code to the previous program to measure the time of computation using `MPI_Wtime()` which returns the elapsed time from some point in the past, in seconds. Hence one can instrument the code with:

```
double start_time, end_time, exe_time;
start_time = MPI_Wtime();
    .
    .
    .
end_time = MPI_Wtime();
exe_time = end_time - start_time;
```

Experiment with running the program on one computer and on multiple computers. Does it go faster on multiple computers?

Apart from instrumenting the code as above to get execution time, one might consider using the Linux time command. Repeat measuring the time with the time command, i.e. time <command>, and comment upon the results.

Exercise 3 Using Broadcast operation (30 minutes, 25%)

Re-write the hello world program hello.c to use the MPI broadcast routine rather than send and receives. The master broadcasts a "Hello World" message to the slaves and slaves print out the messages. Both the master and slaves execute the same broadcast routine. The master is identified as the root (rank 0). The 4th argument/parameter of the broadcast routine is zero in all cases.

Does the broadcast operation reduce the execution time? Discuss.

Grading

Every task and subtask specified will be allocated a score so make sure you clearly identify each part you did.

Assignment Submission

Produce a document that show that you successfully followed the instructions and performs all tasks by taking screen shots and include these screen shots in the document. Give at least one screen shot for each numbered task in each exercise. (To include screen shots from Windows XP, select window, press Alt-Printscreen, and paste to file.) Provide insightful conclusions. Submit by the due date as described on the course home page. **Include all code, not as screen shots of the listings but complete properly documented code listing.**