



A MANUAL FOR MPICH



A MANUAL FOR MPICH

Submitted By:

Kishan Kumar Ganguly

BSSE0505

Institute of Information Technology

University of Dhaka

Submitted To:

Amit Seal Ami

Lecturer

Institute of Information Technology

University of Dhaka

Date of Submission

13 June, 2015

LETTER OF TRANSMITTAL

13th June, 2015.

Amit Seal Ami

Lecturer

Institute of Information Technology

University of Dhaka

Sir,

I have conducted a thorough study on MPICH under your guidance. This report reflects my findings on the mentioned subject.

The objective of this report is to provide a short documentation on MPICH and also present the problems faced at different levels while working with this tool. I gathered the information from my hands on experience with this tool and also from different sources from internet.

Finally, I would like to thank you for the opportunity this report gave me to familiarize myself with MPICH and to develop my skills, which will be invaluable for future researches.

Sincerely Yours,

Kishan Kumar Ganguly (BSSE-0505)

Enclosure: Report

Contents

Preface	1
Chapter 1	2
Introduction.....	2
1.1 Parallel Computation Model.....	2
1.1.1 Shared Memory.....	2
1.1.2 Message Passing.....	2
1.1.3 Message Passing Model	3
1.2 Message Passing Interface (MPI).....	3
1.3 Goals of MPI	4
1.4 MPICH	4
1.5 Conclusion	4
Chapter 2	5
Hello World!	5
2.1 Hello World Code	5
2.2 Installation.....	5
2.3 Running the Hello World Program.....	7
2.4 Understanding the code.....	9
2.5 Conclusion	9
Chapter 3	10
Setting up A Cluster.....	10
3.1 MPICH Cluster Running the Hello World Program.....	11
3.2 Conclusion	18
3.3 Troubleshooting Guide.....	19
Chapter 4	21
Send and Receive	21
4.1 Definition	21
4.2 Implementation.....	22
4.3 Conclusion	25
4.4 Troubleshooting Guide.....	26
References:.....	27

This Page Intentionally Left Blank

Preface

MPICH has been very popular for parallel processing now-a-days. In fact it is one of the most popular MPI implementations out there. But sadly there are almost no good or easy tutorials for the beginners to get started. Some those are easy ones, often do not work well.

Also people seem to face several issues while programming on MPICH. Those issues need to be resolved as well.

In this report, I have tried to gather the resources around and make the presentations as easy as possible. It should be good enough for someone who is stepping into the MPICH world. It does not discuss any advanced stuffs. Still basics are discussed properly to help any one's future learning.

I hope that this report come out successful on the purpose it was written for.

Kishan Kumar Ganguly

BSSE 0505

Chapter 1

Introduction

In the modern world, computations are complex and thus time consuming. Data manipulations on real time systems and scientific modeling need quick and correct results which is extremely difficult to achieve because we need to operate on billions even trillions of data (or sometimes even more!). Till now, even the most powerful processor of the world (claimed to be created by IBM, 5.5 GHz, Hexa core) does not seem to operate on such humongous amount of data within a satisfactory time limit. Thus if we could increase our resources, we could have achieved a better execution time. Also, we would have the chance to scale our system to the increased problem size which is mandatory in some applications such as image and signal processing. These demands over the time have opened the door to the world of parallel programming.

1.1 Parallel Computation Model

Parallel computation models can be divided into following categories:

- a. Directive Based
- b. Transparent
- c. Task Farming
- d. Shared Memory
- e. Message Passing

Also other models are available such as OpenMP. Of the five models stated above we will discuss the more important two: shared memory and message passing.

1.1.1 Shared Memory

On shared memory model, a memory is shared between multiple processors. Also communication between processors are done by shared variables on the memory.

1.1.2 Message Passing

The main problem of shared memory model is that they share the same memory space. So concurrency issues arise if multiple processes try to access the same memory block. On message passing model, single process run on single processor computer. We describe the message passing model on brief on the next section.

1.1.3 Message Passing Model

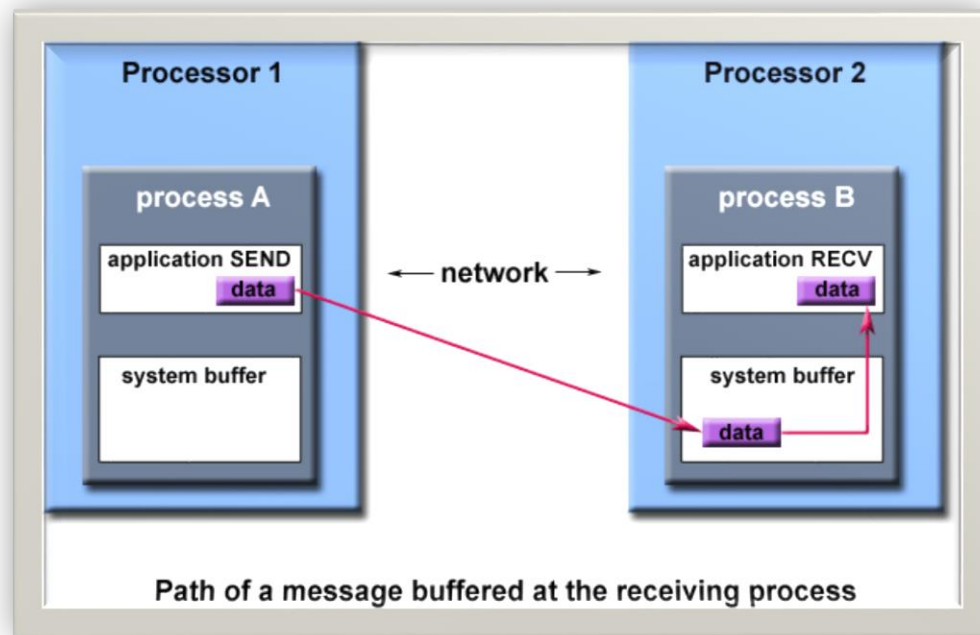


Fig: The Message Passing Model

[Image Source: <https://computing.llnl.gov/tutorials/mpi/>]

- i. Processes perform their task on different computers each having a local memory.
- ii. No shared memory is present.
- iii. If a processor needs to share something it simply sends the data to the other processor.
- iv. The other processor receives the data and data wait in the system buffer until they are used by the second processor.
- v. There is no shared memory. So a processor does not need to worry about what another processor is doing on the data. It just performs its assigned tasks.

1.2 Message Passing Interface (MPI)

MPI is a specification for a message passing library for parallel processing with message passing model. It is not a product or a direct implementation, rather it says how a proper implementable message passing model should be. It has gathered the most exciting and promising features of different implementation of message passing over the years and its main goal is to standardize the message passing system.

1.3 Goals of MPI

- a. Standardize different implementations.
- b. Allow a parallel program network to be consist of various different processors of heterogeneous types.
- c. The programming interface should be language independent.
- d. Reliable and efficient communication.
- e. Platform independent and thread safe.

1.4 MPICH

MPI stands for Message Passing Interface as mentioned before. And the ch stands for "Chameleon", which was another parallel programming library developed by one of the founders of MPICH, William Gropp. MPICH is one of the most popular implementations of MPI. It is portable and a implementation of the standard (MPI-1, MPI-2 and MPI-3). It is open source. High performance and portability across different platforms is the main reason for its popularity.

1.5 Conclusion

We have discussed in this chapter some basic stuffs about parallel processing and MPI. Now we will directly jump into the main topic, MPICH. The source codes provided here are written in C. However, sources codes can also be written in C++ or Fortran.

Chapter 2

Hello World!

In this chapter we learn to write a basic hello world program. First we write for parallel processing on a single computer. Then we extend our simple project for multiple PCs.

2.1 Hello World Code

```
#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int rank;
    int size;

    MPI_Init( 0, 0 );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Write the code into notepad or any of your favorite text editor. For now do not focus on the code, just write it down and move onto the next step where we set up our environment for MPICH.

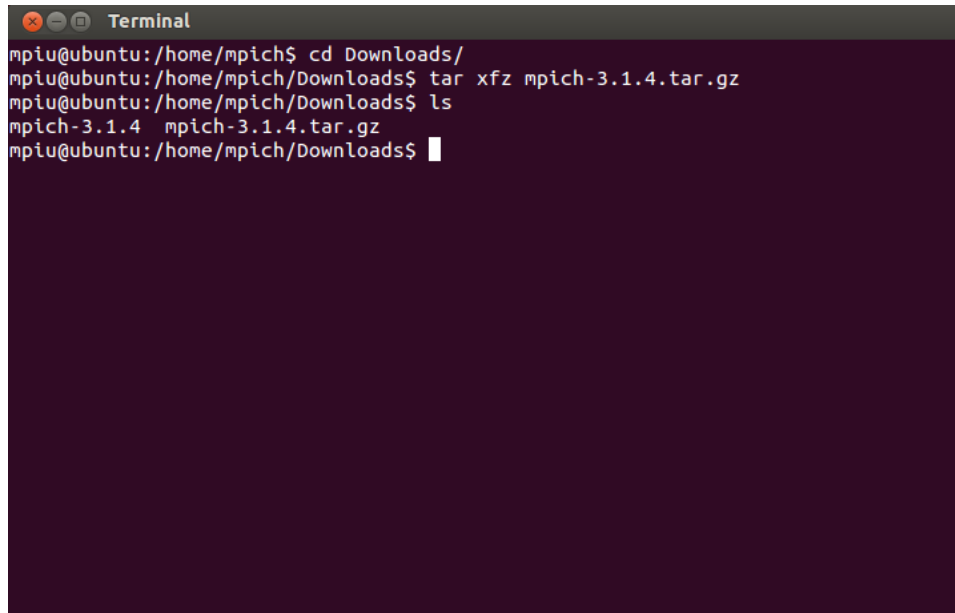
2.2 Installation

We will work on the Linux operating system and assume that the readers are also on the same platform. However for installation on other operating systems you can lookup Google. There are several resources available which we have listed down to references. You can run the following command on the terminal to install MPICH.

`sudo apt-get install libcr-dev mpich2` (Be careful, other installations of MPI may cause problems)

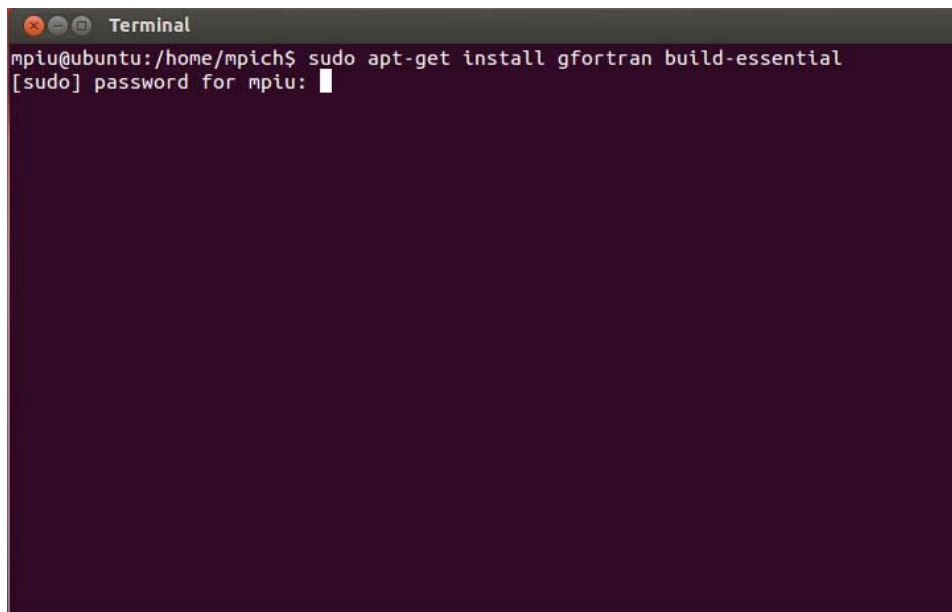
However it is not recommended because you may not get the latest version. So you do it manually. Download the stable release of MPICH from here <https://www.mpich.org/downloads/>.

- a. First we download the file and extract it. To extract open terminal (CTRL+ALT+T). Then enter:
- ```
tar xzf mpich-version.tar.gz mpich-version.tar.gz.
```

A terminal window titled "Terminal" with a dark background. The prompt is "mpiu@ubuntu:/home/mpich\$". The user enters "cd Downloads/" and the prompt changes to "mpiu@ubuntu:/home/mpich/Downloads\$". Then the user enters "tar xzf mpich-3.1.4.tar.gz" and the prompt changes to "mpiu@ubuntu:/home/mpich/Downloads\$". Finally, the user enters "ls" and the output "mpich-3.1.4 mpich-3.1.4.tar.gz" is displayed.

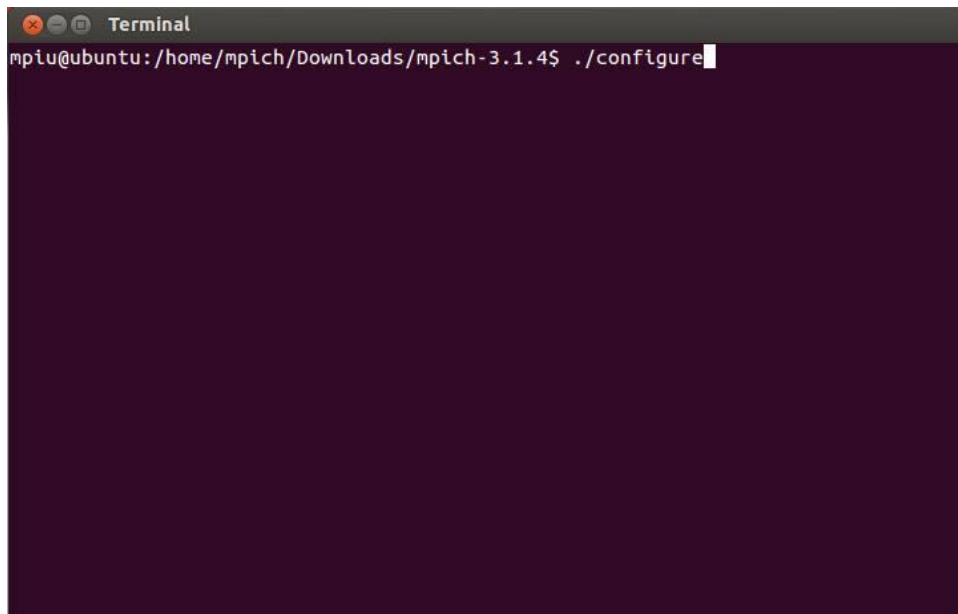
```
mpiu@ubuntu:/home/mpich$ cd Downloads/
mpiu@ubuntu:/home/mpich/Downloads$ tar xzf mpich-3.1.4.tar.gz
mpiu@ubuntu:/home/mpich/Downloads$ ls
mpich-3.1.4 mpich-3.1.4.tar.gz
mpiu@ubuntu:/home/mpich/Downloads$
```

Then install build-essential and gfortran.

A terminal window titled "Terminal" with a dark background. The prompt is "mpiu@ubuntu:/home/mpich\$". The user enters "sudo apt-get install gfortran build-essential" and the prompt changes to "[sudo] password for mpiu:". The user then enters their password, represented by a series of dots.

```
mpiu@ubuntu:/home/mpich$ sudo apt-get install gfortran build-essential
[sudo] password for mpiu: .
```

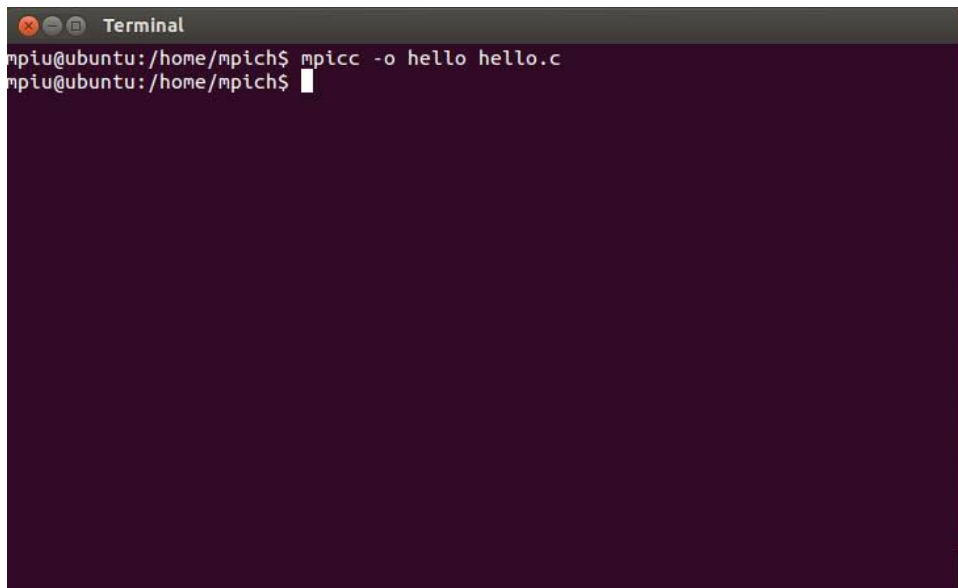
- b. Then navigate to the folder where you extracted MPICH and run a `./configure`.



- c. Then run `sudo make install`. And your environment is ready.

## 2.3 Running the Hello World Program

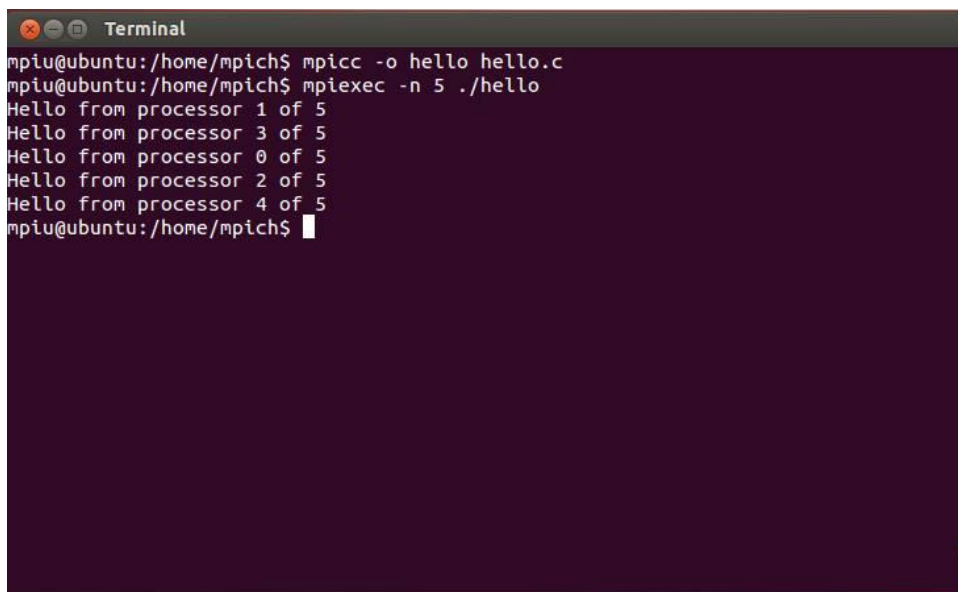
- a. Navigate to the folder from terminal where you kept the code mentioned before.
- b. Now run the command:  
`mpicc -o hello hello.c`



```
Terminal
mpiubuntu:/home/mpich$ mpicc -o hello hello.c
mpiubuntu:/home/mpich$
```

Here hello is the executable file's name. We may set it anything you like.

- c. Now type in the following command: `mpiexec -n 5 ./hello`



```
Terminal
mpiubuntu:/home/mpich$ mpicc -o hello hello.c
mpiubuntu:/home/mpich$ mpiexec -n 5 ./hello
Hello from processor 1 of 5
Hello from processor 3 of 5
Hello from processor 0 of 5
Hello from processor 2 of 5
Hello from processor 4 of 5
mpiubuntu:/home/mpich$
```

Here `mpiexec` is the routine that will provide parallel execution of our program on runtime. By specifying `-n`, we mention the number of processes we would like to use. Here we have run the program for 5 processes. And next is the name of the executable file. And we will see the 5 lines mentioning processor name and processor count. Next we proceed to understanding the hello world program.

## 2.4 Understanding the code

---

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
 int rank;
 int size;

 MPI_Init(0, 0);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 printf("Hello world from process %d of %d\n", rank, size);
 MPI_Finalize();
 return 0;
}
```

From the second line we included the mpi.h library which is necessary for using the mpi specific functions and variables in our code. After the directive, into the main function we take two variables rank and size. Size is simply the number of processors that we mention after the -n part of our previous command. We discuss rank a bit later. Then MPI\_init initializes the execution environment. The two parameters are pointer to the number of arguments and pointer to the argument vector. Now we define groups to be a ordered set of unique processes. The processes of a group are ranked from 0 to N-1, where N is the number of processes in a group. A communicator is a collection of groups that define the scope of the communication, that is, the valid participants for communicating within a group or among groups. The process ranks within the group comprises the sender and receiver addresses of the message. Here MPI\_COMM\_WORLD is the default communicator and consists of all the processes. MPI\_COMM\_RANK thus outputs the rank of the processor in the rank variable. And MPI\_COMM\_SIZE outputs the number of processes. Now as the program runs on our computer on different processors, we get its rank and total number of processes and thus the result.

## 2.5 Conclusion

We have learnt a basic hello world program. On the next chapter clusters will be discussed.

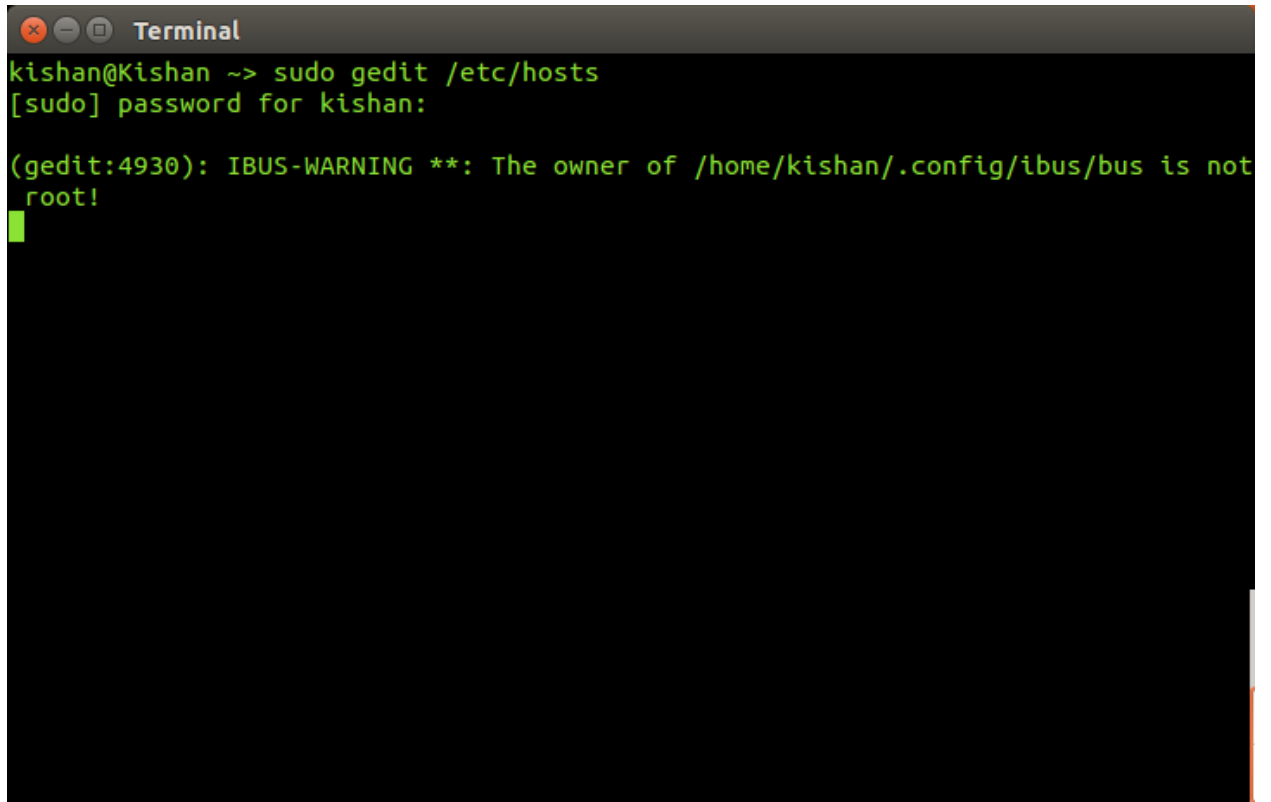
## **Chapter 3**

### **Setting up A Cluster**

Now we will run our code on multiple PC and set up a cluster. We will directly jump into the implementation in the following section.

### 3.1 MPICH Cluster Running the Hello World Program

- a) At first open terminal (CTRL+ALT+T) and type in the `sudo gedit /etc/hosts`.

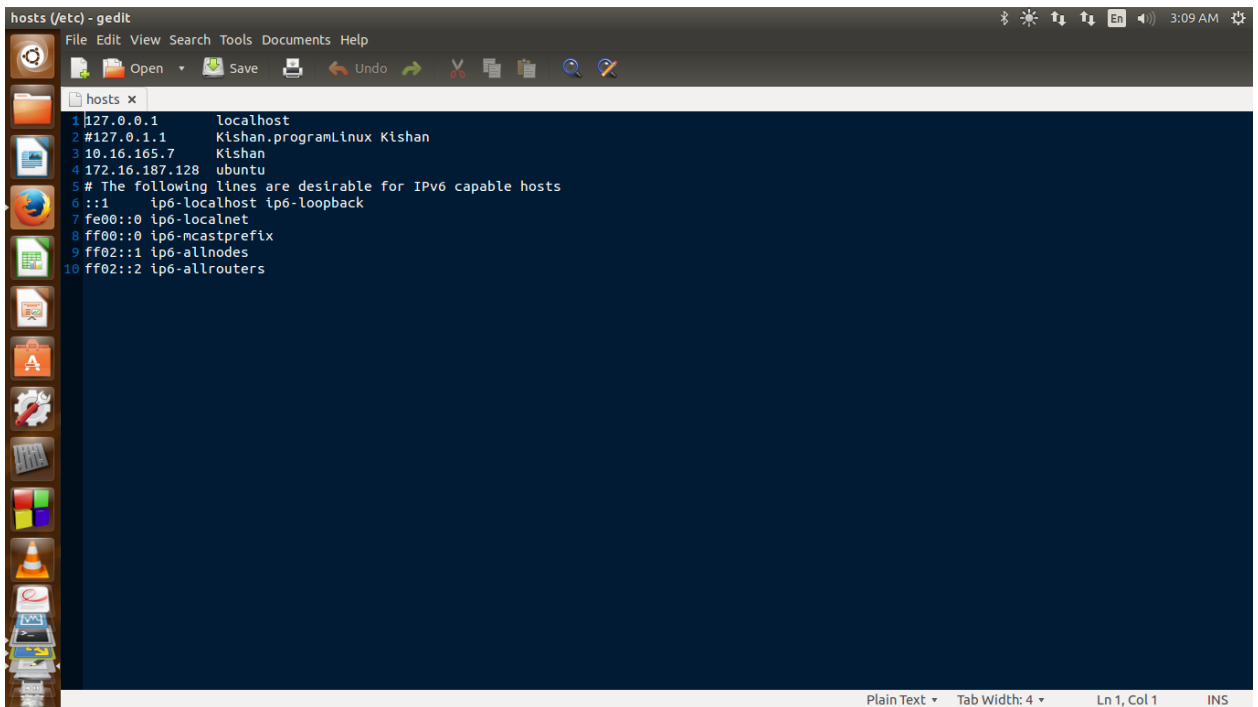
A terminal window titled "Terminal" with a dark background and light green text. The prompt is "kishan@Kishan ~>". The user enters "sudo gedit /etc/hosts". The prompt changes to "[sudo] password for kishan:". The user presses enter. The prompt changes to "(gedit:4930):". An IBUS warning is displayed: "IBUS-WARNING \*\*: The owner of /home/kishan/.config/ibus/bus is not root!". A light green cursor is visible on the line following the warning.

```
kishan@Kishan ~> sudo gedit /etc/hosts
[sudo] password for kishan:
(gedit:4930): IBUS-WARNING **: The owner of /home/kishan/.config/ibus/bus is not
root!
```

Then edit the file. First comment out the 127.0.1.1 line. Then add your computer's ip address and a name. Any name would do as it is an alias. But I am using my pc's hostname here. Also type in the other computer's ip address and name. Repeat the same thing on

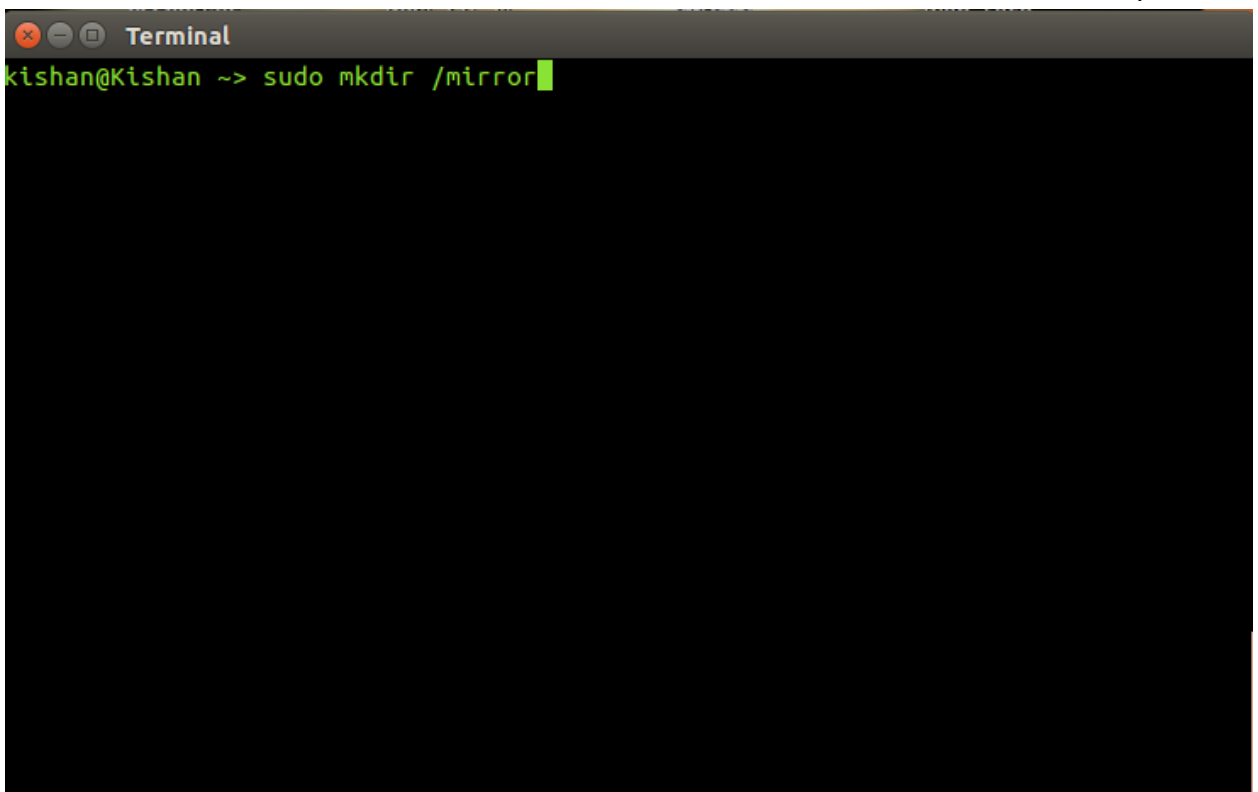


the second computer also. After that simply save and close the file.



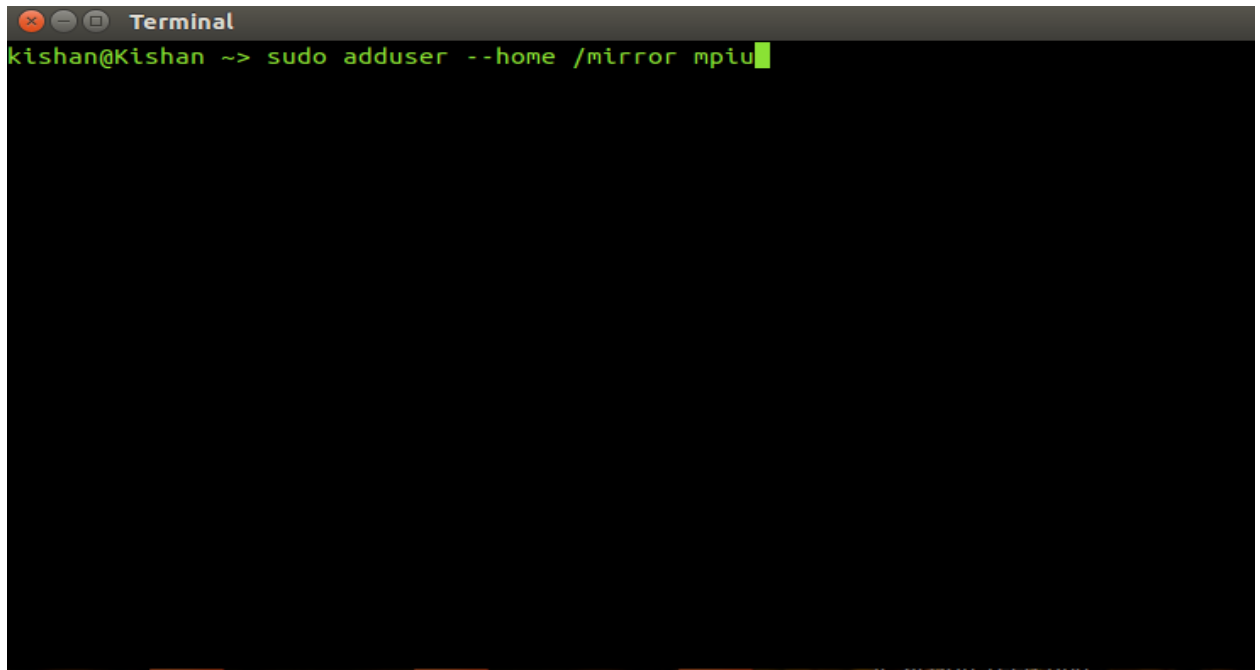
```
hosts (/etc) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
hosts x
1 #127.0.0.1 localhost
2 #127.0.1.1 Kishan.programLinux Kishan
3 10.16.165.7 Kishan
4 172.16.187.128 ubuntu
5 # The following lines are desirable for IPv6 capable hosts
6 ::1 ip6-localhost ip6-loopback
7 fe00::0 ip6-localnet
8 ff00::0 ip6-mcastprefix
9 ff02::1 ip6-allnodes
10 ff02::2 ip6-allrouters
```

- b) Now create a directory. I will name it mirror but anything would do. Type in the commands in the terminal `sudo mkdir /mirror` to create the directory.

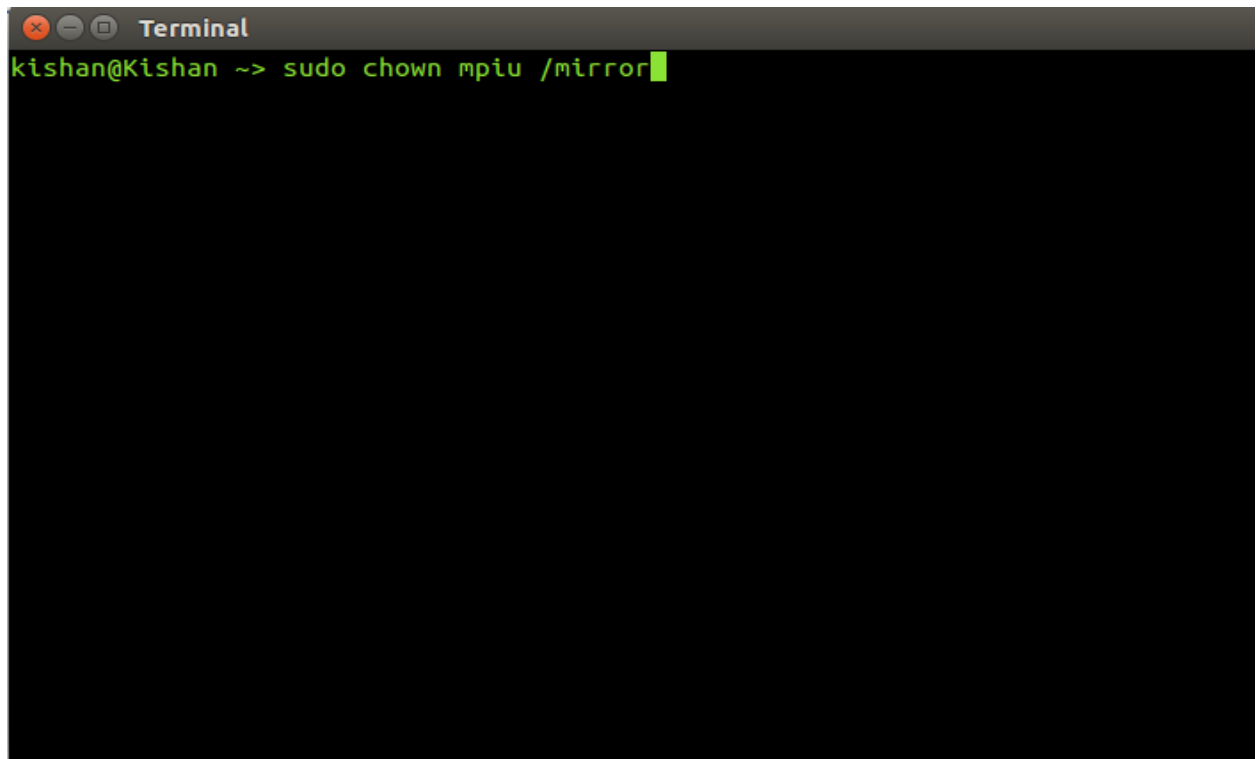


```
Terminal
kishan@Kishan ~-> sudo mkdir /mirror
```

- c) Create a user called mpiu (or anything you like) on both of the nodes whose home directory is the /mirror folder that we had created at step (b).  
sudo adduser --home /mirror mpiu

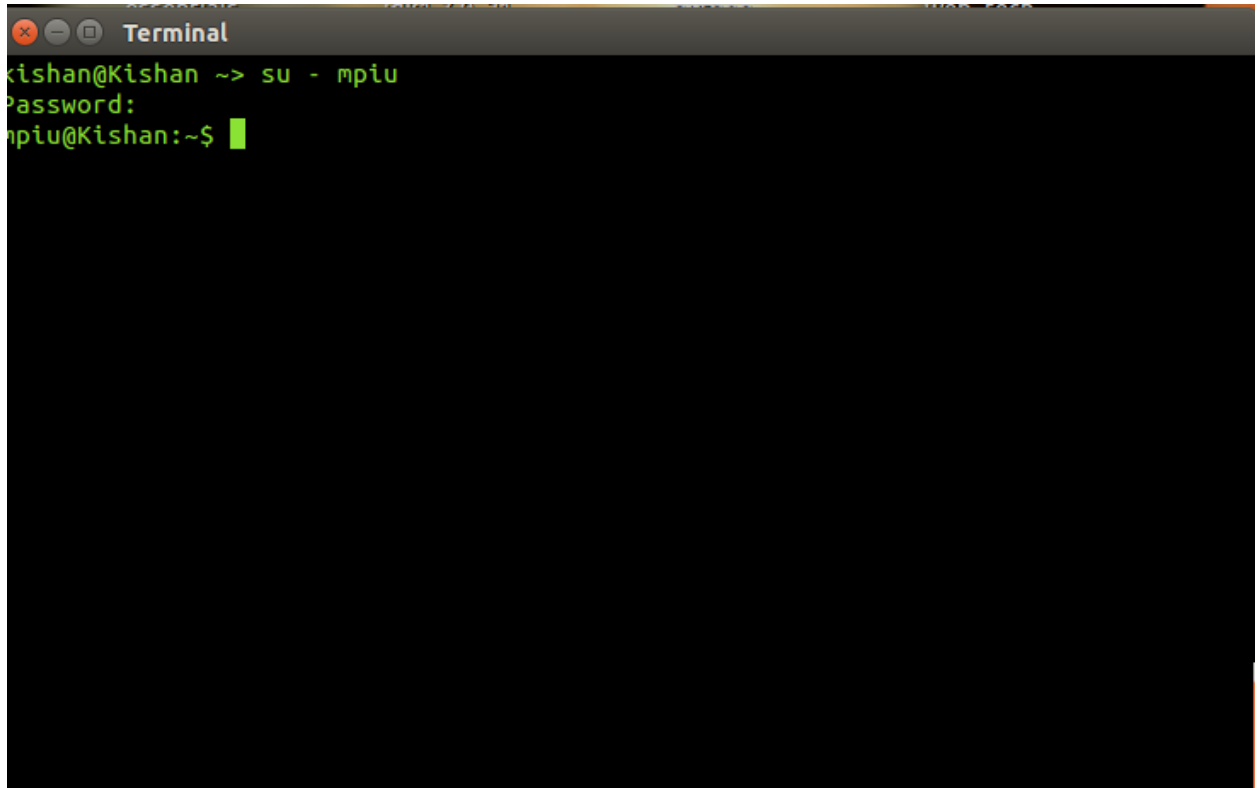
A terminal window titled "Terminal" with a dark background. The prompt is "kishan@Kishan ~>". The command "sudo adduser --home /mirror mpiu" is entered in green text, followed by a green cursor. The rest of the terminal area is empty.

- d) Now we will hand over the ownership /mirror to the folder mpiu. sudo chown mpiu /mirror.

A terminal window titled "Terminal" with a dark background. The prompt is "kishan@Kishan ~>". The command "sudo chown mpiu /mirror" is entered in green text, followed by a green cursor. The rest of the terminal area is empty.

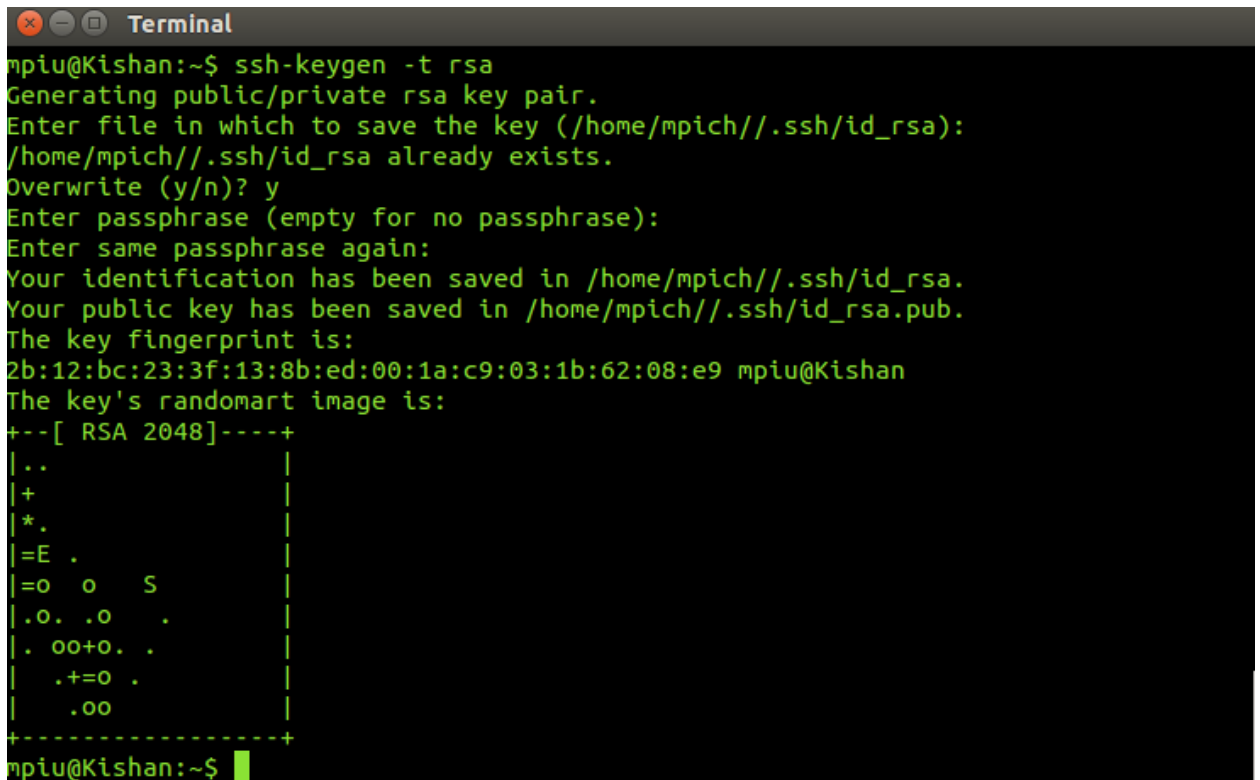
- e) Now install openssh-server. The command is: sudo apt-get install openssh-server.

- f) Log in to the new user we created on step (c) providing proper credentials. `su - mpiu`.



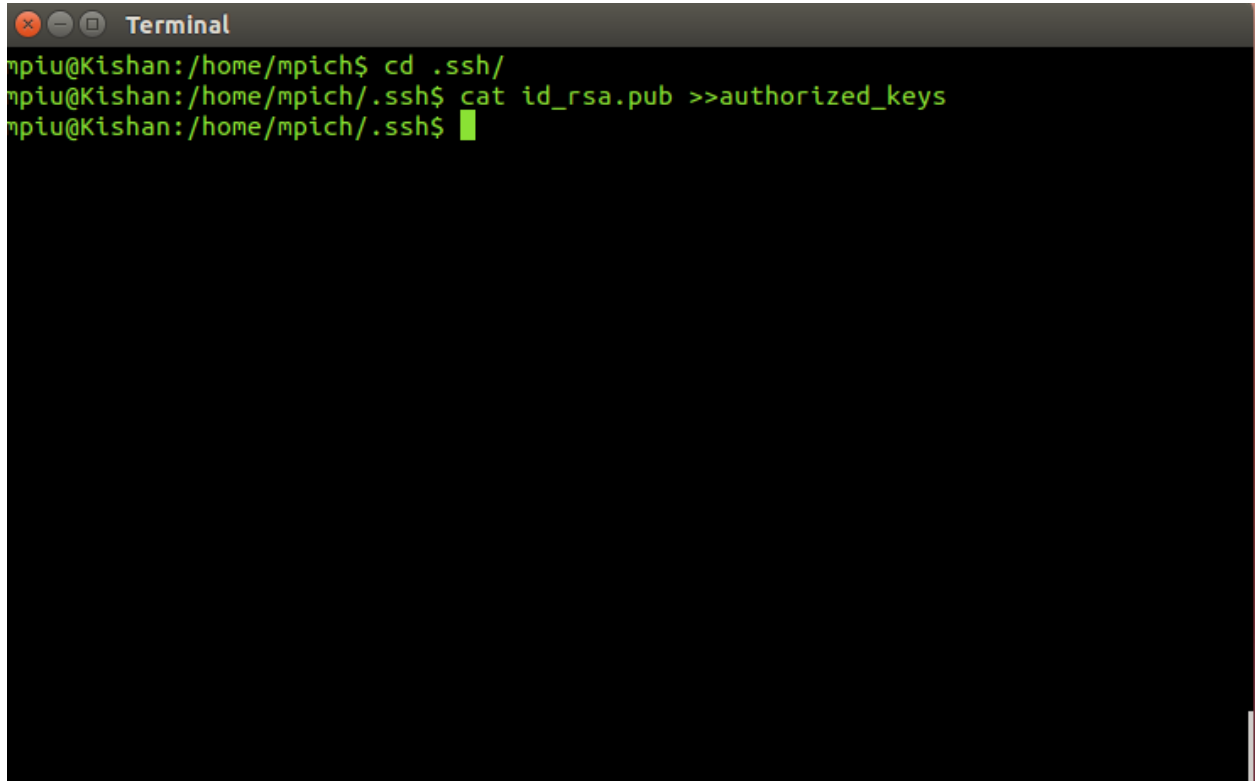
```
Terminal
kishan@Kishan ~> su - mpiu
Password:
mpiu@Kishan:~$
```

- g) Then generate a rsa key pair for mpiu. The command to be written is: `ssh-keygen -t rsa`.



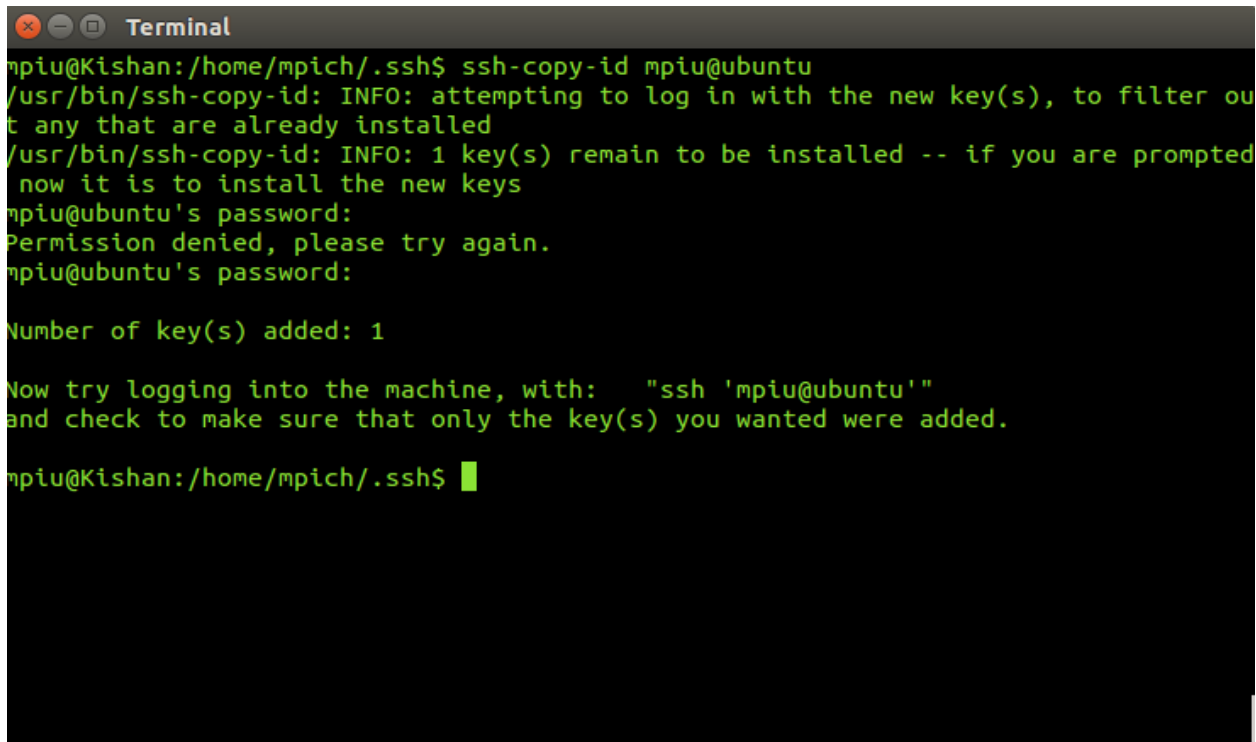
```
Terminal
mpiu@Kishan:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mpich//.ssh/id_rsa):
/home/mpich//.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mpich//.ssh/id_rsa.
Your public key has been saved in /home/mpich//.ssh/id_rsa.pub.
The key fingerprint is:
2b:12:bc:23:3f:13:8b:ed:00:1a:c9:03:1b:62:08:e9 mpiu@Kishan
The key's randomart image is:
+--[RSA 2048]-----+
|.. |
|+ |
|* |
|=E . |
|=o o S |
|.O. .O . |
|. oo+O. . |
|. +=O . |
|.OO |
+-----+
mpiu@Kishan:~$
```

- h) Now navigate to the ssh folder. The public key has been stored in id\_rsa.pub and now we need to copy the key in an authorized key file. `cat id_rsa.pub >> authorized_keys`



```
Terminal
mpiu@Kishan:/home/mpich$ cd .ssh/
mpiu@Kishan:/home/mpich/.ssh$ cat id_rsa.pub >>authorized_keys
mpiu@Kishan:/home/mpich/.ssh$
```

- i) Do the same thing on the child node and pass the key using: `ssh-copy-id <hostname>`



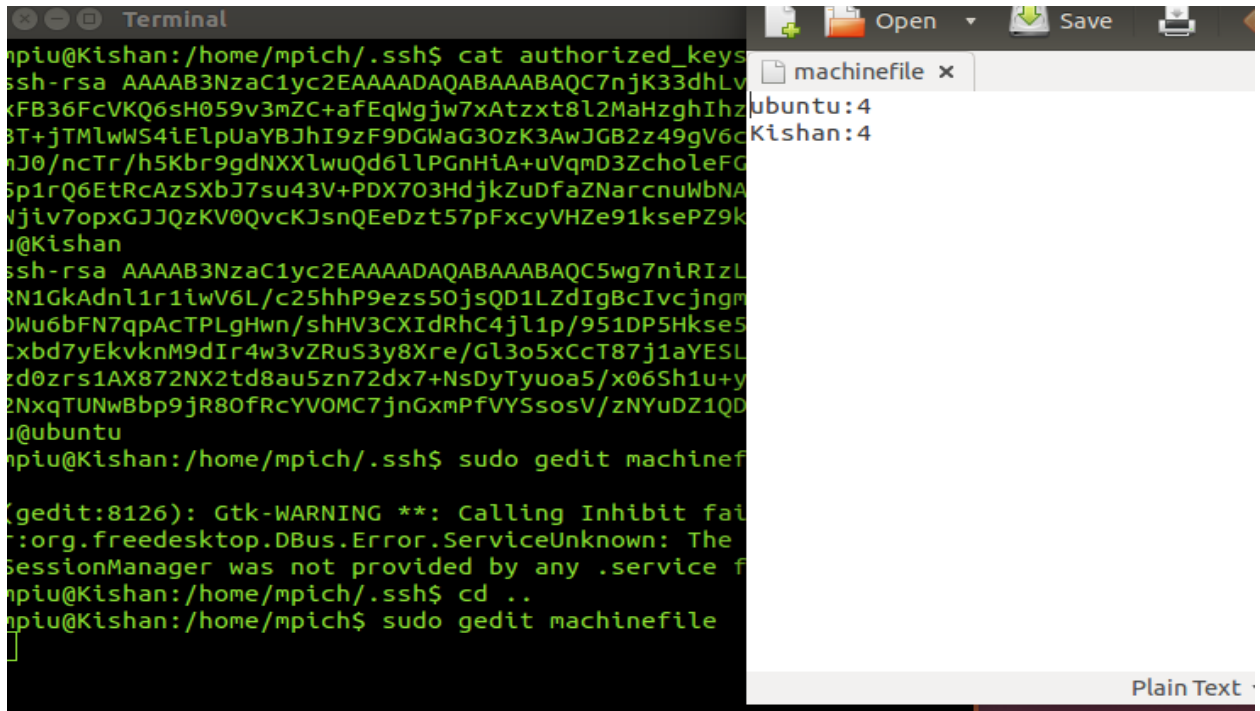
```
Terminal
mpiu@Kishan:/home/mpich/.ssh$ ssh-copy-id mpiu@ubuntu
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
now it is to install the new keys
mpiu@ubuntu's password:
Permission denied, please try again.
mpiu@ubuntu's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'mpiu@ubuntu'"
and check to make sure that only the key(s) you wanted were added.

mpiu@Kishan:/home/mpich/.ssh$
```

- j) Now try to ssh to the child node from the master `ssh mpiu@childnodename`  
If you can connect proceed to the next steps, otherwise see the troubleshooting information provided at the back of this chapter.
- k) Now set up a machine file. Create the file inside mpiu's home directory. Then edit the file.  
A machine file is a file where the nodes and the number of assigned tasks for each of the nodes are written down. Provided below is a sample edit of the machine file.



The screenshot shows a terminal window on the left and a file editor on the right. The terminal window displays the following commands and output:

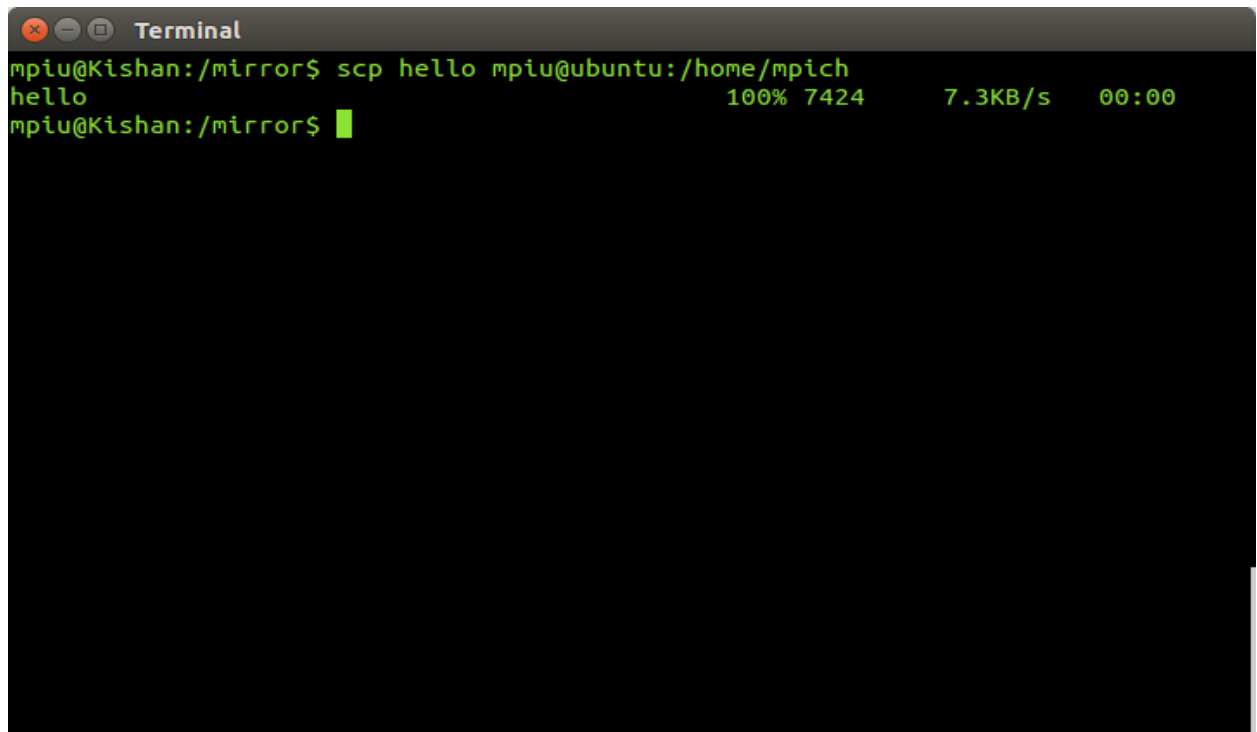
```
mpiu@Kishan:/home/mpich/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC7njK33dhLv
xkFB36FcVKQ6sH059v3mZC+afEqWgJw7xAtzxt8l2MaHzghIhz
BT+jTmLwWS4iElpUaYBJhI9zF9DGWAG30zK3AwJGB2z49gV6c
hJ0/ncTr/h5Kbr9gdNXXLwuQd6lLPgNHIA+uVqmD3ZchoLeFG
5p1rQ6EtRcAzSxbJ7su43V+PDX703HdjkZuDfaZNarcnuWbNA
Ujiv7opxGJJQzKV0QvcKJsnQEeDzt57pFxcyVHZe91ksePZ9k
u@Kishan
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC5wg7niRIzL
RN1GkAdnl1r1iwV6L/c25hhP9ezs50jsQD1LZdIgBcIvcjngm
DWu6bFN7qpAcTPLgHwn/shHV3CXIdRhC4jl1p/951DP5Hkse5
Cxbd7yEkvknM9dIr4w3vZRuS3y8Xre/GL3o5xCcT87j1aYESL
zd0zrs1AX872NX2td8au5zn72dx7+NsDyTyuoas5/x06Sh1u+y
2NxqTUNwBbp9jR80fRcYVOMC7jnGxmPfVYSsosV/zNYuDZ1QD
u@ubuntu
mpiu@Kishan:/home/mpich/.ssh$ sudo gedit machinefile
(gedit:8126): Gtk-WARNING **: Calling Inhibit fai
r:org.freedesktop.DBus.Error.ServiceUnknown: The
SessionManager was not provided by any .service f
mpiu@Kishan:/home/mpich/.ssh$ cd ..
mpiu@Kishan:/home/mpich$ sudo gedit machinefile
```

The file editor on the right shows the content of the `machinefile`:

```
ubuntu:4
Kishan:4
```

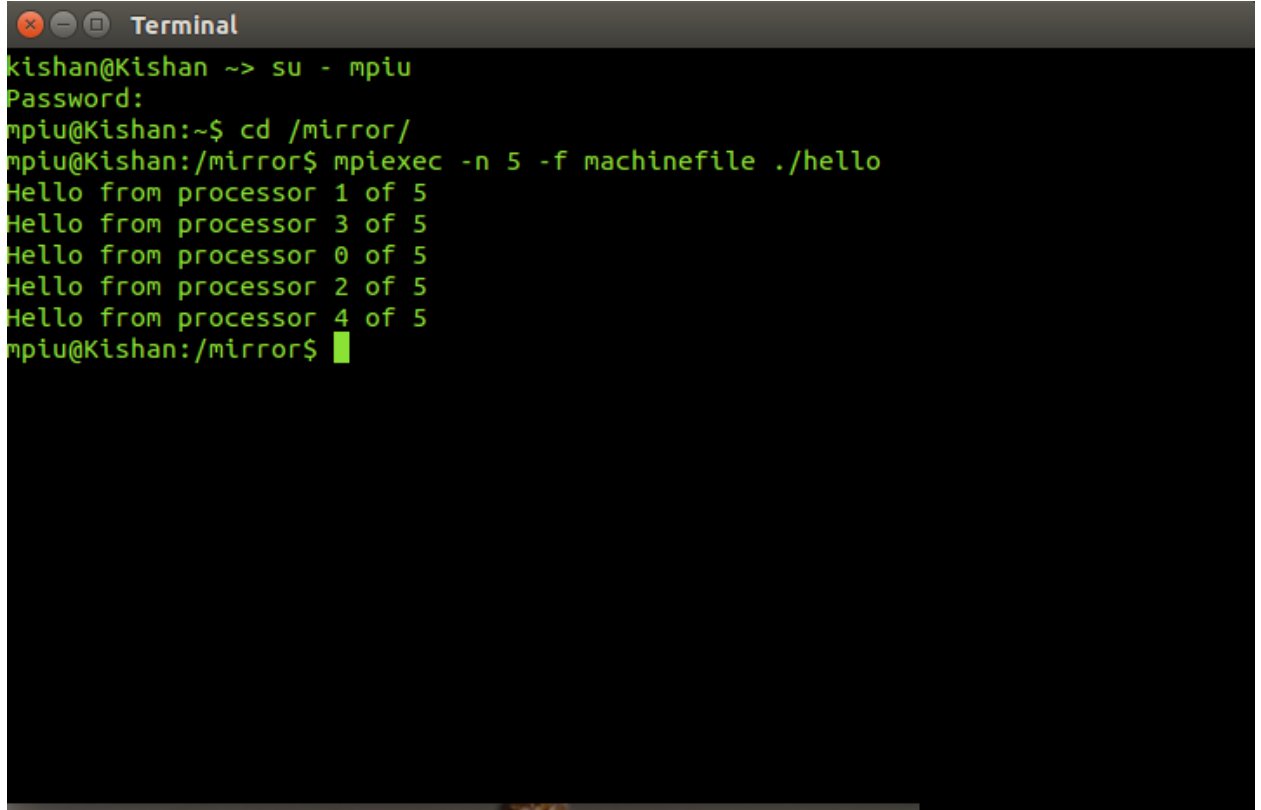
- l) Now compile the program in the master pc using mpicc, as I have mentioned in the previous chapter. Then pass the executable from the master node to the child node by

ssh, even a usb. The command is: `scp hello mpiu@hostname:/path to destination`.

A terminal window titled "Terminal" with a dark background and light green text. The prompt is "mpiu@Kishan:/mirror\$". The command entered is "scp hello mpiu@ubuntu:/home/mpich". The output shows "hello" followed by "100% 7424 7.3KB/s 00:00". The prompt returns to "mpiu@Kishan:/mirror\$".

```
Terminal
mpiu@Kishan:/mirror$ scp hello mpiu@ubuntu:/home/mpich
hello 100% 7424 7.3KB/s 00:00
mpiu@Kishan:/mirror$
```

- m) Now run mpiexec: `mpiexec -n 5 -f machinefile ./hello`. And the simple hello world program, with a little bit of help from MPI is now running on 2 different nodes!

A terminal window titled "Terminal" with a dark background and light green text. The user is at a prompt "kishan@Kishan ~>". They enter "su - mpiu". A "Password:" prompt appears. Then they enter "cd /mirror/". The prompt changes to "mpiu@Kishan:/mirror\$". They enter "mpiexec -n 5 -f machinefile ./hello". The output shows five lines: "Hello from processor 1 of 5", "Hello from processor 3 of 5", "Hello from processor 0 of 5", "Hello from processor 2 of 5", and "Hello from processor 4 of 5". The prompt returns to "mpiu@Kishan:/mirror\$".

```
kishan@Kishan ~> su - mpiu
Password:
mpiu@Kishan:~$ cd /mirror/
mpiu@Kishan:/mirror$ mpiexec -n 5 -f machinefile ./hello
Hello from processor 1 of 5
Hello from processor 3 of 5
Hello from processor 0 of 5
Hello from processor 2 of 5
Hello from processor 4 of 5
mpiu@Kishan:/mirror$
```

## 3.2 Conclusion

In this chapter we have learnt to use MPICH to make a cluster. Next we extend our ideas to communication of nodes.

### 3.3 Troubleshooting Guide

**Problem:** unrecognized argument pmi\_args.

**Solution:**

- a. Check if you are writing the commands correctly.
- b. Are master and slaves both using the same version of MPICH? Sometimes you may also need to check if the Linux kernel version is same for both.
- c. Is your code correct? You need to check for syntax issues.
- d. Was your installation process correct? Reinstall if any doubt.

**Problem:** You don't have the permission.

**Solution:**

- a. Did you change ownership of the folder to mpiu user?
- b. Give the mirror folder read write permission: `sudo chmod -R ugo+rw /mirror`.
- c. Check if all the permission settings in your Linux distribution is working properly.

**Problem:** Chown Operation not permitted.

**Solution:**

- a. Chown can be performed by root only, so you either do not have root access or you are not using sudo. Use sudo before the chown command

**Problem:** User not in the sudoers group.

**Solution:**

- a. Add User to the sudoers group. `sudo usermod -aG sudo [username]`

**Problem:** No route to host.

**Solution:**

- a. Edit your `/etc/hosts` file and add ip address and name of the child nodes.

**Problem:** No route to host on port x

**Solution:**

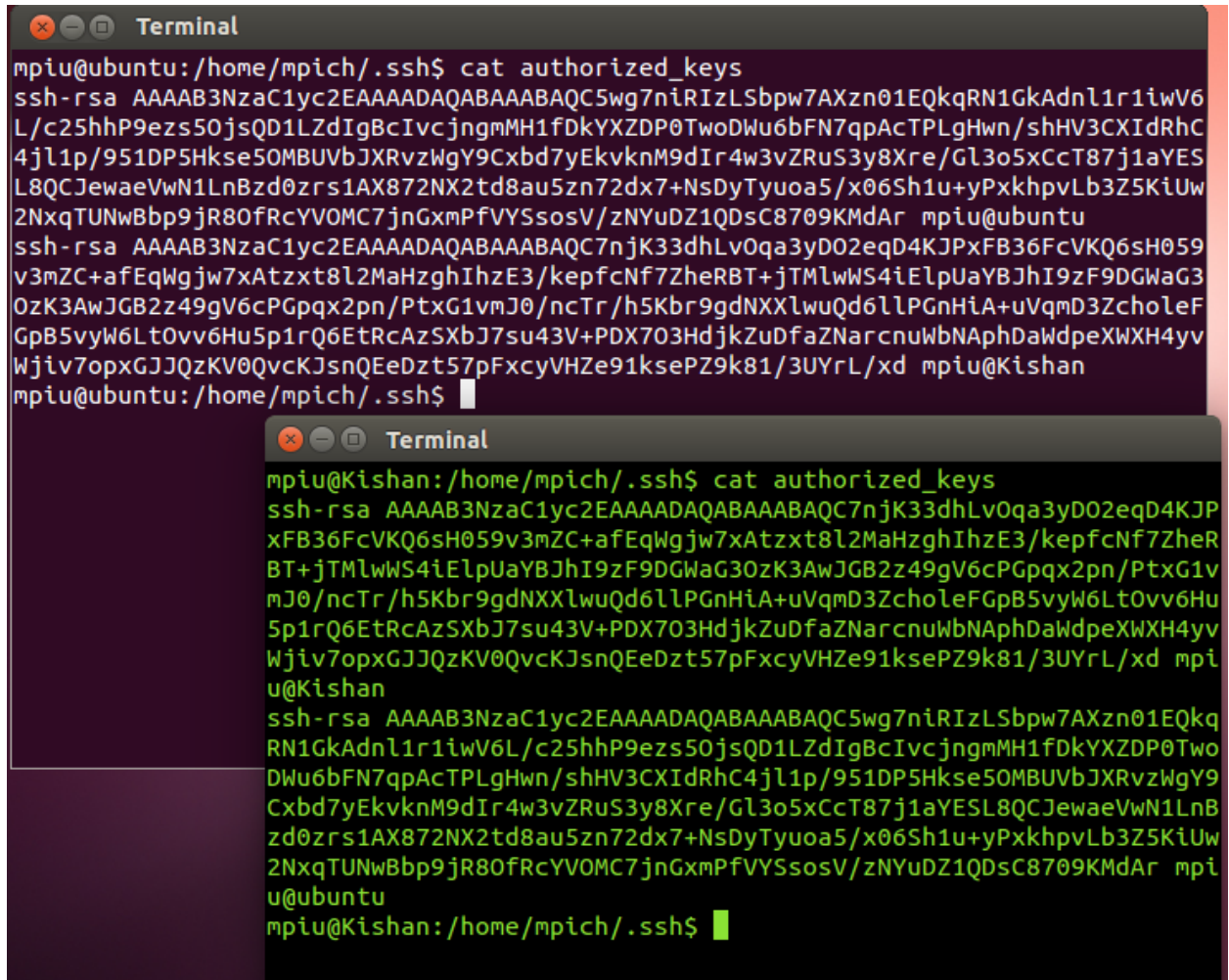
- a. See if your port is blocked by firewall.
- b. See if you have running processes on the port. If it is the case let the mpich use another port with the `-p [port no]` option.



**Problem:** Cannot connect through ssh.

**Solution:**

- a. See if the passed keys from master and child matches. Here is a sample ideal case:



```
Terminal
mpiu@ubuntu:/home/mpich/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC5wg7nIRIZLSbpw7AXzn01EQkqRN1GkAdnl1r1iwV6
L/c25hhP9ezs50jsQD1LZdIgBcIvcjngmMH1fDkYXZDP0TwoDWu6bFN7qpAcTPlgHwn/shHV3CXIdRhC
4jl1p/951DP5Hkse50MBUVbJXRvzWgY9Cxbd7yEkvknM9dIr4w3vZRU53y8Xre/GL3o5xCcT87j1aYES
L8QCJewaeVwN1LnBzd0zrs1AX872NX2td8au5zn72dx7+NsdYTyuoas5/x06Sh1u+yPxkhpvLb3Z5KiUw
2NxqTUNwBbp9jR80fRcYVOMC7jnGxmPfVYSsosV/zNYuDZ1QDsC8709KMdAr mpiu@ubuntu
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC7njK33dhLv0qa3yD02eqD4KJPxFB36FcVKQ6sH059
v3mZC+afEqWgJw7xAtzxt8l2MaHzghIhzE3/kepfCnf7ZheRBT+jTmlwWS4iElpUaYBJhI9zF9DgWag3
OzK3AwJGB2z49gV6cPGpqx2pn/PtxG1vmJ0/ncTr/h5Kbr9gdNXXlwQd6lLPgnHiA+uVqmD3ZcholeF
GpB5vyW6Lt0vv6Hu5p1rQ6EtRcAzSxbJ7su43V+PDX703HdjKZuDfaZNarcnuWbNaphDaWdpeXWXH4yv
Wjiv7opxGJJQzKV0QvcKJsnQEEdzt57pFxcyVHZe91ksePZ9k81/3UYrL/xd mpiu@Kishan
mpiu@ubuntu:/home/mpich/.ssh$

Terminal
mpiu@Kishan:/home/mpich/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC7njK33dhLv0qa3yD02eqD4KJP
xFB36FcVKQ6sH059v3mZC+afEqWgJw7xAtzxt8l2MaHzghIhzE3/kepfCnf7ZheR
BT+jTmlwWS4iElpUaYBJhI9zF9DgWag3OzK3AwJGB2z49gV6cPGpqx2pn/PtxG1v
mJ0/ncTr/h5Kbr9gdNXXlwQd6lLPgnHiA+uVqmD3ZcholeFGpB5vyW6Lt0vv6Hu
5p1rQ6EtRcAzSxbJ7su43V+PDX703HdjKZuDfaZNarcnuWbNaphDaWdpeXWXH4yv
Wjiv7opxGJJQzKV0QvcKJsnQEEdzt57pFxcyVHZe91ksePZ9k81/3UYrL/xd mpi
u@Kishan
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC5wg7nIRIZLSbpw7AXzn01EQkq
RN1GkAdnl1r1iwV6L/c25hhP9ezs50jsQD1LZdIgBcIvcjngmMH1fDkYXZDP0Two
DWu6bFN7qpAcTPlgHwn/shHV3CXIdRhC4jl1p/951DP5Hkse50MBUVbJXRvzWgY9
Cxbd7yEkvknM9dIr4w3vZRU53y8Xre/GL3o5xCcT87j1aYESL8QCJewaeVwN1LnB
zd0zrs1AX872NX2td8au5zn72dx7+NsdYTyuoas5/x06Sh1u+yPxkhpvLb3Z5KiUw
2NxqTUNwBbp9jR80fRcYVOMC7jnGxmPfVYSsosV/zNYuDZ1QDsC8709KMdAr mpi
u@ubuntu
mpiu@Kishan:/home/mpich/.ssh$
```

- b. Examine if you have passed the keys at all. Also have you copied the keys from id\_rsa.pub to authorized\_keys. Check it by issuing cat authorized\_keys.

## Chapter 4

### Send and Receive

In this chapter we will learn how to use one of the most useful features of MPICH: MPI\_Send and MPI\_Recv. They operate exactly in the same manner as described in chapter 1. We now look at their definition and implementation issues.

#### 4.1 Definition

MPI\_Send( void\* data, int count, MPI\_Datatype datatype, int destination, int tag, MPI\_Comm communicator)

MPI\_Recv( void\* data, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm communicator, MPI\_Status\* status)

The first parameter is the pointer to the data buffer. Second argument is the number of data in the data buffer. And the third argument is the datatype of the data in the data buffer. Important data types include:

- a. MPI\_SHORT
- b. MPI\_INT
- c. MPI\_LONG
- d. MPI\_FLOAT
- e. MPI\_DOUBLE

The destination and source is the rank of the senders and the receivers. Tag is a tag in the message for identification. Then comes the communicator and the last one which is specific for MPI\_Recv informs us of the status of a received message.

## 4.2 Implementation

Now we will distribute some numbers among some processors and use the mentioned functions for adding all of them up. We will first observe the program by parts:

```
#include <stdio.h>
#include <mpi.h>

int array[100000];
int array2[100000];

main(int argc, char **argv)
{
 long int sum, partial;
 MPI_Status status;
 int process_id, r_process, init, i, n_rows, n_proc,
 j, receive_rows, rows_p_p,
 sender, received_rows, start, end, send_rows;

 init = MPI_Init(&argc, &argv);

 r_process = 0;

 init = MPI_Comm_rank(MPI_COMM_WORLD, &process_id);
 init = MPI_Comm_size(MPI_COMM_WORLD, &n_proc);
```

Here some variables have been declared and we have initialized as we have seen before. Just this time we pass the arguments from the main function into the MPI\_init. Now we find the rank of the processor and size of the communicator.

```

if(process_id == r_process) {
 printf("please enter the number of numbers to sum: ");
 scanf("%i", &n_rows);
 rows_p_p = n_rows / n_proc;

 for(i = 0; i < n_rows; i++) {
 array[i] = i + 1;
 printf("%d\n",i+1);
 }

 for(j = 1; j < n_proc; j++) {
 start = j*rows_p_p + 1;
 end = (j + 1)*rows_p_p;

 if((n_rows - end) < rows_p_p)
 end = n_rows - 1;

 send_rows = end - start + 1;

 init = MPI_Send(&send_rows, 1 , MPI_INT,
 j, 2001, MPI_COMM_WORLD);

 init = MPI_Send(&array[start], send_rows, MPI_INT,
 j, 2001, MPI_COMM_WORLD);
 }
 sum = 0;
 for(i = 0; i < rows_p_p + 1; i++) {
 sum += array[i];
 }

 printf("sum %i calculated by root process\n", sum);
}

```

Here we find that we have the root process and divide the array into `n_proc` number of sub arrays that we distribute or send to each of the machine using `MPI_Send`. First it sends the sub array siz. Then it sends the sub array itself. Here the data buffer is the subarray `array[start]`. we are sending `send_rows` of rows so count is `send_rows`, datatype is `MPI_INT`, rank is the loop index, and tag is set to 2001 for identification and the communicator is the default one. Then the master node calculates its portion.

```

for(j = 1; j < n_proc; j++) {
 init = MPI_Recv(&partial, 1, MPI_LONG, MPI_ANY_SOURCE,
 2002, MPI_COMM_WORLD, &status);

 sender = status.MPI_SOURCE;

 printf("Partial sum %i returned from process %i\n", partial, sender);

 sum += partial;
}

printf("The grand total is: %i\n", sum);
}

```

Here the root process collects all partial sums from the child processes. We will come to the fact how the child processes send them. For now, assume that root processes have received the partial sums from child processes with data buffer named partial. Also note the argument MPI\_ANY\_SOURCE, it means that the source of the data sent may be any one of the processors. 2002 is the receive tag. We get the sender's rank by status.MPI\_SOURCE. Then we sum up all the partial sums and show the result.

```

else {

 init = MPI_Recv(&receive_rows, 1, MPI_INT,
 r_process, 2001, MPI_COMM_WORLD, &status);

 init = MPI_Recv(&array2, receive_rows, MPI_INT,
 r_process, 2001, MPI_COMM_WORLD, &status);

 received_rows = receive_rows;

 partial = 0;
 for(i = 0; i < received_rows; i++) {
 partial += array2[i];
 }

 init = MPI_Send(&partial, 1, MPI_LONG, r_process,
 2002, MPI_COMM_WORLD);
}

init = MPI_Finalize();
}

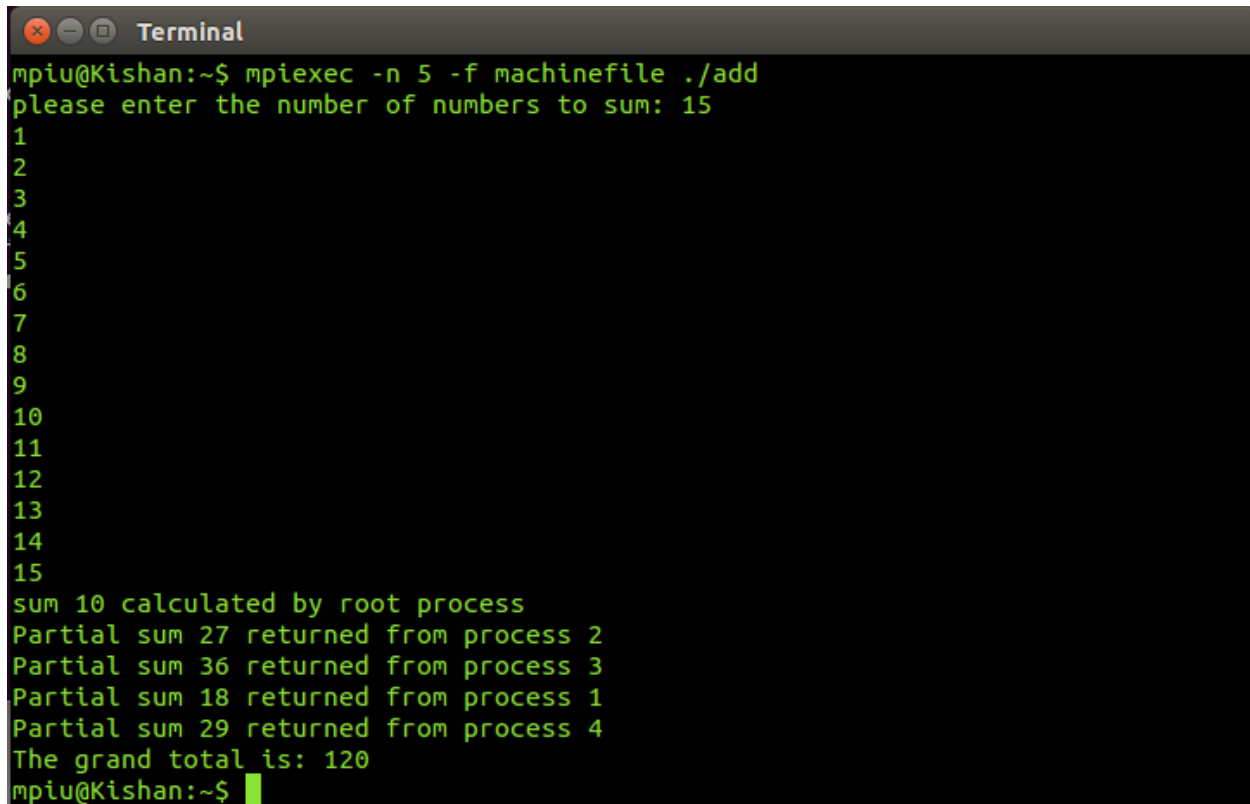
```

Now if a process is a slave one, then it receives the sub array passed by the root process. It matches the sender address which is 0 and tag to identify the exact message sent to it. First it receives the sub array

size, then it receives the sub array. Then it calculates the sum of the sub array elements. After that, it sends the result to the master identified by `r_process` and sent message tag identified by 2002.

Then we call `MPI_Finalize()` to terminate MPI execution environment.

Here's a sample run from our pc.



```
Terminal
mpiu@Kishan:~$ mpiexec -n 5 -f machinefile ./add
please enter the number of numbers to sum: 15
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
sum 10 calculated by root process
Partial sum 27 returned from process 2
Partial sum 36 returned from process 3
Partial sum 18 returned from process 1
Partial sum 29 returned from process 4
The grand total is: 120
mpiu@Kishan:~$
```

### 4.3 Conclusion

In this chapter we have gone through the implementation and uses of the mpi command `MPI_Send` and `MPI_Receive`. Thus we have almost covered most of the very basic useful features of MPICH.

## 4.4 Troubleshooting Guide

In this chapter, if you get the things right at the previous chapters, you should not run into any problem. However while running the code, you may face an issue with warnings. But if you execute command `mpicc -w -o add add.c`, then the compiler will ignore the warnings and generate executable.

## References:

- a. <https://help.ubuntu.com/community/MpichCluster>
- b. <http://mpitutorial.com/beginner-mpi-tutorial/>
- c. <https://wiki.ubuntu.com/MpichCluster>