



**VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

**CSE4001 Parallel and Distributed Computing**

**Digital Assignment-4 (ELA)**

**Name: Nakul Jadeja**

**Reg no.: 19BCE0660**

**School of Computer Science and Engineering**

**Course Code: CSE4001**

**Slot: L11+L12**

**Winter Semester 2021-22**

**Professor: Narayanamoorthi M**

## **Question 1) Compare and contrast open MP and MPI**

**Answer: -**

**OpenMP** is a way to program on shared memory devices. This means that the parallelism occurs where every parallel thread has access to all of your data.

Example: -You can think of it as: parallelism can happen during execution of a specific for loop by splitting up the loop among the different threads.

**MPI** is a way to program on distributed memory devices. This means that the parallelism occurs where every parallel process is working in its own memory space in isolation from the others.

Example: -You can think of it as: every bit of code you've written is executed independently by every process. The parallelism occurs because you tell each process exactly which part of the global problem, they should be working on based entirely on their process ID.

**MPI** stands for Message Passing Interface. It is a set of API declarations on message passing (such as send, receive, broadcast, etc.), and what behaviour should be expected from the implementations.

The idea of "message passing" is rather abstract. It could mean passing message between local processes or processes distributed across networked hosts, etc. Modern implementations try very hard to be versatile and abstract away the multiple underlying mechanisms (shared memory access, network IO, etc.).

**OpenMP** is an API which is all about making it (presumably) easier to write shared-memory multi-processing programs. There is no notion of passing messages around. Instead, with a set of standard functions and compiler directives, you write programs that execute local threads in parallel, and you control the behavior of those threads (what resource they should have access to, how they are synchronized, etc.). OpenMP requires the support of the compiler, so you can also look at it as an extension of the supported languages.

<b>MPI</b>	<b>OpenMP</b>
1. Available from different vendor and can be compiled in desired platform with desired compiler. One can use any of MPI API i.e MPICH, OpenMPI or other	1 .OpenMP are hooked with compiler so with gnu compiler and with Intel compiler one have specific implementation. User is at liberty with changing compiler but not with openmp implementation.
2. MPI support C,C++ and FORTRAN	2.OpenMP support C,C++ and FORTRAN
3. OpenMPI one of API for MPI is providing provisional support for Java	3.Few projects try to replicate openmp for Java.
4. MPI target both distributed as well shared memory system	4.OpenMP target only shared memory system
5. Based on both process and thread based approach .(Earlier it was mainly process based parallelism but now with MPI 2 and 3 thread based parallelism is there too. Usually a process can contain more than 1 thread and call MPI subroutine as desired	5.Only thread based parallelism.
6. Overhead for creating process is one time	6. Depending on implementation threads can be created and joined for particular task which add overhead
7. There are overheads associated with transferring message from one process to another	7.No such overheads, as thread can share variables
8. Process in MPI has private variable only, no shared variable	8. In OpenMP , threads have both private as well shared variable

9. Data racing is not there if not using any thread in process .	9. Data racing is inherent in OpenMP model
10. Compilation of MPI program require 1. Adding header file : #include "mpi.h" 2. compiler as:(in linux ) mpic++ mpi.cxx -o mpiExe(User need to set environment variable PATH and LD_LIBRARY_PATH to MPI as OpenMPI installed folder or binaries) (For Linux)	10. Need to add omp.h and then can directly compile code with -fopenmp in Linux environment g++ -fopenmp openmp.cxx -o openmpExe
11. Running MPI program . a ) User need to make sure that bin and library folder from MPI installation are included in environmental variable PATH and LD_LIBRARY_PATH. b) For running executable from command line ,user need to supply following command and specify number of processor as in example below it is four . <b>mpirun -np 4 mpiExe</b>	11. User can launch executable openmpExe in normal way <b>./openmpExe</b>

**Question 2) Write a c program using MPI to perform the following: Print hello world with process id Perform arithmetic operations using process id**

**Answer: -**

**Code for printing hello world: -**

```
#include <stdio.h>

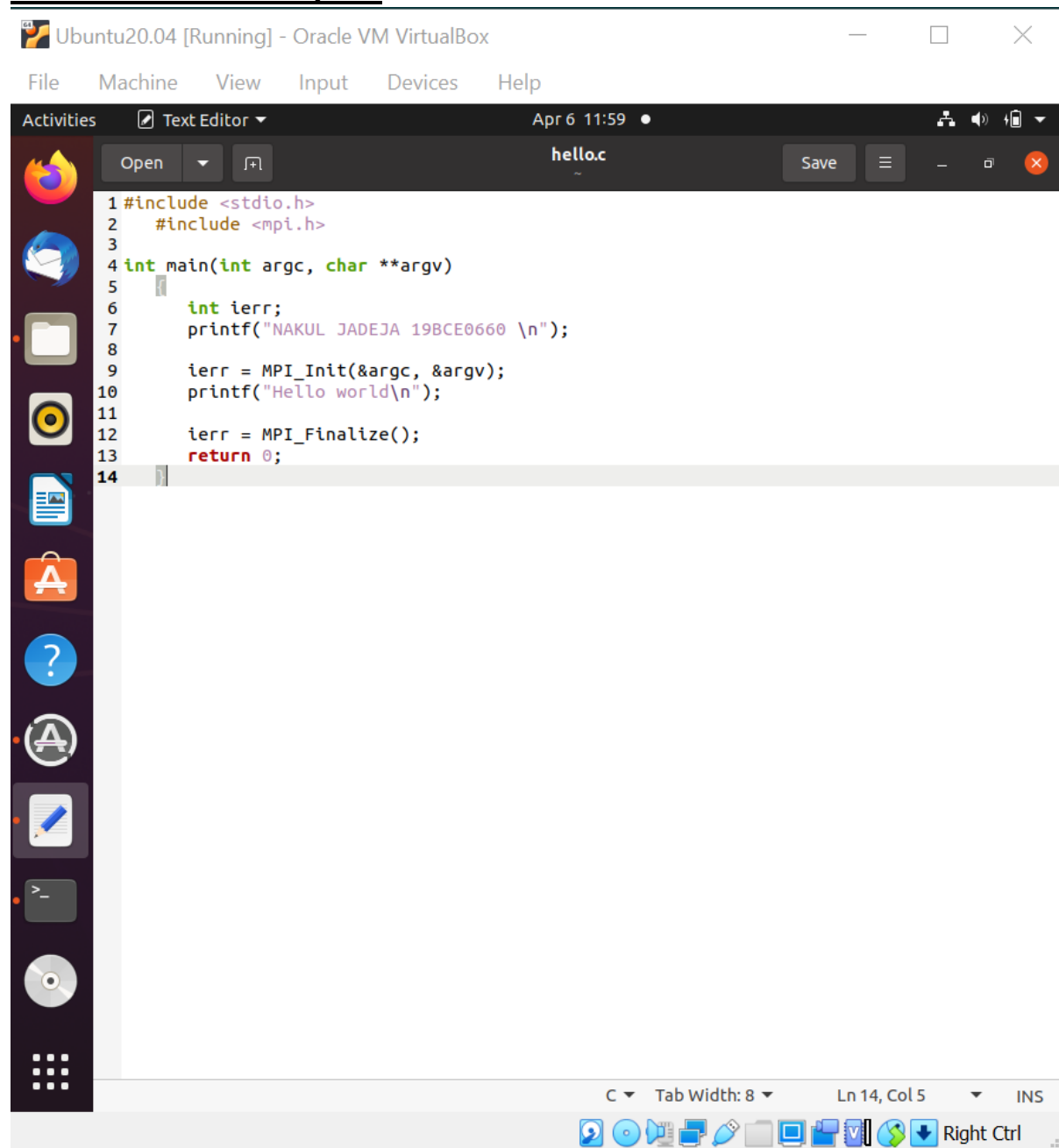
#include <mpi.h>
```

```
int main(int argc, char **argv)
{
    int ierr;
    printf("NAKUL JADEJA 19BCE0660 \n");

    ierr = MPI_Init(&argc, &argv);
    printf("Hello world\n");

    ierr = MPI_Finalize();
    return 0;
}
```

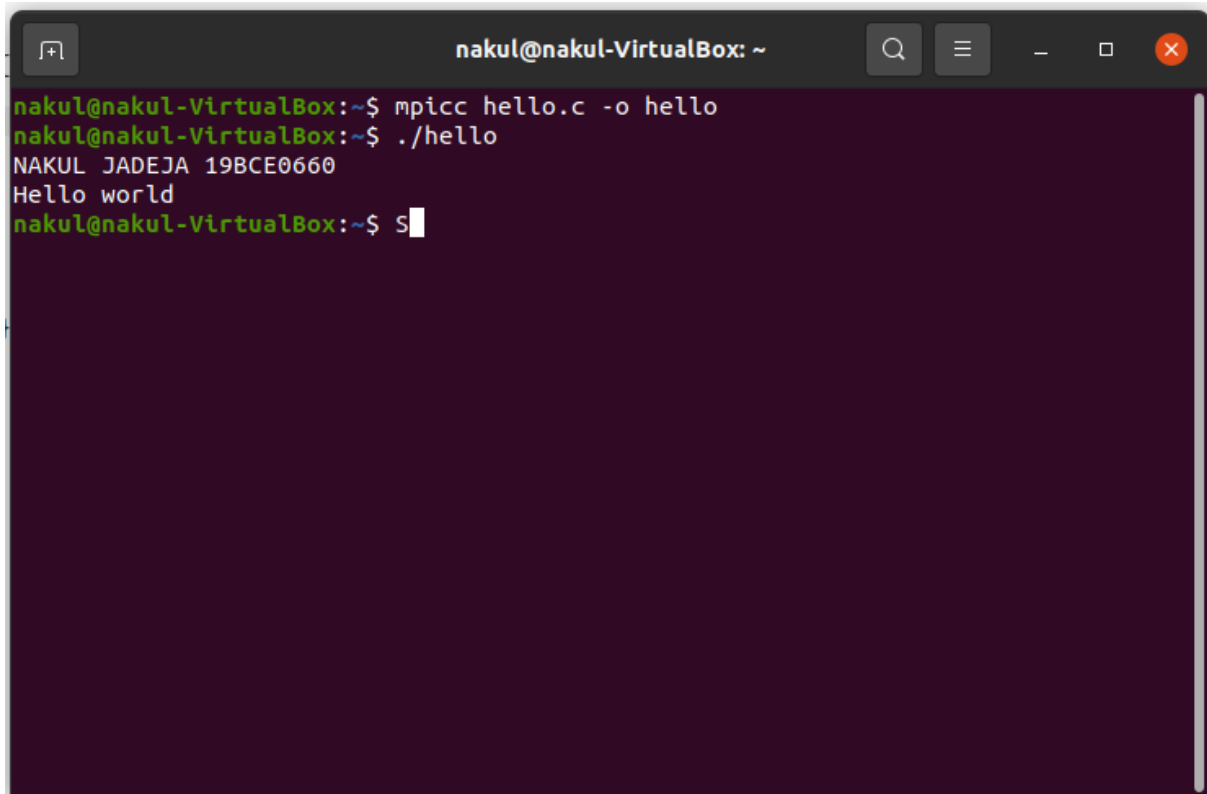
## Screenshot of the Output:-



The screenshot shows a VirtualBox window titled "Ubuntu20.04 [Running] - Oracle VM VirtualBox". Inside the VM, the Ubuntu desktop environment is visible. The "Text Editor" application is open, displaying a C program named "hello.c". The code is as follows:

```
1 #include <stdio.h>
2 #include <mpi.h>
3
4 int main(int argc, char **argv)
5 {
6     int ierr;
7     printf("NAKUL JADEJA 19BCE0660 \n");
8
9     ierr = MPI_Init(&argc, &argv);
10    printf("Hello world\n");
11
12    ierr = MPI_Finalize();
13    return 0;
14 }
```

The status bar at the bottom of the Text Editor window indicates "C", "Tab Width: 8", "Ln 14, Col 5", and "INS". The system tray at the bottom of the VM window shows various icons, including network, volume, and power, along with the text "Right Ctrl".

A terminal window titled 'nakul@nakul-VirtualBox: ~' with search, menu, and window control icons. It shows the following commands and output:

```
nakul@nakul-VirtualBox:~$ mpicc hello.c -o hello
nakul@nakul-VirtualBox:~$ ./hello
NAKUL JADEJA 19BCE0660
Hello world
nakul@nakul-VirtualBox:~$ S
```

**Code to perform arithmetic operation: -**

```
#include<stdio.h>
#include<mpi.h>
int main( int arg , char **args)
{
printf("19BCE0660 NAKUL JADEJA \n");
int size,rankid;
int a=426,b=9;
MPI_Init(NULL,NULL);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rankid);
if(rankid==0)
{
printf("\nHello world!! (printed by rankid %d)\n",rankid);
}
```

```
if(rankid==1)
{
printf("\nSum of %d and %d is %d (printed by rankid %d)\n",a,b,a+b,rankid);
}
if(rankid==2)
{
printf("\nThe subtration of %d and %d is %d (printed by rankid %d)\n",a,b,a-
b,rankid);
}
if(rankid==3)
{
float div=(double)a/b;
printf("\nThe multiplication of %d and %d is %d (printed by rankid
%d)\n",a,b,a*b,rankid);
printf("\nThe division of %d and %d is %f (printed by rankid
%d)\n",a,b,div,rankid);
}
MPI_Finalize();
}
```

**Screenshot of the Output: -**



Ubuntu20.04 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

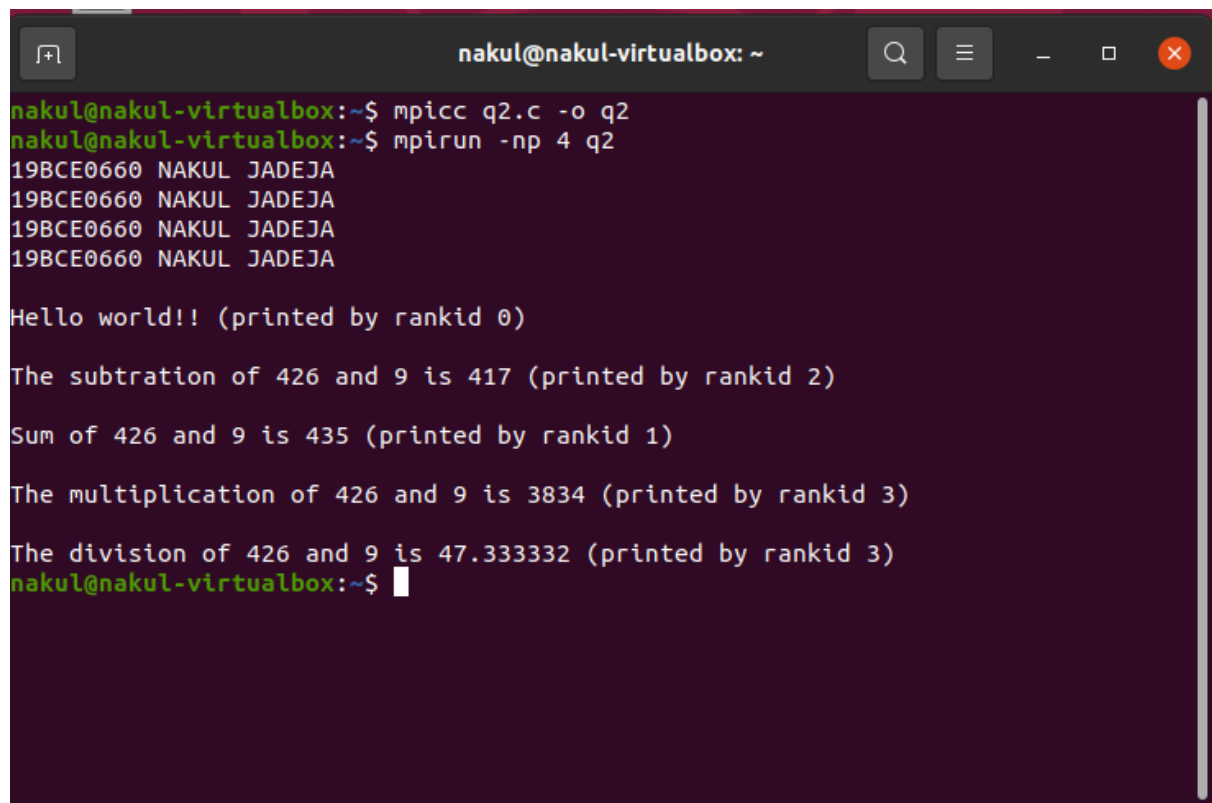
Activities Text Editor Apr 10 21:56

Open q2.c Save

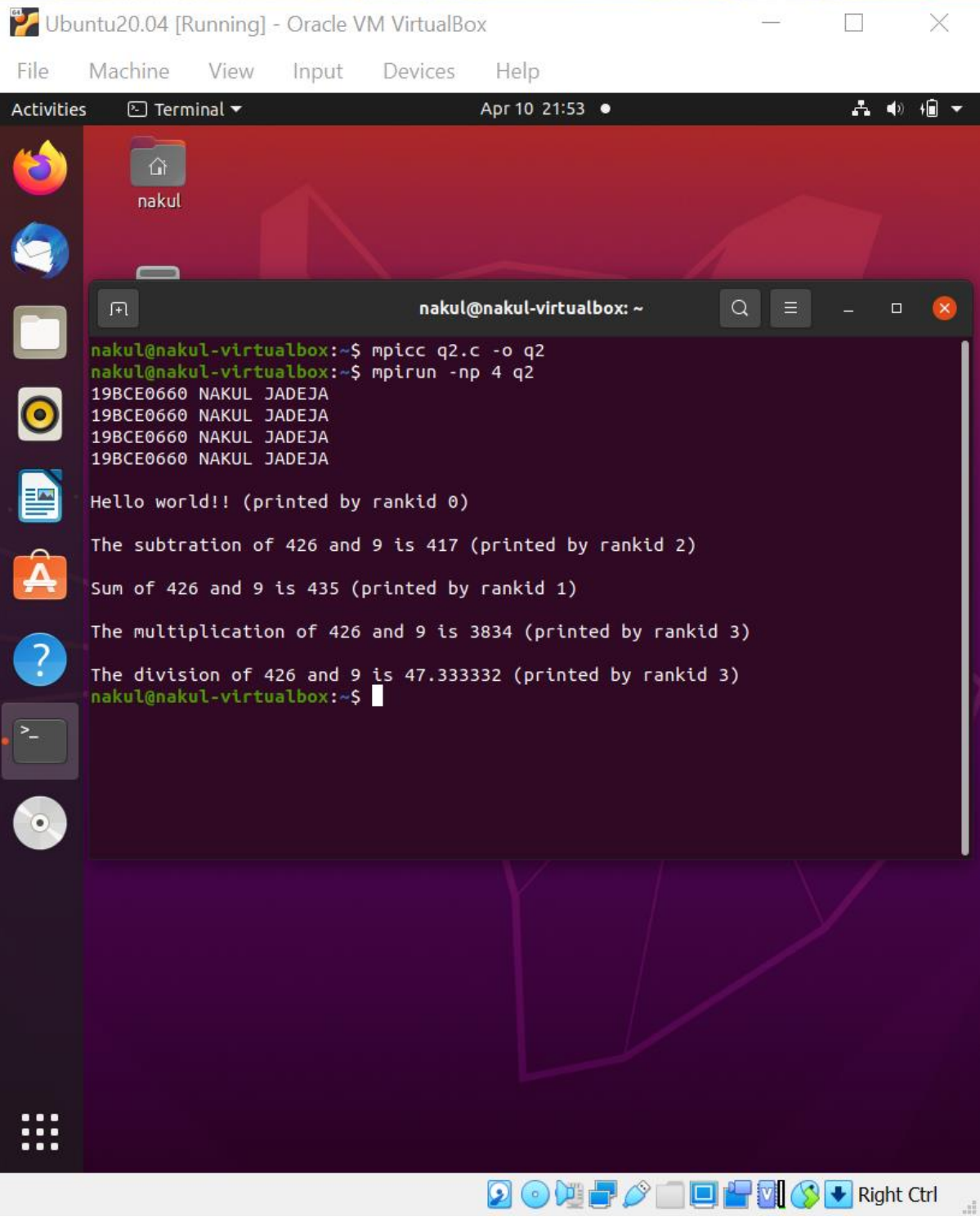
```
1 #include<stdio.h>
2 #include<mpi.h>
3 int main( int arg , char **args)
4 {
5     printf("19BCE0660 NAKUL JADEJA \n");
6     int size,rankid;
7     int a=426,b=9;
8     MPI_Init(NULL,NULL);
9     MPI_Comm_size(MPI_COMM_WORLD,&size);
10    MPI_Comm_rank(MPI_COMM_WORLD,&rankid);
11    if(rankid==0)
12    {
13        printf("\nHello world!! (printed by rankid %d)\n",rankid);
14    }
15    if(rankid==1)
16    {
17        printf("\nSum of %d and %d is %d (printed by rankid %d)\n",a,b,a+b,rankid);
18    }
19    if(rankid==2)
20    {
21        printf("\nThe subtration of %d and %d is %d (printed by rankid %d)\n",a,b,a-
22            b,rankid);
23    }
24    if(rankid==3)
25    {
26        float div=(double)a/b;
27        printf("\nThe multiplication of %d and %d is %d (printed by rankid %d)
28            \n",a,b,a*b,rankid);
29        printf("\nThe division of %d and %d is %f (printed by rankid %d)
30            \n",a,b,div,rankid);
31    }
32    MPI_Finalize();
33 }
```

C Tab Width: 8 Ln 30, Col 2 INS

Right Ctrl



```
nakul@nakul-virtualbox: ~  
nakul@nakul-virtualbox:~$ mpicc q2.c -o q2  
nakul@nakul-virtualbox:~$ mpirun -np 4 q2  
19BCE0660 NAKUL JADEJA  
19BCE0660 NAKUL JADEJA  
19BCE0660 NAKUL JADEJA  
19BCE0660 NAKUL JADEJA  
  
Hello world!! (printed by rankid 0)  
  
The subtration of 426 and 9 is 417 (printed by rankid 2)  
  
Sum of 426 and 9 is 435 (printed by rankid 1)  
  
The multiplication of 426 and 9 is 3834 (printed by rankid 3)  
  
The division of 426 and 9 is 47.333332 (printed by rankid 3)  
nakul@nakul-virtualbox:~$
```



### Question 3) Estimate the value of PI using OpenMP or MPI

#### Algorithm: -

Monte Carlo Simulation for Pi: -

Here the idea is to use parallel computing using OpenMp to solve the problem. Follow the steps below to solve the problem:

- Initialize 3 variables say x, y, and d to store the X and Y co-ordinates of a random point and the square of the distance of the random point from origin.
- Initialize 2 variables say pCircle and pSquare with values 0 to store the points lying inside circle of radius 0.5 and square of side length 1.
- Now starts the parallel processing with OpenMp together with reduction() of the following section:
- Iterate over the range [0, N] and find x and y in each iteration using srand48() and drand48() then find the square of distance of point (x, y) from origin and then if the distance is less than or equal to 1 then increment pCircle by 1.
- In each iteration of the above step, increment the count of pSquare by 1.
- Finally, after the above step calculate the value of estimated pi as below and then print the obtained value.
- $Pi = 4.0 * ((double)pCircle / (double)(pSquare))$

#### Code: -

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

```
int main (int argc, char* argv[]) {
    printf("19BCE0660 NAKUL JADEJA \n");

    int pid, numberOfNodes, error, i, count = 0, sum = 0, iterations =
1000000000;

    double pi = 0.0, begin = 0.0, end = 0.0, x, y, z;
```

```
error = MPI_Init(&argc, &argv);
```

```
/*
```

1. Get process IDS
2. Get processes number
3. Synchronize all processes
4. Get begin time

```
*/
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &numberOfNodes);
```

```
MPI_Barrier(MPI_COMM_WORLD);
```

```
begin = MPI_Wtime();
```

```
srand((int)time(0));
```

```
/* Each process caculates a part of the sum/result */
```

```
for (i = pid; i < iterations; i += numberOfNodes) {
```

```
    x = rand() / (RAND_MAX + 1.0);
```

```
    y = rand() / (RAND_MAX + 1.0);
```

```
    z = ((x*x)+(y*y));
```

```
    if(z <= 1.0)
```

```
    {
```

```
        count++;
```

```
    }
```

```
}
```

```

/*
    1. Sum all the results
    2. Synchronize all processes
    3. Get end time
*/

MPI_Reduce(&count, &sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);

end = MPI_Wtime();

if (pid == 0)
{
    pi = 4 * 1E-8 * sum;

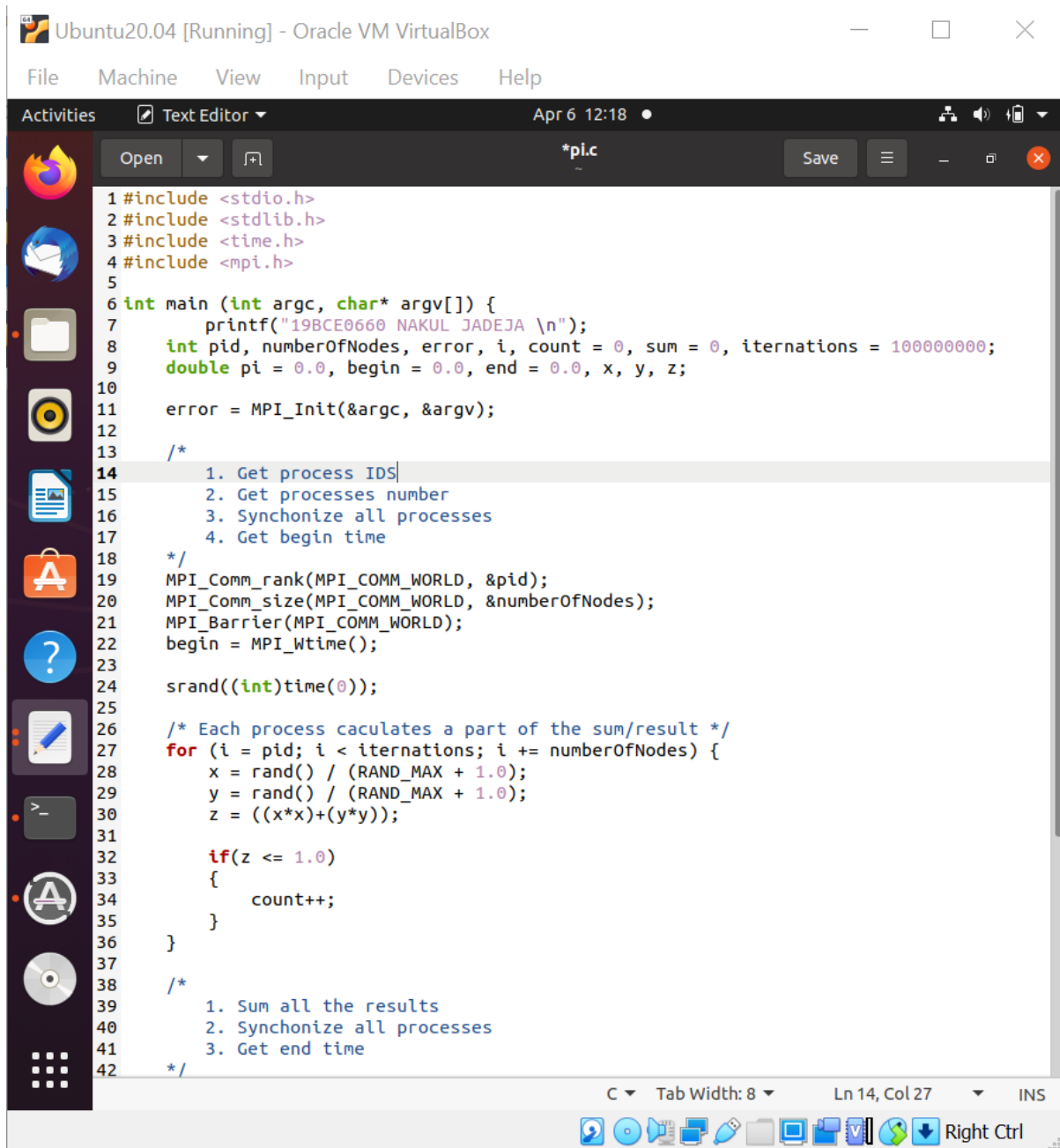
    printf("Processors = %2d;   Time = %f s;   PI = %0.10f\n",
numberOfNodes, end-begin, pi);
}

error=MPI_Finalize();

return 0;
}

```

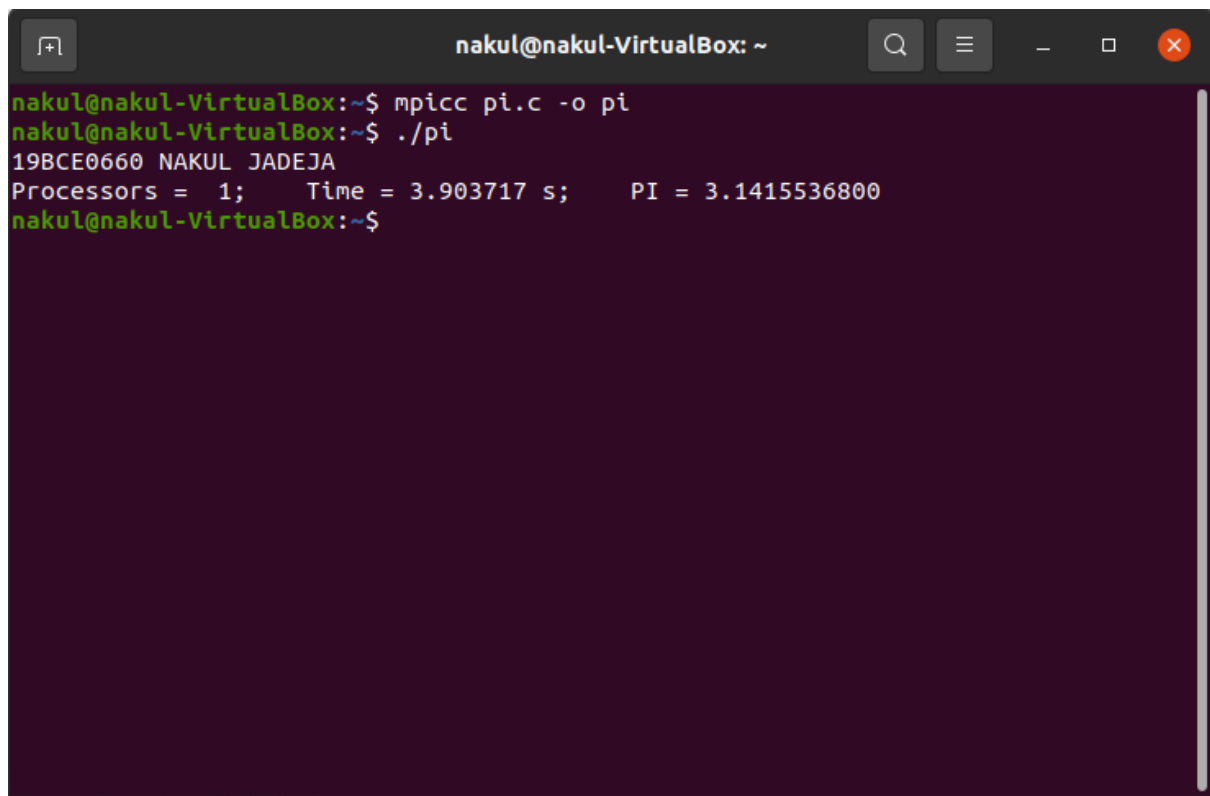
## Screenshot of the Output: -



The screenshot shows a text editor window titled "Ubuntu20.04 [Running] - Oracle VM VirtualBox". The editor is displaying a C program named "pi.c". The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <mpi.h>
5
6 int main (int argc, char* argv[]) {
7     printf("19BCE0660 NAKUL JADEJA \n");
8     int pid, numberOfNodes, error, i, count = 0, sum = 0, iterations = 100000000;
9     double pi = 0.0, begin = 0.0, end = 0.0, x, y, z;
10
11     error = MPI_Init(&argc, &argv);
12
13     /*
14      1. Get process IDS
15      2. Get processes number
16      3. Synchronize all processes
17      4. Get begin time
18     */
19     MPI_Comm_rank(MPI_COMM_WORLD, &pid);
20     MPI_Comm_size(MPI_COMM_WORLD, &numberOfNodes);
21     MPI_Barrier(MPI_COMM_WORLD);
22     begin = MPI_Wtime();
23
24     srand((int)time(0));
25
26     /* Each process caculates a part of the sum/result */
27     for (i = pid; i < iterations; i += numberOfNodes) {
28         x = rand() / (RAND_MAX + 1.0);
29         y = rand() / (RAND_MAX + 1.0);
30         z = ((x*x)+(y*y));
31
32         if(z <= 1.0)
33         {
34             count++;
35         }
36     }
37
38     /*
39      1. Sum all the results
40      2. Synchronize all processes
41      3. Get end time
42     */
```

The editor interface includes a menu bar (File, Machine, View, Input, Devices, Help), a toolbar (Open, Save, etc.), and a status bar at the bottom showing "C", "Tab Width: 8", "Ln 14, Col 27", and "INS". The system tray at the bottom right shows various icons and the text "Right Ctrl".



```
nakul@nakul-VirtualBox: ~  
nakul@nakul-VirtualBox:~$ mpicc pi.c -o pi  
nakul@nakul-VirtualBox:~$ ./pi  
19BCE0660 NAKUL JADEJA  
Processors = 1;    Time = 3.903717 s;    PI = 3.1415536800  
nakul@nakul-VirtualBox:~$
```

**THE END**