

**Veermata Jijabai Technological Institute, Mumbai 400019**

**Experiment No.:** 07

**Aim :** Implementation using MPI .

1. Calculating Rank and Number of processors.,
2. Pi calculation.,
3. Advanced MPI program that has a total number of 4 processes, where the process with rank = 0 should send VJTI letter to all the processes using MPI\_Scatter call.,
4. Find the maximum value in array of six integers with 6 processes, and print the result in root process using MPI\_Reduce call,
5. Ring topology.

**Name:** Kiran K Patil

**Enrolment No.:** 211070904

**Branch:** Computer Engineering

**Batch:** IV

**1. Calculating Rank and Number of processors.,**

Objective - To write a simple MPI program for calculating Rank and Number of processor.

**Program :**

#include <mpi.h>

#include <stdio.h>

int main (int argc, char\* argv[]){

int rank, size;

MPI\_Init (&argc, &argv); /\* starts MPI \*/

MPI\_Comm\_rank (MPI\_COMM\_WORLD, &rank); /\* get current process id \*/

MPI\_Comm\_size (MPI\_COMM\_WORLD, &size); /\* get number of processes \*/

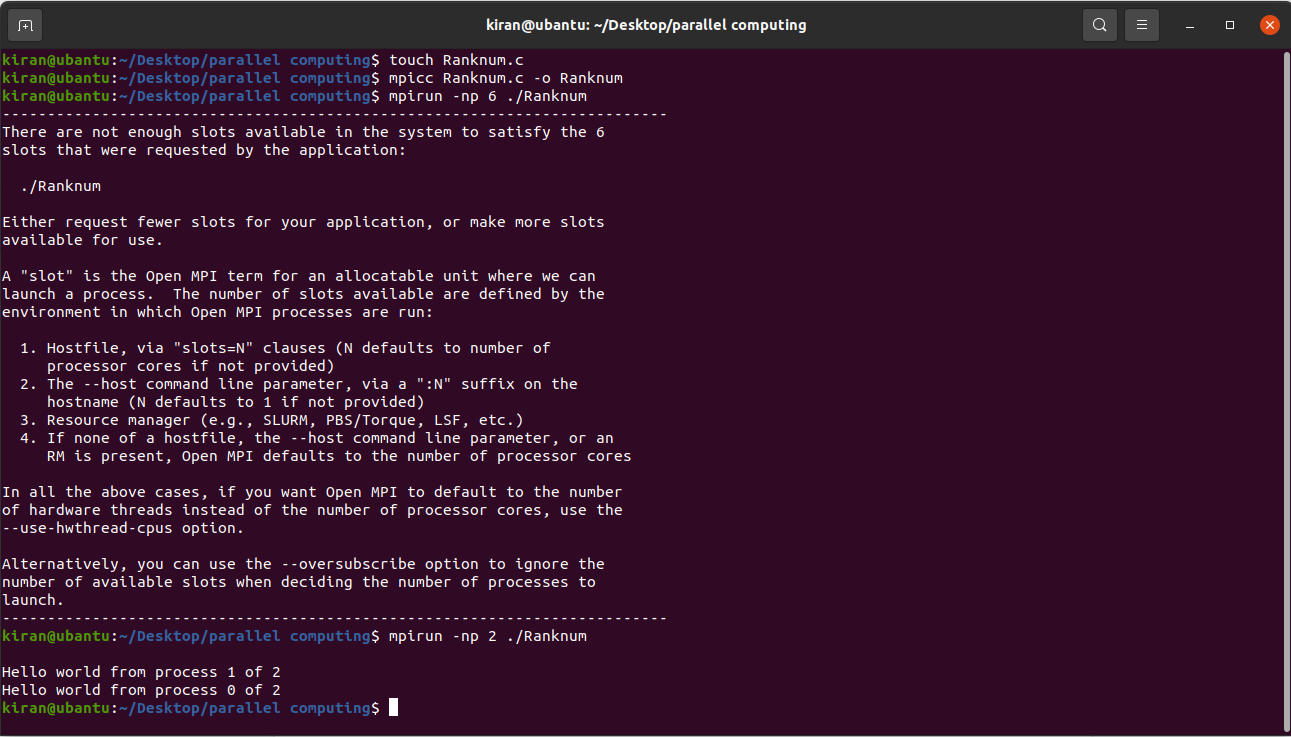
printf( "Hello world from process %d of %d\n", rank, size );

MPI\_Finalize();

return 0;

}

**Output:**



**Conclusion:** Thus, I have implemented MPI program for calculating rank and number of processors.

**2. Pi calculation**

Objective - To write an MPI program for Pi calculation.

Program :

#include “mpi.h”

#include <stdio.h>

#include <math.h>

int main( int argc, char \*argv[] )

{

int n, myid, numprocs, I;

double PI25DT = 3.141592653589793238462643;

double mypi, pi, h, sum, x;

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&myid);

while (1) {

if (myid == 0) {

printf(“Enter the number of intervals: (0 quits) “);

scanf(“%d”,&n);

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (n == 0)

break;

else {

h = 1.0 / (double) n;

sum = 0.0;

for (I = myid + 1; I <= n; I += numprocs) {

x = h \* ((double)I – 0.5);

sum += (4.0 / (1.0 + x\*x));

}

mypi = h \* sum;

MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0,

MPI\_COMM\_WORLD);

if (myid == 0)

printf(“pi is approximately %.16f, Error is %.16f\n”,

pi, fabs(pi – PI25DT));

}

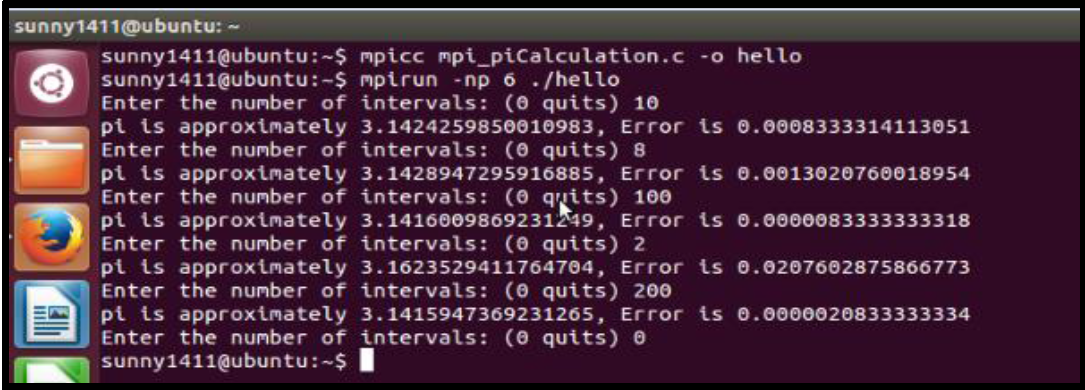
}

MPI\_Finalize();

return 0;

}

Output :

****

**Conclusion:** Thus, I have implemented an MPI program for calculating the value of Pi.

3. Advanced MPI program that has a total number of 4 processes, where the process with rank = 0 should send VJTI letter to all the processes using MPI\_Scatter call.,

Program :

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#define SIZE 4

int main (int argc, char \*argv[]){

int numtasks, rank, sendcount, recvcount, source;

char sendbuf[SIZE][SIZE] = {

{'V','J','T','I'},

{'V','J','T','I'},

{'V','J','T','I'},

{'V','J','T','I'}};

char recvbuf[SIZE];

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);

if (numtasks == SIZE) {

source = 0;

sendcount = SIZE;

recvcount = SIZE;

MPI\_Scatter(sendbuf,sendcount,MPI\_CHAR,recvbuf,recvcount,

MPI\_CHAR,source,MPI\_COMM\_WORLD);

printf("rank= %d Results: %c %c %c %c\n",rank,recvbuf[0],

recvbuf[1],recvbuf[2],recvbuf[3]); }

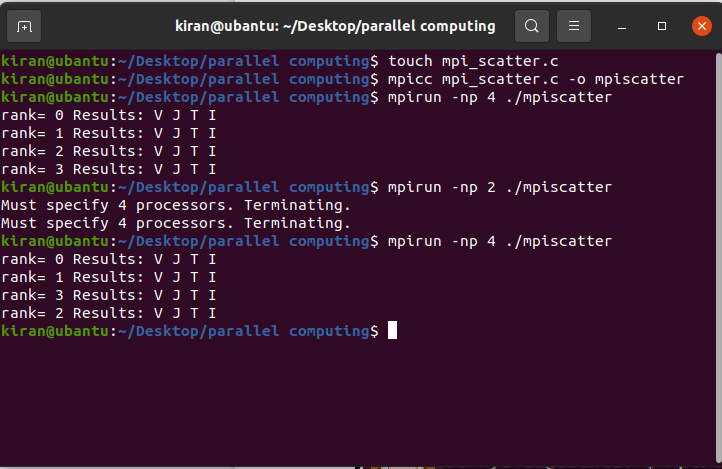
else

printf("Must specify %d processors. Terminating.\n",SIZE);

MPI\_Finalize();

}

Output :



**Conclusion:** Thus, I have implemented an advanced MPI program for scattering “VJTI” to all the processes from the root process using MPI\_Scatter Call.

4. Find the maximum value in array of six integers with 6 processes, and print the result in root process using MPI\_Reduce call,

Program :

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#define SIZE 4

int main (int argc, char \*argv[])

{

int rank,numtasks,array[6] = {100,600,300,800,250,720},i,inputNumber;

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);

printf("Local Input for process %d is %d\n",rank,array[rank]);

inputNumber = array[rank];

int maxNumber;

MPI\_Reduce(&inputNumber, &maxNumber, 1, MPI\_INT, MPI\_MAX, 0,

MPI\_COMM\_WORLD);

// Print the result

if (rank == 0) {

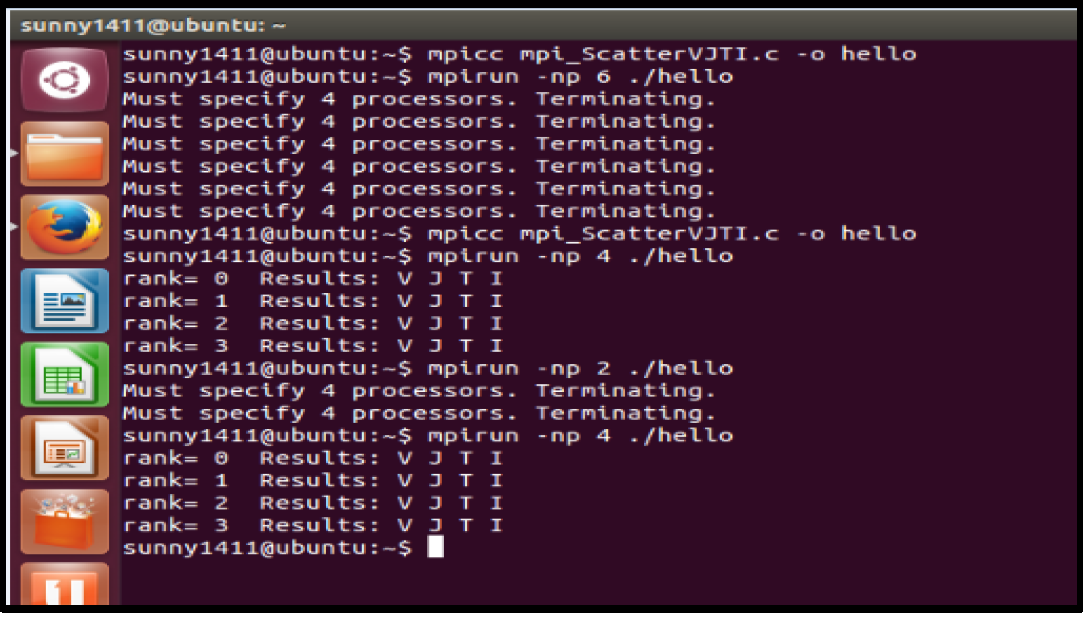
printf("Maximum of all is: %d\n",maxNumber);

}

MPI\_Finalize();

}

Output:

****

**5. Ring topology.**

To write an MPI program for Ring topology.

Program :

#include <stdio.h>

#include "mpi.h"

int main(int argc,char \*argv[])

{

int MyRank, Numprocs, Root = 0;

int value, sum = 0;

int Source, Source\_tag;

int Destination, Destination\_tag;

MPI\_Status status;

/\* Initialize MPI \*/

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&Numprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&MyRank);

if (MyRank == Root){

Destination = MyRank + 1;

Destination\_tag = 0;

MPI\_Send(&MyRank, 1, MPI\_INT, Destination, Destination\_tag,

MPI\_COMM\_WORLD);

}

else{

if(MyRank<Numprocs-1){

Source = MyRank - 1;

Source\_tag = 0;

MPI\_Recv(&value, 1, MPI\_INT, Source, Source\_tag,

MPI\_COMM\_WORLD, &status);

sum = MyRank + value;

Destination = MyRank + 1;

Destination\_tag = 0;

MPI\_Send(&sum, 1, MPI\_INT, Destination, Destination\_tag,

MPI\_COMM\_WORLD);

}

else{

Source = MyRank - 1;

Source\_tag = 0;

MPI\_Recv(&value, 1, MPI\_INT, Source, Source\_tag,

MPI\_COMM\_WORLD, &status);

sum = MyRank + value;

}

}

if (MyRank == Root)

{

Source = Numprocs - 1;

Source\_tag = 0;

MPI\_Recv(&sum, 1, MPI\_INT, Source, Source\_tag,

MPI\_COMM\_WORLD, &status);

printf("MyRank %d Final SUM %d\n", MyRank, sum);

}

if(MyRank == (Numprocs - 1)){

Destination = 0;

Destination\_tag = 0;

MPI\_Send(&sum, 1, MPI\_INT, Destination, Destination\_tag,

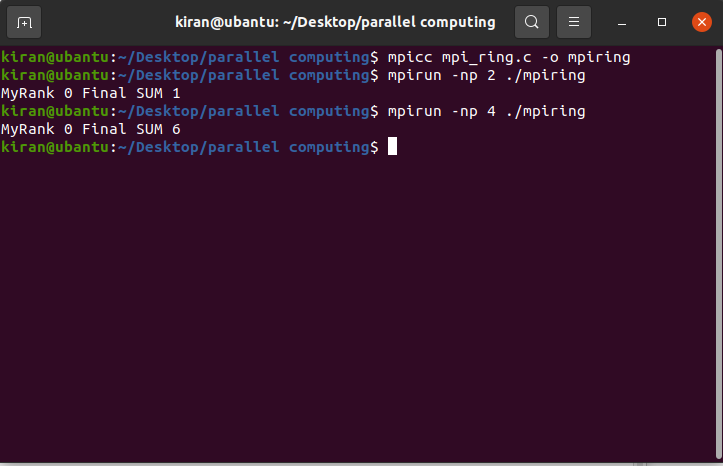
MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

Output :



**Conclusion:** Thus, I have implemented an MPI program for ring topology.

**Conclusion:**

* In conclusion, MPI (Message Passing Interface) is a popular programming model used for developing parallel applications that can run on a distributed computing system. It provides a standardized way of sending and receiving messages between processes, allowing for efficient communication and coordination between them.
* MPI programming can be complex and requires careful consideration of issues such as load balancing, synchronization, and data partitioning. However, it can significantly improve the performance of large-scale scientific simulations, data analytics, and other computationally intensive applications.