**Ex no 5: Implementation of Pi calculation and Matrix multiplication using Point to Point communication**

* **Pi calculation**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <mpi.h>

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank, size;

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

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

// Number of total random points

int total\_points = 1000000;

// Points inside the quarter-circle

int points\_inside = 0;

// Initialize random seed based on process rank

srand(time(NULL) + rank);

// Generate random points and count those inside the quarter-circle

for (int i = 0; i < total\_points; ++i) {

double x = (double)rand() / RAND\_MAX;

double y = (double)rand() / RAND\_MAX;

if (x \* x + y \* y <= 1.0) {

points\_inside++;

}}

// Sum up points\_inside across all processes using MPI\_Reduce

int global\_points\_inside;

MPI\_Reduce(&points\_inside, &global\_points\_inside, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

// Only the master process (rank 0) performs the final calculation and prints the result

if (rank == 0) {

double pi\_estimate = 4.0 \* global\_points\_inside / (double)(total\_points \* size);

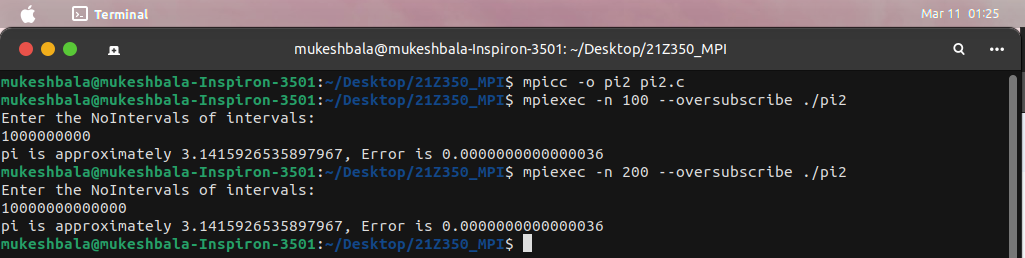
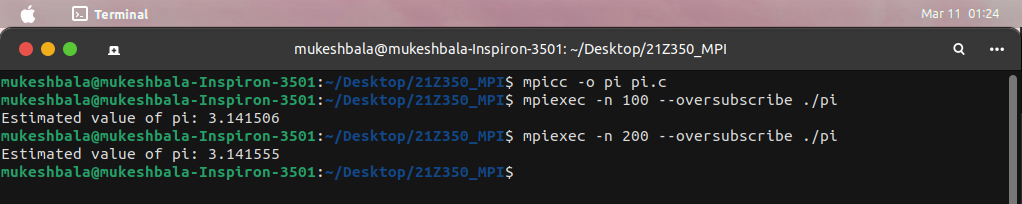
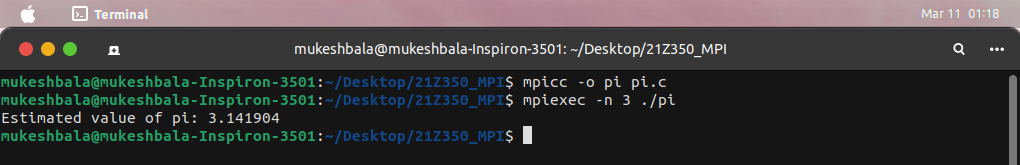
printf("Estimated value of pi: %lf\n", pi\_estimate);

}

MPI\_Finalize();

return 0;}

**Output:**

****

* **Matrix Multiplication**

#include <stdlib.h>

#include <stdio.h>

#include "mpi.h"

#include <time.h>

#include <sys/time.h>

// Number of rows and columnns in a matrix

#define N 4

MPI\_Status status;

// Matrix holders are created

double matrix\_a[N][N],matrix\_b[N][N],matrix\_c[N][N];

int main(int argc, char \*\*argv)

{

int processCount, processId, slaveTaskCount, source, dest, rows, offset;

struct timeval start, stop;

MPI\_Init(&argc, &argv);

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

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

slaveTaskCount = processCount - 1;

if (processId == 0) {

srand ( time(NULL) );

for (int i = 0; i<N; i++) {

for (int j = 0; j<N; j++) {

matrix\_a[i][j]= rand()%10;

matrix\_b[i][j]= rand()%10;

}

}

printf("\n\t\tMatrix - Matrix Multiplication using MPI\n");

printf("\nMatrix A\n\n");

for (int i = 0; i<N; i++) {

for (int j = 0; j<N; j++) {

printf("%.0f\t", matrix\_a[i][j]);

}

printf("\n");

}

printf("\nMatrix B\n\n");

for (int i = 0; i<N; i++) {

for (int j = 0; j<N; j++) {

printf("%.0f\t", matrix\_b[i][j]);

}

printf("\n");

}

rows = N/slaveTaskCount;

offset = 0;

for (dest=1; dest <= slaveTaskCount; dest++)

{

MPI\_Send(&offset, 1, MPI\_INT, dest, 1, MPI\_COMM\_WORLD);

MPI\_Send(&rows, 1, MPI\_INT, dest, 1, MPI\_COMM\_WORLD);

MPI\_Send(&matrix\_a[offset][0], rows\*N, MPI\_DOUBLE,dest,1, MPI\_COMM\_WORLD);

// Matrix B is sent

MPI\_Send(&matrix\_b, N\*N, MPI\_DOUBLE, dest, 1, MPI\_COMM\_WORLD);

// Offset is modified according to number of rows sent to each process

offset = offset + rows;

}

// Root process waits untill the each slave proces sent their calculated result with message tag 2

for (int i = 1; i <= slaveTaskCount; i++)

{

source = i;

// Receive the offset of particular slave process

MPI\_Recv(&offset, 1, MPI\_INT, source, 2, MPI\_COMM\_WORLD, &status);

// Receive the number of rows that each slave process processed

MPI\_Recv(&rows, 1, MPI\_INT, source, 2, MPI\_COMM\_WORLD, &status);

// Calculated rows of the each process will be stored int Matrix C according to their offset and

// the processed number of rows

MPI\_Recv(&matrix\_c[offset][0], rows\*N, MPI\_DOUBLE, source, 2, MPI\_COMM\_WORLD, &status);

}

// Print the result matrix

printf("\nResult Matrix C = Matrix A \* Matrix B:\n\n");

for (int i = 0; i<N; i++) {

for (int j = 0; j<N; j++)

printf("%.0f\t", matrix\_c[i][j]);

printf ("\n");

}

printf ("\n");

}

// Slave Processes

if (processId > 0) {

// Source process ID is defined

source = 0;

MPI\_Recv(&offset, 1, MPI\_INT, source, 1, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&rows, 1, MPI\_INT, source, 1, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&matrix\_a, rows\*N, MPI\_DOUBLE, source, 1, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&matrix\_b, N\*N, MPI\_DOUBLE, source, 1, MPI\_COMM\_WORLD, &status); for (int k = 0; k<N; k++) {

for (int i = 0; i<rows; i++) {

matrix\_c[i][k] = 0.0;

for (int j = 0; j<N; j++)

matrix\_c[i][k] = matrix\_c[i][k] + matrix\_a[i][j] \* matrix\_b[j][k];

}

}

MPI\_Send(&offset, 1, MPI\_INT, 0, 2, MPI\_COMM\_WORLD);

// Number of rows the process calculated will be sent to root process

MPI\_Send(&rows, 1, MPI\_INT, 0, 2, MPI\_COMM\_WORLD);

// Resulting matrix with calculated rows will be sent to root process

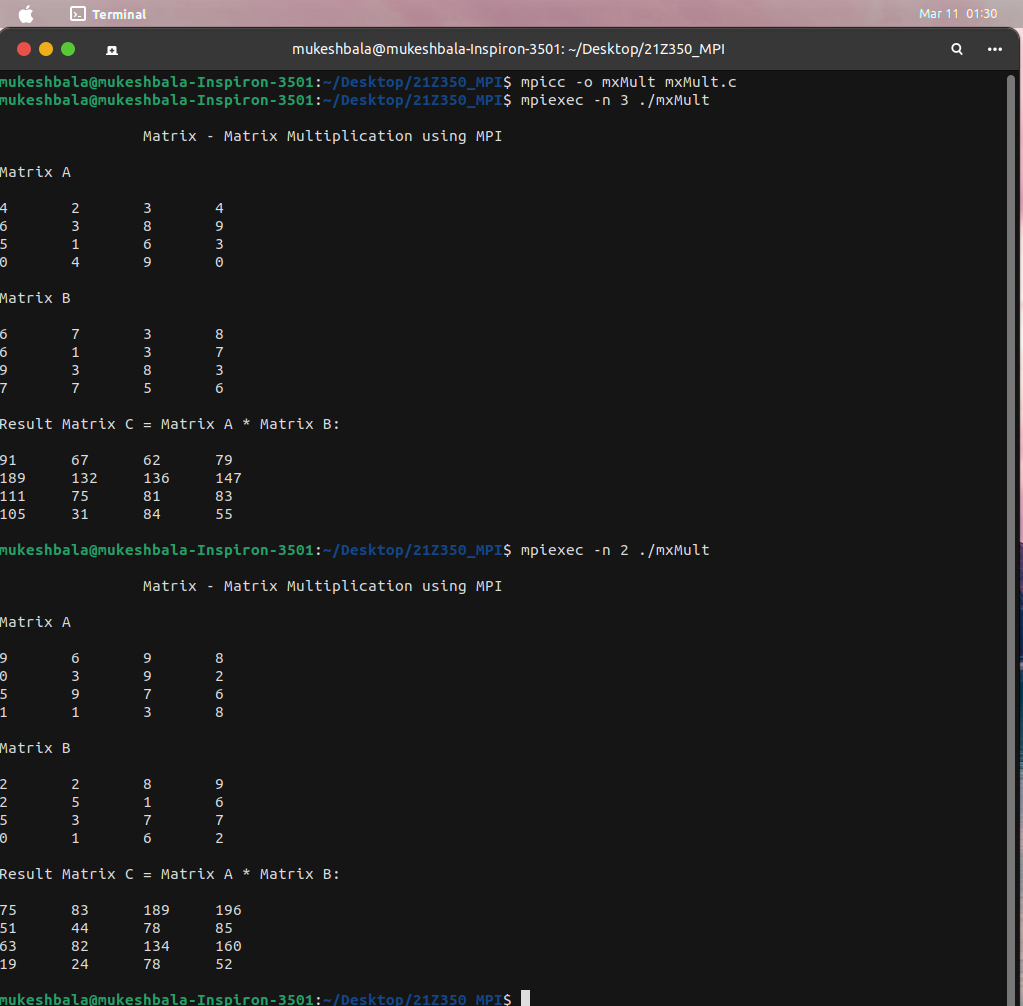
MPI\_Send(&matrix\_c, rows\*N, MPI\_DOUBLE, 0, 2, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

**Output:**

****