**Ex no 6: Implementation of Pi calculation and Matrix multiplication using Broadcast communication**

* **Pi calculation**

#include "mpi.h"

#include <stdio.h>

#include <math.h>

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

int done = 0, 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;

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));

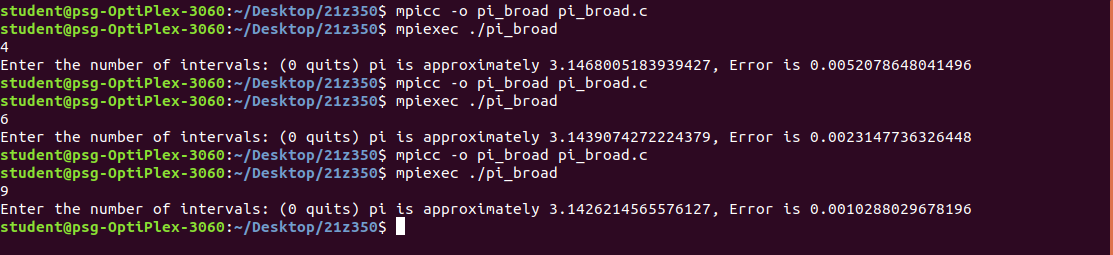
break; // Exit loop after calculating and printing pi value

} 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 columns 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, 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;

// Broadcast matrix\_a and matrix\_b to all processes

MPI\_Bcast(matrix\_a, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(matrix\_b, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

for (int 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);

offset = offset + rows;

}

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

source = i;

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

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

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

}

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");

}

if (processId > 0) {

// Broadcast matrix\_a and matrix\_b to all slave processes

MPI\_Bcast(matrix\_a, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(matrix\_b, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&offset, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&rows, 1, MPI\_INT, 0, 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\_a[i][j] \* matrix\_b[j][k];

}

}

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

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

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

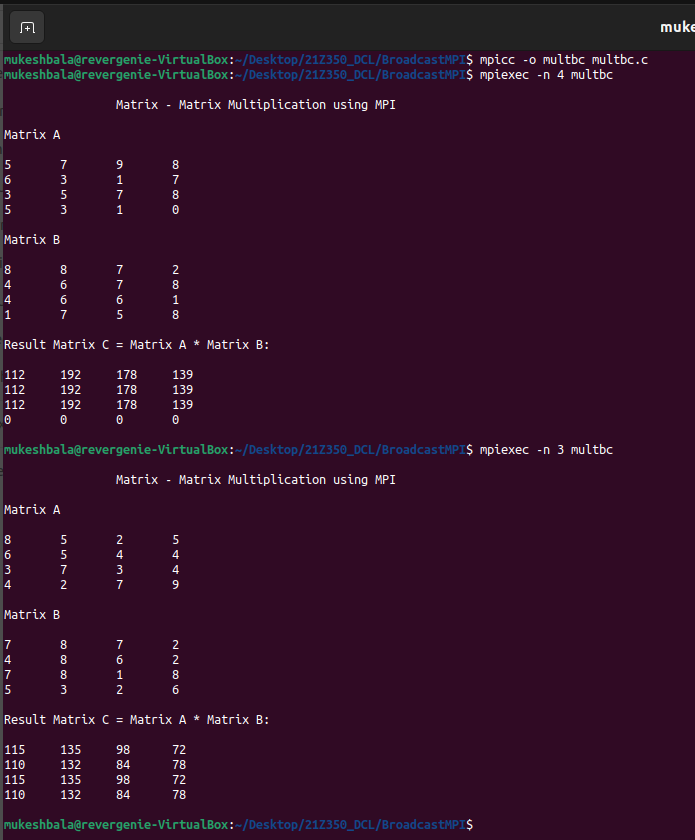
}

MPI\_Finalize();

return 0;

}

**Output:**

****