**Class:** Final Year (Computer Science and Engineering)

**Year:** 2022-23 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 7**

**Exam Seat No:** 2019BTECS00070

**Name:** Killedar Prathmesh

1. Implement Matrix-Vector Multiplication using MPI. Use different number of processes and analyze the performance.

**Code:-**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

// size of matrix

#define N 1000

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

{

*int* np, rank, numworkers, rows, i, j, k;

// a\*b = c

*double* a[N][N], b[N], c[N];

MPI\_Status status;

MPI\_Init(&argc, &argv);

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

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

numworkers = np - 1; // total process - 1 ie process with rank 0

// rank with 0 is a master process

*int* dest, source;

*int* tag;

*int* rows\_per\_process, extra, offset;

// master process, process with rank = 0

if (rank == 0)

{

printf("Running with %d tasks.\n", np);

// matrix a and b initialization

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

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

a[i][j] = 1;

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

b[i] = 1;

// start time

*double* start = MPI\_Wtime();

// Send matrix data to other worker processes

rows\_per\_process = N / numworkers;

extra = N % numworkers;

offset = 0;

tag = 1;

// send data to other nodes

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

{

rows = (dest <= extra) ? rows\_per\_process + 1 : rows\_per\_process;

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

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

MPI\_Send(&a[offset][0], rows \* N, MPI\_DOUBLE, dest, tag, MPI\_COMM\_WORLD);

MPI\_Send(&b, N, MPI\_DOUBLE, dest, tag, MPI\_COMM\_WORLD);

offset = offset + rows;

}

// receive data from other nodes and add it to the ans matrix c

tag = 2;

for (i = 1; i <= numworkers; i++)

{

source = i;

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

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

MPI\_Recv(&c[offset], N, MPI\_DOUBLE, source, tag, MPI\_COMM\_WORLD, &status);

}

// print multiplication result

// printf("Result Matrix:\n");

// for (i = 0; i < N; i++)

// {

// printf("%6.2f ", c[i]);

// }

// printf("\n");

*double* finish = MPI\_Wtime();

printf("Done in %f seconds.\n", finish - start); // total time spent

}

// all other process than process with rank = 0

if (rank > 0)

{

tag = 1;

// receive data from process with rank 0

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

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

MPI\_Recv(&a, rows \* N, MPI\_DOUBLE, 0, tag, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&b, N, MPI\_DOUBLE, 0, tag, MPI\_COMM\_WORLD, &status);

// calculate multiplication of given rows

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

{

c[i] = 0.0;

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

c[i] = c[i] + a[i][j] \* b[j];

}

// send result back to process with rank 0

tag = 2;

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

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

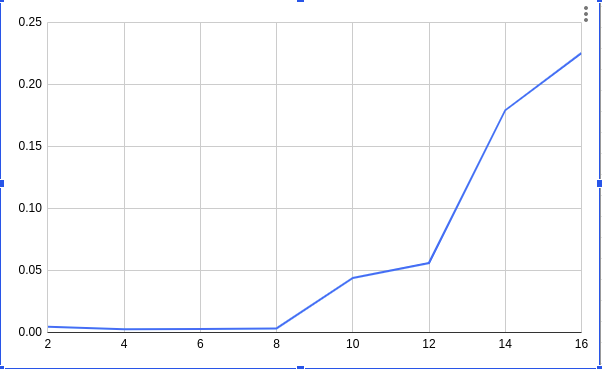
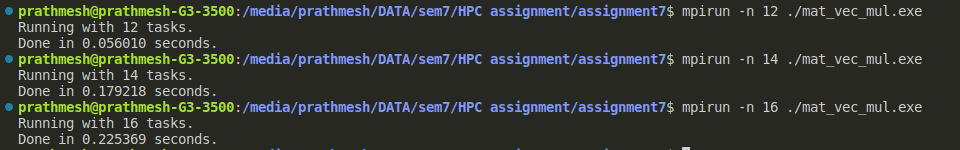
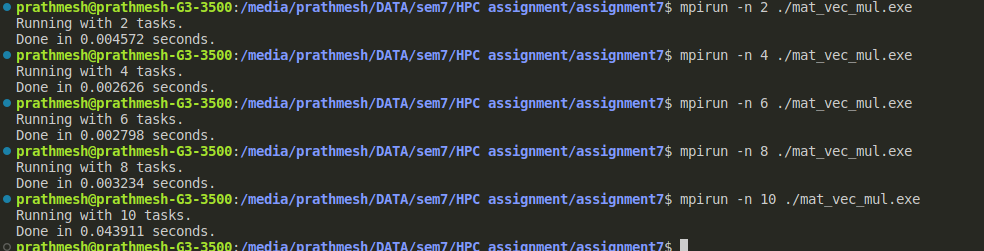
MPI\_Send(&c, N, MPI\_DOUBLE, 0, tag, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

**Output:-**

****

2. Implement Matrix-Matrix Multiplication using MPI. Use different number of processes and analyze the performance.

**Code:**

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#define MATSIZE 500

#define NRA MATSIZE /\* number of rows in matrix A \*/

#define NCA MATSIZE /\* number of columns in matrix A \*/

#define NCB MATSIZE /\* number of columns in matrix B \*/

#define MASTER 0 /\* taskid of first task \*/

#define FROM\_MASTER 1 /\* setting a message type \*/

#define FROM\_WORKER 2 /\* setting a message type \*/

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

{

*int* numtasks, /\* number of tasks in partition \*/

taskid, /\* a task identifier \*/

numworkers, /\* number of worker tasks \*/

source, /\* task id of message source \*/

dest, /\* task id of message destination \*/

mtype, /\* message type \*/

rows, /\* rows of matrix A sent to each worker \*/

averow, extra, offset, /\* used to determine rows sent to each worker \*/

i, j, k, rc; /\* misc \*/

*double* a[NRA][NCA], /\* matrix A to be multiplied \*/

b[NCA][NCB], /\* matrix B to be multiplied \*/

c[NRA][NCB]; /\* result matrix C \*/

MPI\_Status status;

MPI\_Init(&argc, &argv);

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

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

if (numtasks < 2)

{

printf("Need at least two MPI tasks. Quitting...\n");

MPI\_Abort(MPI\_COMM\_WORLD, rc);

exit(1);

}

numworkers = numtasks - 1;

/\*\*\*\*\*\*\*\*\*\* master task \*\*\*\*\*\*\*\*\*\*\*\*/

if (taskid == MASTER)

{

printf("mpi\_mm has started with %d tasks.\n", numtasks);

// printf("Initializing arrays...\n");

for (i = 0; i < NRA; i++)

for (j = 0; j < NCA; j++)

a[i][j] = i + j;

for (i = 0; i < NCA; i++)

for (j = 0; j < NCB; j++)

b[i][j] = i \* j;

/\* Measure start time \*/

*double* start = MPI\_Wtime();

/\* Send matrix data to the worker tasks \*/

averow = NRA / numworkers;

extra = NRA % numworkers;

offset = 0;

mtype = FROM\_MASTER;

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

{

rows = (dest <= extra) ? averow + 1 : averow;

// printf("Sending %d rows to task %d offset=%d\n",rows,dest,offset);

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

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

MPI\_Send(&a[offset][0], rows \* NCA, MPI\_DOUBLE, dest, mtype,

MPI\_COMM\_WORLD);

MPI\_Send(&b, NCA \* NCB, MPI\_DOUBLE, dest, mtype, MPI\_COMM\_WORLD);

offset = offset + rows;

}

/\* Receive results from worker tasks \*/

mtype = FROM\_WORKER;

for (i = 1; i <= numworkers; i++)

{

source = i;

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

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

MPI\_Recv(&c[offset][0], rows \* NCB, MPI\_DOUBLE, source, mtype,

MPI\_COMM\_WORLD, &status);

// printf("Received results from task %d\n",source);

}

/\* Print results \*/

/\*

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Result Matrix:\n");

for (i=0; i<NRA; i++)

{

printf("\n");

for (j=0; j<NCB; j++)

printf("%6.2f ", c[i][j]);

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

\*/

/\* Measure finish time \*/

*double* finish = MPI\_Wtime();

printf("Done in %f seconds.\n", finish - start);

}

/\*\*\*\*\*\*\*\*\*\* worker task \*\*\*\*\*\*\*\*\*\*\*\*/

if (taskid > MASTER)

{

mtype = FROM\_MASTER;

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

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

MPI\_Recv(&a, rows \* NCA, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&b, NCA \* NCB, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD, &status);

for (k = 0; k < NCB; k++)

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

{

c[i][k] = 0.0;

for (j = 0; j < NCA; j++)

c[i][k] = c[i][k] + a[i][j] \* b[j][k];

}

mtype = FROM\_WORKER;

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

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

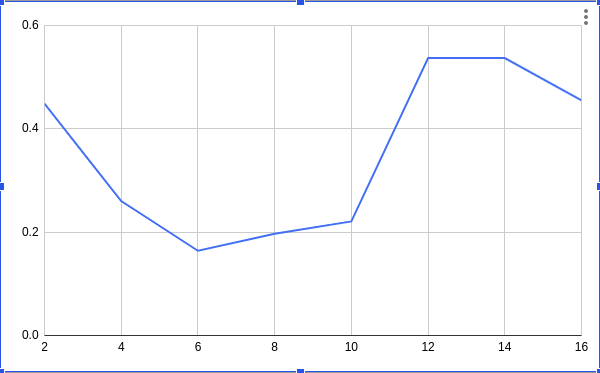
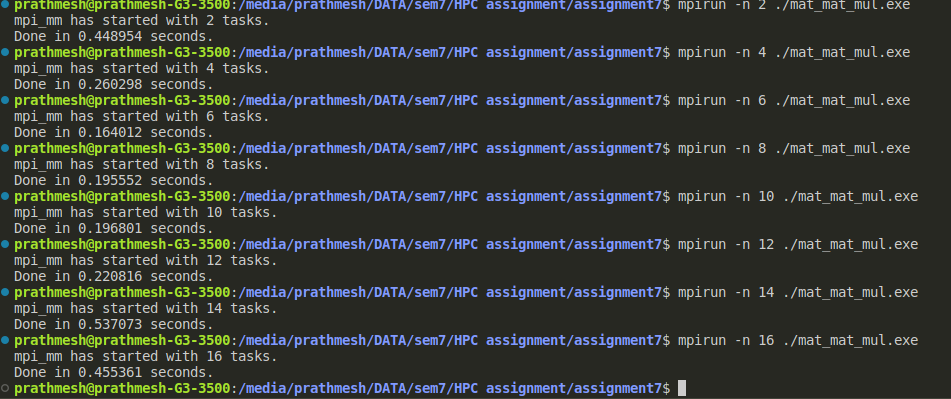
MPI\_Send(&c, rows \* NCB, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

**Output:**



**Github Link:**

**https://github.com/killedar27/HPC-assignments/tree/main/assignment7**