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

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

**Course:** High Performance Computing Lab

**Practical No. 7**

**Exam Seat No:**

2019BTECS00051-Atharv Inamdar

**Title of practical:**

Implementation of Matrix-Vector & Matrix-Matrix Multiplication in MPI

**Problem Statement 1:**

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

**Screenshot 1:**

#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);

        }

        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);

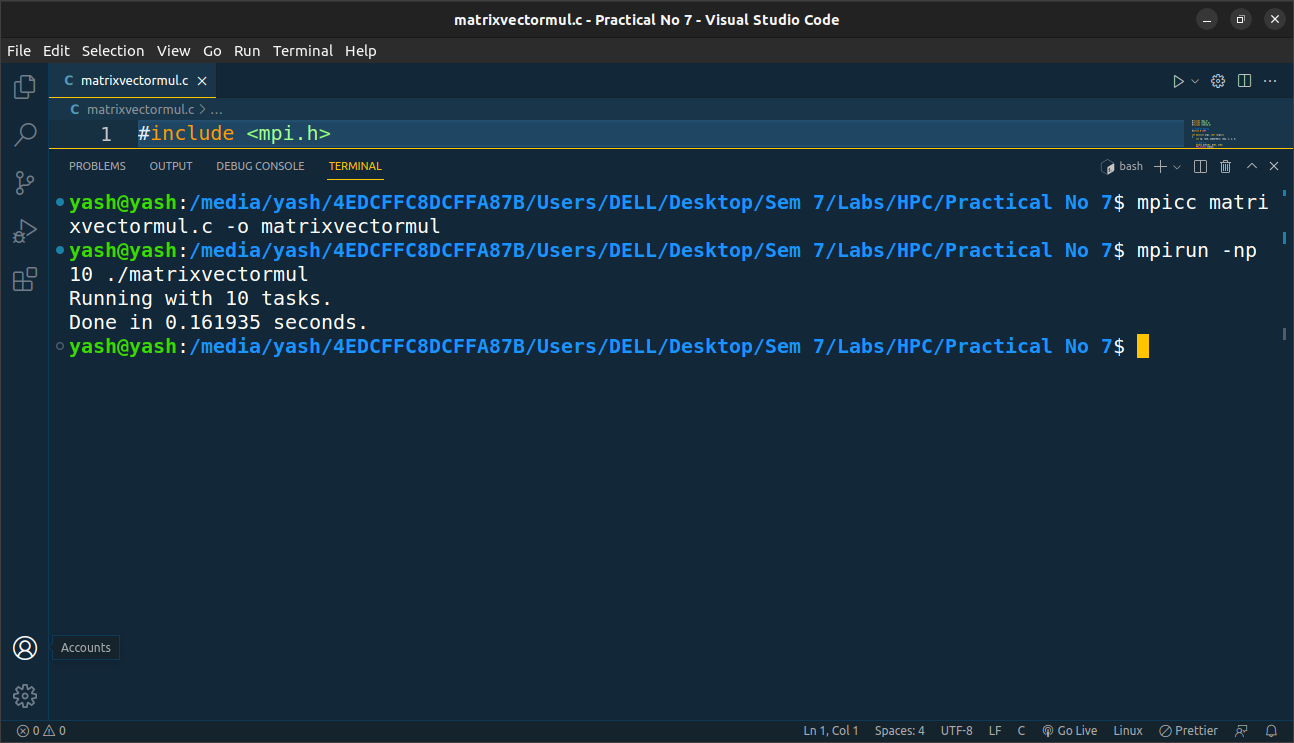
        MPI\_Send(&c, N, MPI\_DOUBLE, 0, tag, MPI\_COMM\_WORLD);

    }

    MPI\_Finalize();

}

**Screenshot 2:**

****

**Problem Statement 2:**

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

**Screenshot 3:**

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 4 *// matrix size*

#define BS N / 2 *// block size*

MPI\_Status status;

void printMatrix(int matrix[N][N]);

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

{

    int nproc, taskId, source, i, j, k, positionX, positionY;

    MPI\_Datatype type;

    int result[BS][BS] = {0};

    int resultFinal[N][N] = {0};

    int a[N][N], b[N][N];

    MPI\_Init(&argc, &argv);

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

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

    MPI\_Type\_vector(N, BS, N, MPI\_INT, &type);

    MPI\_Type\_commit(&type);

*// root*

    if (taskId == 0)

    {

        srand(time(NULL));

*// Generate two NxN matrix*

        for (i = 0; i < N; i++)

        {

            for (j = 0; j < N; j++)

            {

                a[i][j] = rand() % 10;

                b[i][j] = rand() % 10;

            }

        }

        printf("First matrix:\n");

        printMatrix(a);

        printf("Second matrix:\n");

        printMatrix(b);

*//      First matrix first block*

        MPI\_Send(&a[0][0], BS \* N, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

        MPI\_Send(&a[0][0], BS \* N, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

*//      First matrix second block*

        MPI\_Send(&a[BS][0], BS \* N, MPI\_INT, 2, 2, MPI\_COMM\_WORLD);

        MPI\_Send(&a[BS][0], BS \* N, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

*//      Second matrix first block*

        MPI\_Send(&b[0][0], 1, type, 0, 0, MPI\_COMM\_WORLD);

        MPI\_Send(&b[0][0], 1, type, 2, 2, MPI\_COMM\_WORLD);

*//      Second matrix second block*

        MPI\_Send(&b[0][BS], 1, type, 1, 1, MPI\_COMM\_WORLD);

        MPI\_Send(&b[0][BS], 1, type, 3, 3, MPI\_COMM\_WORLD);

    }

*// workers*

    source = 0;

    MPI\_Recv(&a, BS \* N, MPI\_INT, source, taskId, MPI\_COMM\_WORLD, &status);

    MPI\_Recv(&b, 1, type, source, taskId, MPI\_COMM\_WORLD, &status);

    MPI\_Type\_free(&type);

*// multiplication*

    for (k = 0; k < BS; k++)

        for (i = 0; i < BS; i++)

        {

            for (j = 0; j < N; j++)

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

        }

*// Send result to root*

    MPI\_Send(&result[0][0], BS \* BS, MPI\_INT, 0, 4, MPI\_COMM\_WORLD);

*// root receives results*

    if (taskId == 0)

    {

        for (i = 0; i < nproc; i++)

        {

            source = i;

            MPI\_Recv(&result, BS \* BS, MPI\_INT, source, 4, MPI\_COMM\_WORLD, &status);

*// Manage shifting*

            if (source == 0)

            {

                positionX = 0;

                positionY = 0;

            }

            else if (source == 1)

            {

                positionX = 0;

                positionY = BS;

            }

            else if (source == 2)

            {

                positionX = BS;

                positionY = 0;

            }

            else if (source == 3)

            {

                positionX = BS;

                positionY = BS;

            }

            for (k = 0; k < BS; k++)

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

                    resultFinal[k + positionX][j + positionY] = result[k][j];

        }

        printf("Result matrix:\n");

        printMatrix(resultFinal);

    }

    MPI\_Finalize();

}

void printMatrix(int matrix[N][N])

{

    int i, j;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

            printf("%d \t", matrix[i][j]);

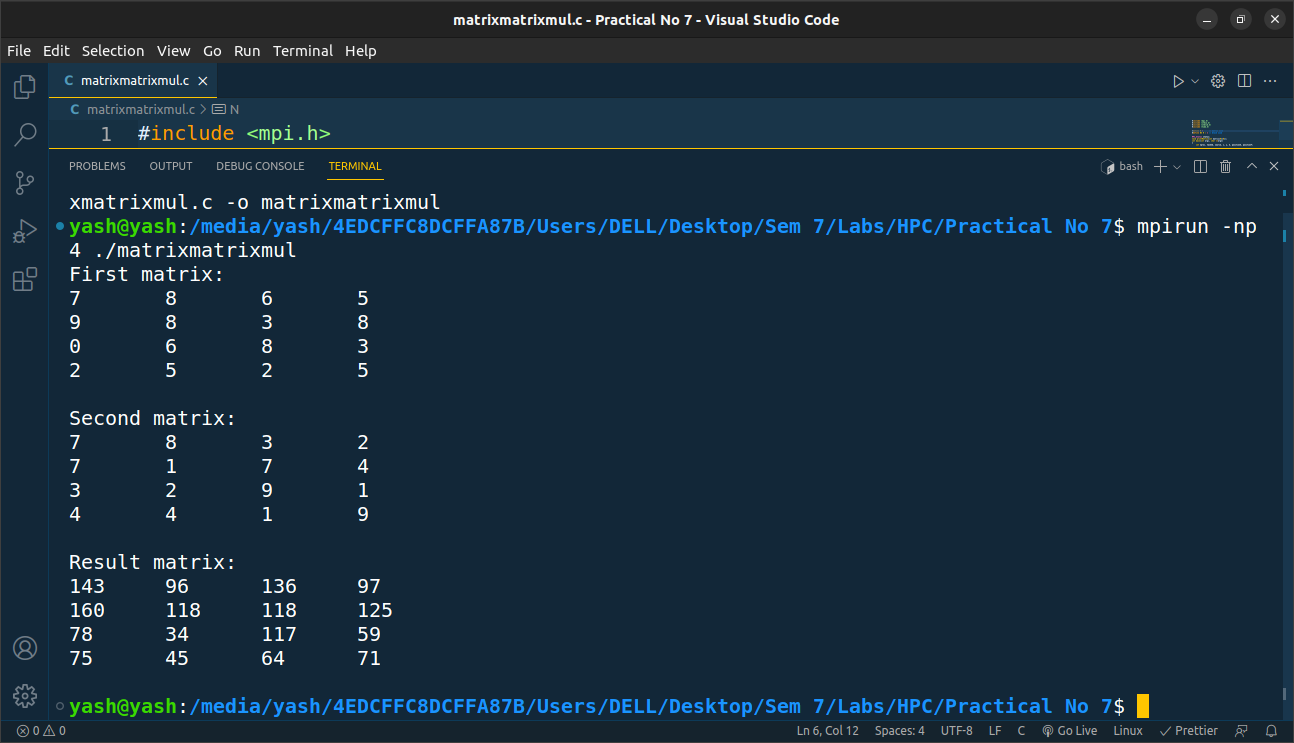
        printf("\n");

    }

    printf("\n");

}

**Screenshot 4:**

****