Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

High Performance Computing Lab

Class: Final Year (Computer Science and Engineering) **Year:** 2022-23

PRN: 2019BTECS00089 – Piyush Pramod Mhaske **Batch**: B3

Practical 7: MPI

Github link: https://github.com/Piyush4620/2019BTECS00089HPCLab

Hosted Link : https://better-sidecar-c10.notion.site/HPC-038e2693a633408c8604841fc50f74e2

Problem Statement 1:

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

```
#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();
}
```

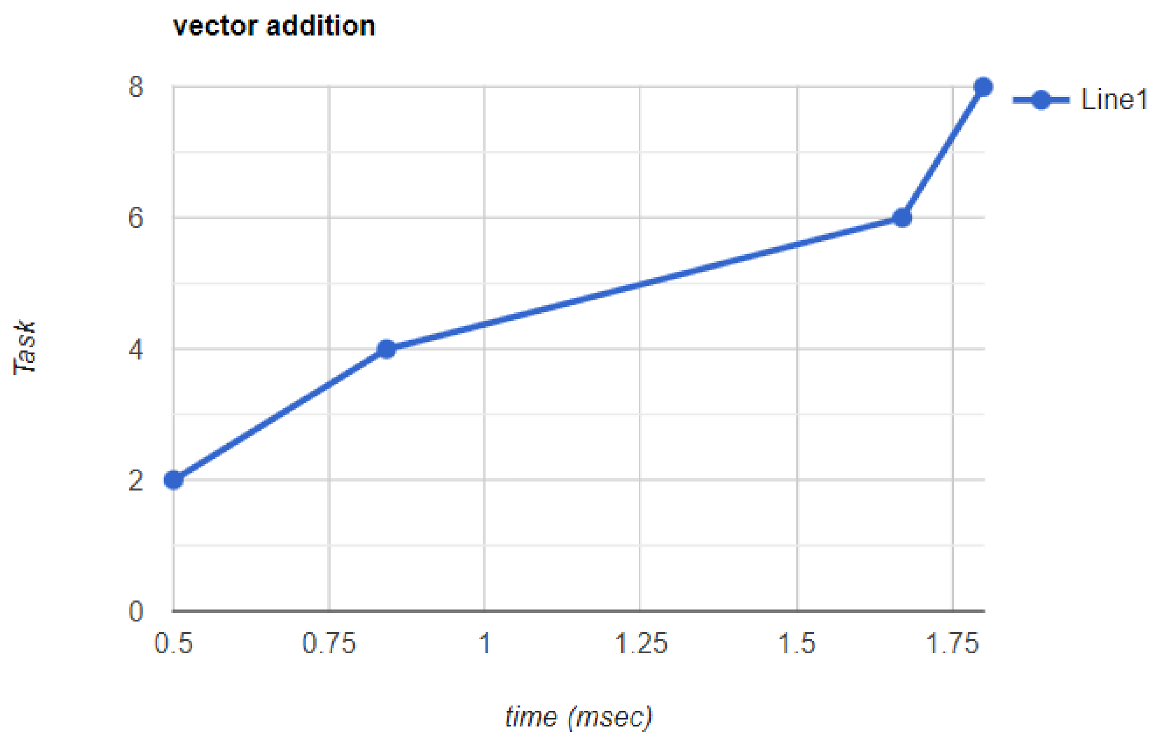
Output:

```
C:\Users\Sameer Dhote\Desktop\MPI>mpiexec -n 2 matrix-vector.exe
Running with 2 tasks.
Done in 0.000500 seconds.

C:\Users\Sameer Dhote\Desktop\MPI>mpiexec -n 4 matrix-vector.exe
Running with 4 tasks.
Done in 0.000842 seconds.

C:\Users\Sameer Dhote\Desktop\MPI>mpiexec -n 6 matrix-vector.exe
Running with 6 tasks.
Done in 0.001679 seconds.

C:\Users\Sameer Dhote\Desktop\MPI>mpiexec -n 8 matrix-vector.exe
Running with 8 tasks.
Done in 0.001803 seconds.
```



Problem Statement 2:

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

```
#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");
}
```

Output:

```
computing slice 1 (from row 4 to 7)
computing slice 0 (from row 0 to 3)



        | 33 34 35 36 37 38 39 40 |
        | 41 42 43 44 45 46 47 48 |
        | 49 50 51 52 53 54 55 56 |
        | 57 58 59 60 61 62 63 64 |
        | 33 34 35 36 37 38 39 40 |
        | 41 42 43 44 45 46 47 48 |
        | 49 50 51 52 53 54 55 56 |
        | 57 58 59 60 61 62 63 64 |

              *

        | 65 66 67 68 69 70 71 72 |
        | 73 74 75 76 77 78 79 80 |
        | 81 82 83 84 85 86 87 88 |
        | 89 90 91 92 93 94 95 96 |
        | 97 98 99 100 101 102 103 104 |
        | 105 106 107 108 109 110 111 112 |
        | 113 114 115 116 117 118 119 120 |
        | 121 122 123 124 125 126 127 128 |

              =

        | 27492 27784 28076 28368 28660 28952 29244 29536 |
        | 33444 33800 34156 34512 34868 35224 35580 35936 |
        | 39396 39816 40236 40656 41076 41496 41916 42336 |
        | 45348 45832 46316 46800 47284 47768 48252 48736 |
        | 27492 27784 28076 28368 28660 28952 29244 29536 |
        | 33444 33800 34156 34512 34868 35224 35580 35936 |
        | 39396 39816 40236 40656 41076 41496 41916 42336 |
        | 45348 45832 46316 46800 47284 47768 48252 48736 |

Exection Time: 0.000632
```