Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

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

Course: High Performance Computing Lab

Practical No. 7

Exam Seat No: 2019BTECS00064

Name – Kunal Santosh Kadam

Problem Statement 1:

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

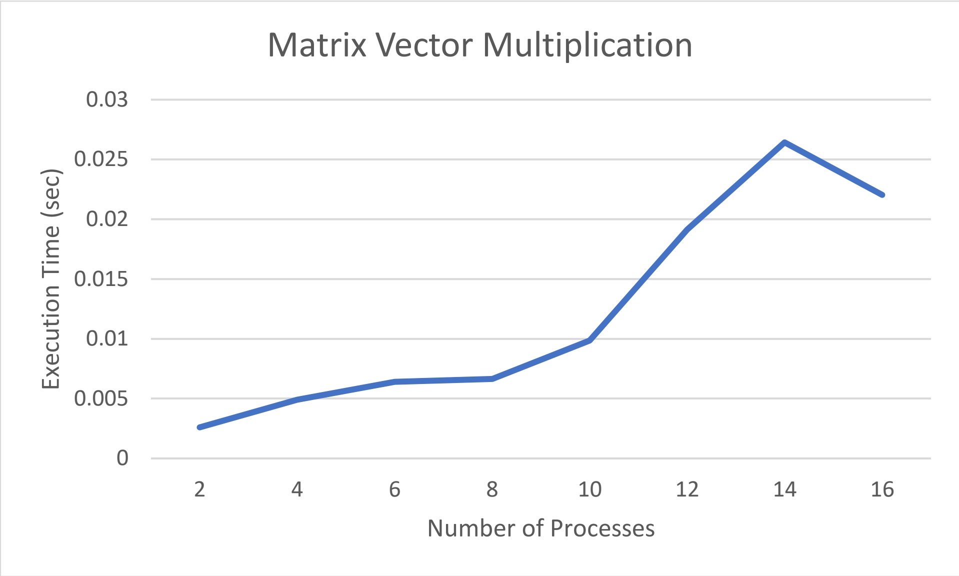
Screenshot #:

```
C:\Windows\System32\cmd.exe
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 2 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 2 processor(s).
Running with 2 tasks.
Done in 0.002598 seconds.
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 4 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 4 processor(s).
Running with 4 tasks.
Done in 0.004906 seconds.
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 6 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 6 processor(s).
Running with 6 tasks.
Done in 0.006398 seconds.
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 8 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 8 processor(s).
Running with 8 tasks.
Done in 0.006654 seconds.
```

```
C:\Windows\System32\cmd.exe

G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 10 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 10 processor(s).
Running with 10 tasks.
Done in 0.009870 seconds.
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 12 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 12 processor(s).
Running with 12 tasks.
Done in 0.019149 seconds.
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 14 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 14 processor(s).
Running with 14 tasks.
Done in 0.026428 seconds.
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 16 Assignment7_1.exe
Multiplying a 500x500 matrix and 500x1 vector using 16 processor(s).
Running with 16 tasks.
Done in 0.022028 seconds.
```

Processors (Size 500)	Execution Time (sec)
2	0.002598
4	0.004906
6	0.006398
8	0.006654
10	0.00987
12	0.019149
14	0.026428
16	0.022028



Information #:

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

// size of matrix
#define N 500

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("\nMultiplying a %dx%d matrix and %dx1 vector
using %d processor(s).\n\n", N, N, N, np);

        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();
}
```

Problem Statement 2:

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

Screenshot #:

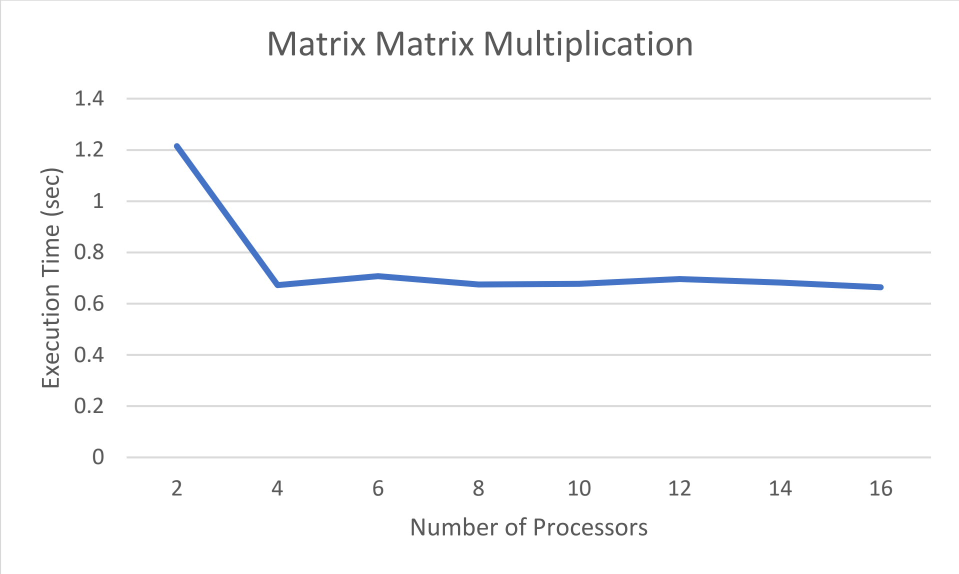
```
C:\Windows\System32\cmd.exe

G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 2 Assignment7_2.exe
Multiplying a 512x512 matrix using 2 processor(s).
Runtime:  1.214729 seconds
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 4 Assignment7_2.exe
Multiplying a 512x512 matrix using 4 processor(s).
Runtime:  0.672029 seconds
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 6 Assignment7_2.exe
Multiplying a 512x512 matrix using 6 processor(s).
Runtime:  0.706800 seconds
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 8 Assignment7_2.exe
Multiplying a 512x512 matrix using 8 processor(s).
Runtime:  0.674825 seconds
```

C:\Windows\System32\cmd.exe

```
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 10 Assignment7_2.exe
Multiplying a 512x512 matrix using 10 processor(s).
Runtime:  0.677104 seconds
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 12 Assignment7_2.exe
Multiplying a 512x512 matrix using 12 processor(s).
Runtime:  0.695681 seconds
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 14 Assignment7_2.exe
Multiplying a 512x512 matrix using 14 processor(s).
Runtime:  0.682626 seconds
G:\Kunal\Sem-7\HPC Assignments\Assignment - 7>mpiexec.exe -np 16 Assignment7_2.exe
Multiplying a 512x512 matrix using 16 processor(s).
Runtime:  0.663857 seconds
```

Processors (Size 512)	Execution Time (sec)
2	1.214726
4	0.672029
6	0.7068
8	0.674825
10	0.677104
12	0.695681
14	0.682626
16	0.663857



Information #:

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <stdbool.h>
#include <mpi.h>

// Size of the matrix (NxN)
#define N 512

MPI_Status status;

// Define matrices
int matrix1[N][N];
int matrix2[N][N];
int productMatrix[N][N];

// Counter variables
int i, j, k;

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

```
{
        int numberOfProcessors;
        int processorRank;
        int numberOfWorkers;

        // Processor sending data
        int sourceProcessor;

        // Processor to receive data
        int destinationProcessor;

        // The number of rows for a worker processor to process
        int rows;

        // The subset of a matrix to be processed by workers
        int matrixSubset;

        // Initialize MPI environment
        MPI_Init(&argc, &argv);

        // Determine number of processors available
        MPI_Comm_size(MPI_COMM_WORLD, &numberOfProcessors);

        // Determine rank of calling process
        MPI_Comm_rank(MPI_COMM_WORLD, &processorRank);

        numberOfWorkers = numberOfProcessors - 1;

        double stime,etime;

        /* ---------- Manager Processor Code ---------- */

        if (processorRank == 0)
        {
                // Initialize a timer
                stime = MPI_Wtime();
```

```
		printf("\nMultiplying a %dx%d matrix using %d
processor(s).\n\n", N, N, numberOfProcessors);

		// Populate the matrices with values
		for (i = 0; i < N; i++)
		{
			for (j = 0; j < N; j++)
			{
				matrix1[i][j] = (rand() % 6) + 1;
				matrix2[i][j] = (rand() % 6) + 1;
			}
		}

		/* Send the matrix to the worker processes */
		rows = N / numberOfWorkers;
		matrixSubset = 0;

		// Iterate through all of the workers and assign work
		for (destinationProcessor = 1; destinationProcessor <=
numberOfWorkers; destinationProcessor++)
		{
			// Determine the subset of the matrix to send to the
destination processor
			MPI_Send(&matrixSubset, 1, MPI_INT,
destinationProcessor, 1, MPI_COMM_WORLD);

			// Send the number of rows to process to the
destination worker processor
			MPI_Send(&rows, 1, MPI_INT, destinationProcessor,
1, MPI_COMM_WORLD);

			// Send rows from matrix 1 to destination worker
processor
			MPI_Send(&matrix1[matrixSubset][0], rows * N,
MPI_INT, destinationProcessor, 1, MPI_COMM_WORLD);
```

```
                    // Send entire matrix 2 to destination worker
processor
                    MPI_Send(&matrix2, N * N, MPI_INT,
destinationProcessor, 1, MPI_COMM_WORLD);

                    // Determine the next chunk of data to send to the
next processor
                    matrixSubset = matrixSubset + rows;
            }

            // Retrieve results from all workers processors
            for (i = 1; i <= numberOfWorkers; i++)
            {
                    sourceProcessor = i;
                    MPI_Recv(&matrixSubset, 1, MPI_INT,
sourceProcessor, 2, MPI_COMM_WORLD, &status);
                    MPI_Recv(&rows, 1, MPI_INT, sourceProcessor, 2,
MPI_COMM_WORLD, &status);
                    MPI_Recv(&productMatrix[matrixSubset][0], rows *
N, MPI_INT, sourceProcessor, 2, MPI_COMM_WORLD, &status);
            }

            // Stop the timer
            etime = MPI_Wtime();

            // Determine and print the total run time
            printf("Runtime:  %f seconds\n",etime-stime);
    }

    /* ---------- Worker Processor Code ---------- */

    if (processorRank > 0)
    {
            sourceProcessor = 0;
            MPI_Recv(&matrixSubset, 1, MPI_INT, sourceProcessor, 1,
MPI_COMM_WORLD, &status);
```

```
			MPI_Recv(&rows, 1, MPI_INT, sourceProcessor, 1, 
MPI_COMM_WORLD, &status);
			MPI_Recv(&matrix1, rows * N, MPI_INT, sourceProcessor, 
1, MPI_COMM_WORLD, &status);
			MPI_Recv(&matrix2, N * N, MPI_INT, sourceProcessor, 1, 
MPI_COMM_WORLD, &status);

			/* Perform matrix multiplication */
			for (k = 0; k < N; k++)
			{
				for (i = 0; i < rows; i++)
				{
					productMatrix[i][k] = 0.0;
					for (j = 0; j < N; j++)
					{
						productMatrix[i][k] = productMatrix[i][k] 
+ matrix1[i][j] * matrix2[j][k];
					}
				}
			}

			MPI_Send(&matrixSubset, 1, MPI_INT, 0, 2, 
MPI_COMM_WORLD);
			MPI_Send(&rows, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
			MPI_Send(&productMatrix, rows * N, MPI_INT, 0, 2, 
MPI_COMM_WORLD);
		}

		MPI_Finalize();
}
```

GitHub Link:

https://github.com/Kunalkadam179/HPC-Assignment/tree/main/Assignment%20-%207