**Final Year B. Tech., Sem VII 2022-23**

# High Performance Computing Lab

**PRN: 2019BTECS00029**

**Full Name: Harshal Kodgire**

**Batch: B1**

## Assignment No. 7

---

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 100

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();
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Walchand\7 Semester\HPC\Assignment_7> mpiexec -n 5 .\a7_q1.exe
Running with 5 tasks.
Done in 0.001861 seconds.
PS D:\Walchand\7 Semester\HPC\Assignment_7> mpiexec -n 2 a7_q1.exe
Running with 2 tasks.
Done in 0.000564 seconds.
PS D:\Walchand\7 Semester\HPC\Assignment_7> mpiexec -n 6 a7_q1.exe
Running with 6 tasks.
Done in 0.001977 seconds.
PS D:\Walchand\7 Semester\HPC\Assignment_7> mpiexec -n 8 a7_q1.exe
Running with 8 tasks.
Done in 0.002164 seconds.
PS D:\Walchand\7 Semester\HPC\Assignment_7>
```

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

```
/*
 * There are some simplifications here. The main one is that matrices B and C
 * are fully allocated everywhere, even though only a portion of them is
 * used by each processor (except for processor 0)
 */
#include <mpi.h>
#include <stdio.h>


#define SIZE 4    /* Size of matrices */


int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];


void fill_matrix(int m[SIZE][SIZE])
{
```

```
  static int n=1;
  int i, j;
  for (i=0; i<SIZE; i++)
    for (j=0; j<SIZE; j++)
    m[i][j] = n++;
}


void print_matrix(int m[SIZE][SIZE])
{
  int i, j = 0;
  for (i=0; i<SIZE; i++) {
    printf("\n\t| ");
    for (j=0; j<SIZE; j++)
    printf("%2d ", m[i][j]);
    printf("|");
  }
}

int main(int argc, char *argv[])
{
  int myrank, P, from, to, i, j, k;
  int tag = 666;    /* any value will do */
  MPI_Status status;
```

```
MPI_Init (&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* who am i */

MPI_Comm_size(MPI_COMM_WORLD, &P); /* number of processors */


/* Just to use the simple variants of MPI_Gather and MPI_Scatter we */

/* impose that SIZE is divisible by P. By using the vector versions, */

/* (MPI_Gatherv and MPI_Scatterv) it is easy to drop this restriction. */


if (SIZE%P!=0) {

  if (myrank==0) printf("Matrix size not divisible by number of processors\n");

  MPI_Finalize();

  exit(-1);

}


from = myrank * SIZE/P;

to = (myrank+1) * SIZE/P;


/* Process 0 fills the input matrices and broadcasts them to the rest */

/* (actually, only the relevant stripe of A is sent to each process) */


if (myrank==0) {

  fill_matrix(A);

  fill_matrix(B);
```

```
}

double start = MPI_Wtime();

MPI_Bcast (B, SIZE*SIZE, MPI_INT, 0, MPI_COMM_WORLD);

MPI_Scatter (A[to], SIZE*SIZE/P, MPI_INT, A[from], SIZE*SIZE/P, MPI_INT, 0,
MPI_COMM_WORLD);

printf("computing slice %d (from row %d to %d)\n", myrank, from, to-1);

for (i=from; i<to; i++)

  for (j=0; j<SIZE; j++) {

    C[i][j]=0;

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

    C[i][j] += A[i][k]*B[k][j];

  }

MPI_Gather (C[from], SIZE*SIZE/P, MPI_INT, C[to], SIZE*SIZE/P, MPI_INT, 0,
MPI_COMM_WORLD);

if (myrank==0) {

  double finish = MPI_Wtime();

  // printf("\n\n");

  // print_matrix(A);

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

  // print_matrix(B);
```

```
  // printf("\n\n\t        = \n");

  // print_matrix(C);

  // printf("\n\n");


  printf("Exection Time: %f\n", finish - start);

 }


 MPI_Finalize();

 return 0;

}
```

```
PS D:\Walchand\7 Semester\HPC\Assignment_7> mpiexec -n 2 a7_q2.exe
computing slice 0 (from row 0 to 1)
Exection Time: 0.000438
computing slice 1 (from row 2 to 3)
PS D:\Walchand\7 Semester\HPC\Assignment_7> mpiexec -n 4 a7_q2.exe
computing slice 3 (from row 3 to 3)
computing slice 2 (from row 2 to 2)
computing slice 1 (from row 1 to 1)
computing slice 0 (from row 0 to 0)
Exection Time: 0.000862
PS D:\Walchand\7 Semester\HPC\Assignment_7>
```