Assignment No. 11

PRN: 2020BTECS00025
Name: Sahil Santosh Otari
Course: High Performance Computing Lab
Title of Practical: Implementation of MPI programs.

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

Code:

```
#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);
      }

      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:

For size 1000:

	2	3	4	5	6
100	0.000125	0.000163	0.000337	0.002897	0.000297
1000	0.008710	0.006261	0.004788	0.004569	0.004838