Final Year B. Tech., Sem VII 2023-24

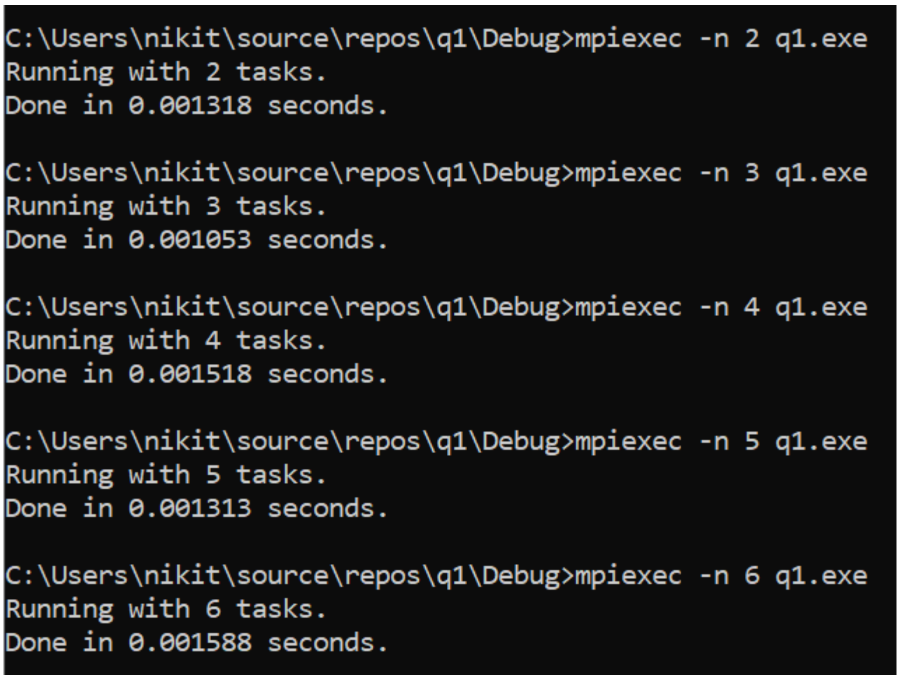
High_Performance_Computing_Lab

Practical No. 11

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

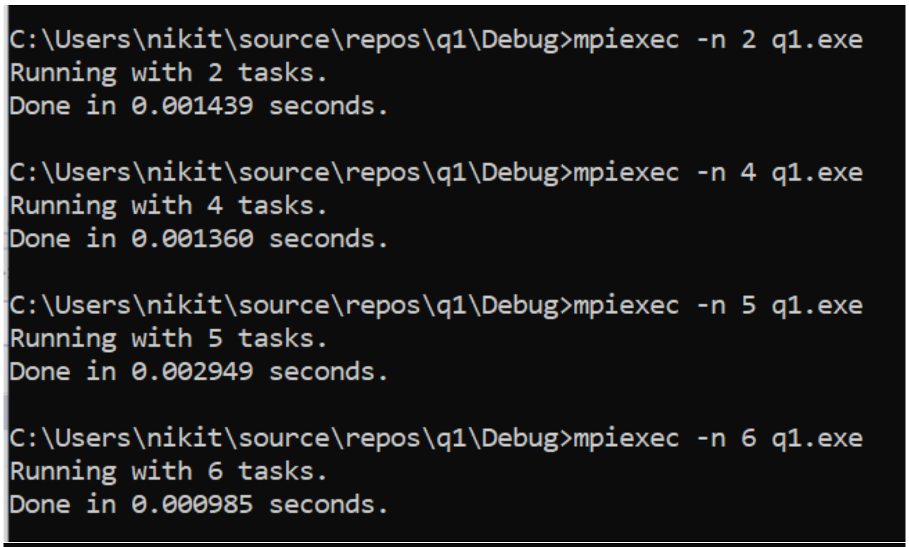
```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 20
int main(int argc, char** argv) {
    int np, rank, numworkers, rows, i = 0, j = 0, k = 0;
    double a[N][N], b[N], c[N];
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    numworkers = np - 1;
    int dest, source;
    int tag;
    int rows_per_process, extra, offset;
    if (rank == 0)
    {
        printf("Running with %d tasks.\n", np);
        for (j = 0; j < N; j++)
            a[i][j] = 1;
        for (i = 0; i < N; i++)
            b[i] = 1;
        double start = MPI_Wtime();
        rows_per_process = N / numworkers;
        extra = N % numworkers;
        offset = 0;
        tag = 1;
        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;
        }
        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);
    }
    if (rank > 0) {
        tag = 1;
        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);
        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];
        }
        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();
}
```

Size: 20

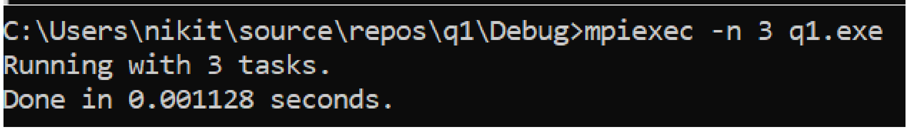


```
C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 2 q1.exe
Running with 2 tasks.
Done in 0.001318 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 3 q1.exe
Running with 3 tasks.
Done in 0.001053 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 4 q1.exe
Running with 4 tasks.
Done in 0.001518 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 5 q1.exe
Running with 5 tasks.
Done in 0.001313 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 6 q1.exe
Running with 6 tasks.
Done in 0.001588 seconds.
```

Size 100

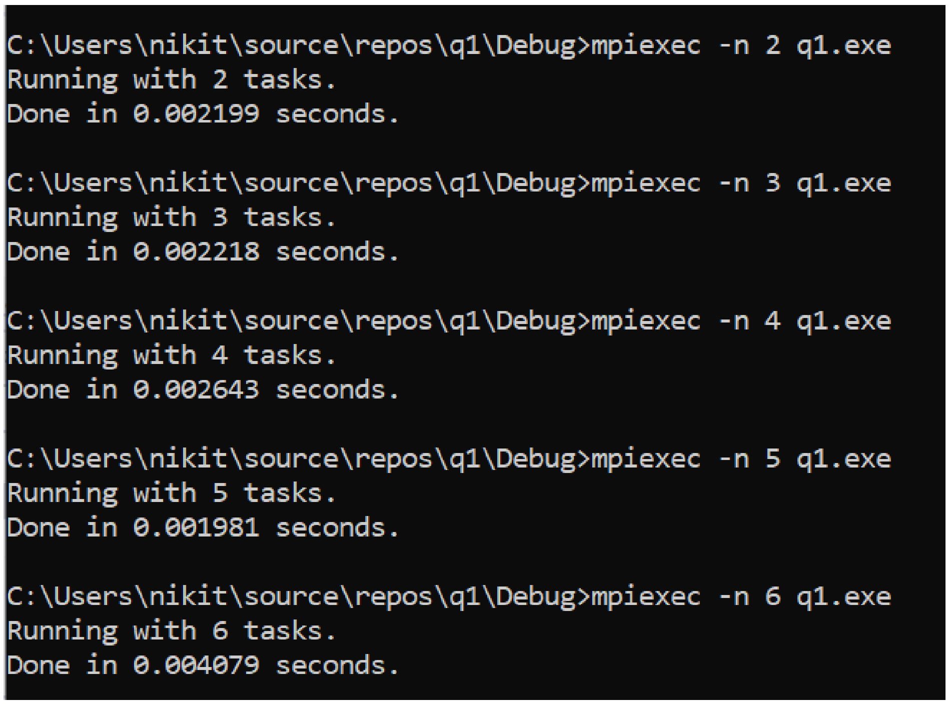


```
C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 2 q1.exe
Running with 2 tasks.
Done in 0.001439 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 4 q1.exe
Running with 4 tasks.
Done in 0.001360 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 5 q1.exe
Running with 5 tasks.
Done in 0.002949 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 6 q1.exe
Running with 6 tasks.
Done in 0.000985 seconds.
```



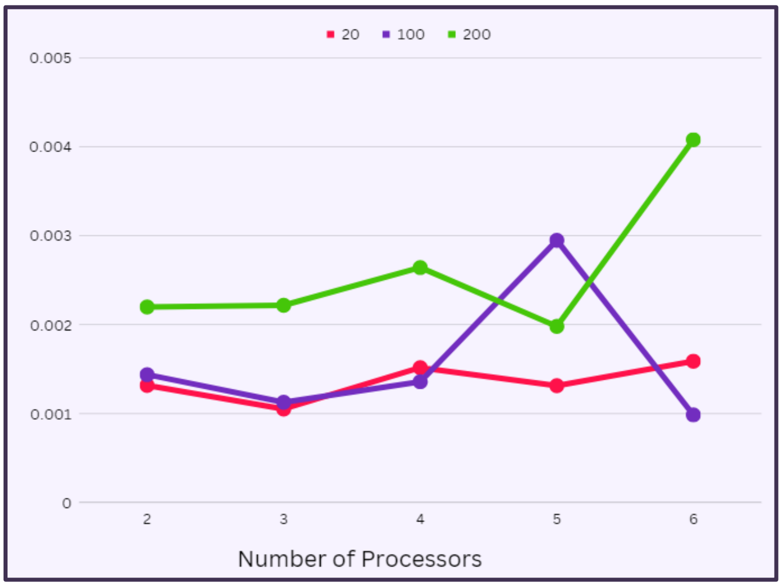
```
C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 3 q1.exe
Running with 3 tasks.
Done in 0.001128 seconds.
```

Size 200



```
C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 2 q1.exe
Running with 2 tasks.
Done in 0.002199 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 3 q1.exe
Running with 3 tasks.
Done in 0.002218 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 4 q1.exe
Running with 4 tasks.
Done in 0.002643 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 5 q1.exe
Running with 5 tasks.
Done in 0.001981 seconds.

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 6 q1.exe
Running with 6 tasks.
Done in 0.004079 seconds.
```

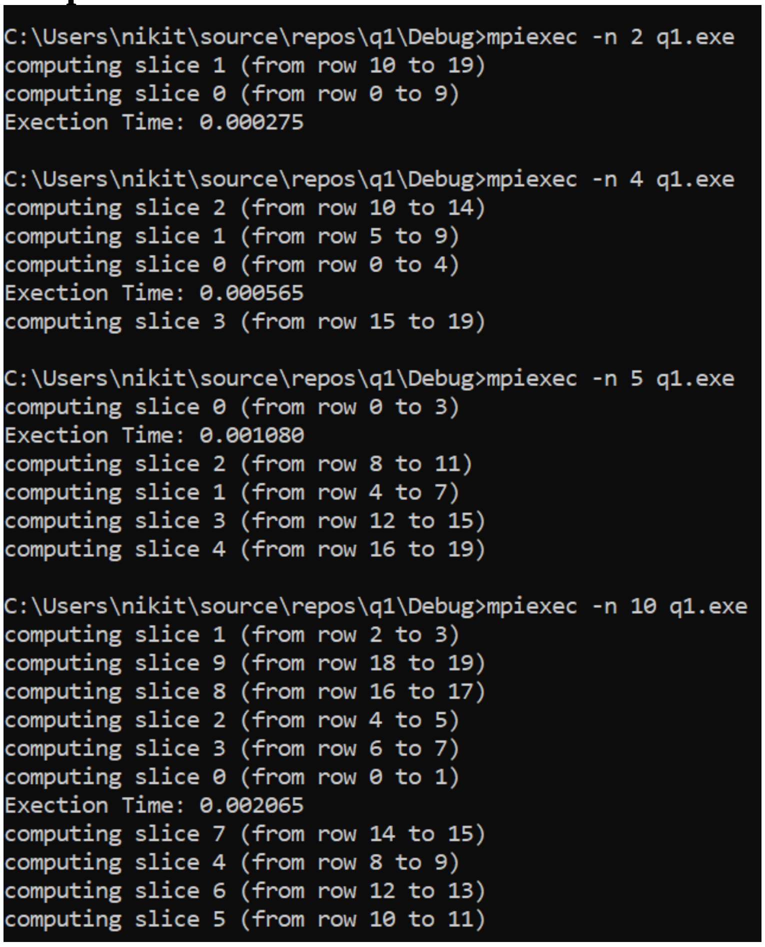
Size\ Processes	2	3	4	5	6
20	0.001318	0.001053	0.001518	0.001313	0.001588
100	0.001439	0.001128	0.001360	0.002949	0.000985
200	0.002199	0.002218	0.002643	0.001981	0.004079



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

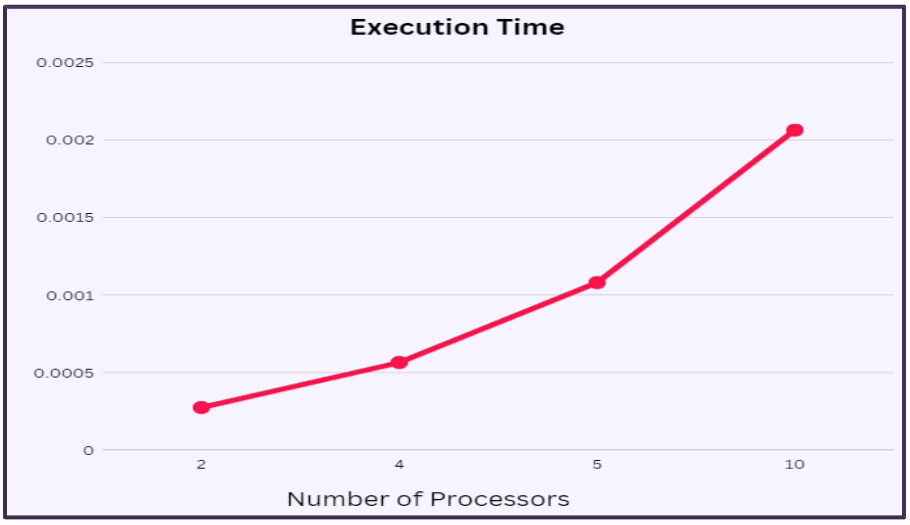
```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20
int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];
void fill_matrix(int m[SIZE][SIZE]) {
    static int n = 1;
    int i, j;
    for (i = 0; i < SIZE; i++)
        for (j = 0; j < SIZE; j++)
            m[i][j] = n++;
}
void print_matrix(int 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;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &P);
    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;
    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("Exection Time: %f\n", finish - start);
    }
    MPI_Finalize();
    return 0;
}
```

Output



```
C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 2 q1.exe
computing slice 1 (from row 10 to 19)
computing slice 0 (from row 0 to 9)
Exection Time: 0.000275

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 4 q1.exe
computing slice 2 (from row 10 to 14)
computing slice 1 (from row 5 to 9)
computing slice 0 (from row 0 to 4)
Exection Time: 0.000565
computing slice 3 (from row 15 to 19)

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 5 q1.exe
computing slice 0 (from row 0 to 3)
Exection Time: 0.001080
computing slice 2 (from row 8 to 11)
computing slice 1 (from row 4 to 7)
computing slice 3 (from row 12 to 15)
computing slice 4 (from row 16 to 19)

C:\Users\nikit\source\repos\q1\Debug>mpiexec -n 10 q1.exe
computing slice 1 (from row 2 to 3)
computing slice 9 (from row 18 to 19)
computing slice 8 (from row 16 to 17)
computing slice 2 (from row 4 to 5)
computing slice 3 (from row 6 to 7)
computing slice 0 (from row 0 to 1)
Exection Time: 0.002065
computing slice 7 (from row 14 to 15)
computing slice 4 (from row 8 to 9)
computing slice 6 (from row 12 to 13)
computing slice 5 (from row 10 to 11)
```

	Processes			
Size	2	4	5	10
20	0.000275	0.000565	0.001080	0.002065


Execution Time
Number of Processors