# High Performance Computing

## ISE-2

Name : Sahil Santosh Otari

PRN : 2020BTECS00025

BATCH : B2

1. Execute the all-reduce operation (Program 3.2.2.c) with varying number of processors (1 to 16) and fixed message size of 10K words. Plot the performance of the operation with varying number of processors (with constant message size). Explain the performance observed.(Question 2 from sheet)

   **Code:**

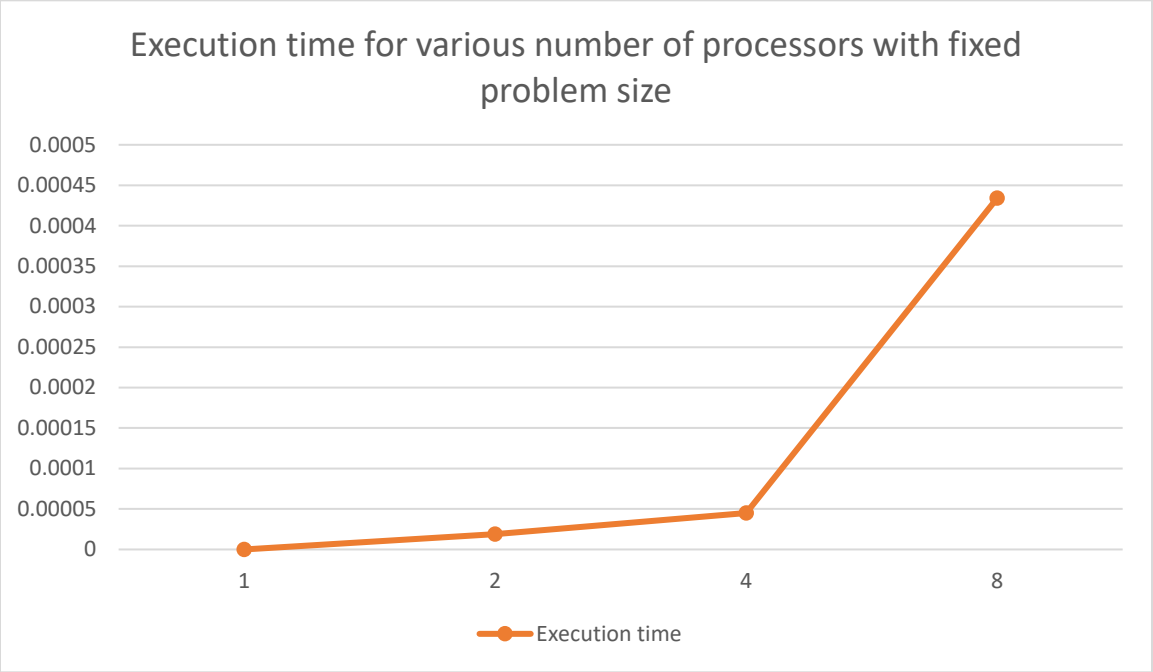
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Usage : allreduce message_size\n");
        return 1;
    }

    int rank;
    int size = atoi(argv[1]);
    char* input_buffer = new char[size];
    char* recv_buffer = new char[size];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int i;
    srand(time(NULL));
    for (i = 0; i < size; i++)
        input_buffer[i] = rand() % 256;
    double total_time = 0.0;
    double start_time = 0.0;
```

```
    for (i = 0; i < 100; i++) {
        MPI_Barrier(MPI_COMM_WORLD);
        start_time = MPI_Wtime();
        MPI_Allreduce(input_buffer, recv_buffer, size, MPI_BYTE,
MPI_BOR, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        total_time += (MPI_Wtime() - start_time);
    }

    if (rank == 0) {
        printf("Average time for allreduce : %f secs\n", total_time /
100);
    }

    MPI_Finalize();
}
```

**Result Analysis for a fixed problem size of 10k:**

| Number of Processors | Execution time |
|---|---|
| 1 | 0.000000 |
| 2 | 0.000019 |
| 4 | 0.000045 |
| 8 | 0.000434 |
| 16 | 0.037894 |

## Charts &Screenshots:




```
C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 1 MatrixVector.exe 10000
Average time for allreduce : 0.000000 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 2 MatrixVector.exe 10000
Average time for allreduce : 0.000019 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 4 MatrixVector.exe 10000
Average time for allreduce : 0.000045 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 8 MatrixVector.exe 10000
Average time for allreduce : 0.000434 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 16 MatrixVector.exe 10000
Average time for allreduce : 0.037894 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>
```

## Observations:

The program initially benefits from parallelism, leading to decreased execution time as the number of processors increases from 1 to 4. However, beyond 4 processors, the execution time increases, suggesting diminishing returns and potential communication overhead. There seems to be an optimal range of processors (between 4 and 8) for the best performance.

2. Consider two implementations of one-to-all broadcast. The first implementation uses the MPI implementation (Program 3.5.1.c). The second implementation splits the message and executes the broadcast in two steps (Program 3.5.1b.c). Plot the runtime of the two implementations with varying number of processors (1, 2, 4, 8) with constant message size 100K. Explain the observed performance of the two implementations.(Question 7 from sheet)

   **Code:**

   **Program 3.5.1.c**

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

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Usage : bcast message_size\n");
        return 1;
    }

    int rank;
    int size = atoi(argv[1]);
    char* buffer = new char[size];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int i;
    if (rank == 0) {
        srand(time(NULL));
        for (i = 0; i < size; i++)
            buffer[i] = rand() % 256;
    }
    double total_time = 0.0;
    double start_time = 0.0;
    for (i = 0; i < 100; i++) {
        MPI_Barrier(MPI_COMM_WORLD);
        start_time = MPI_Wtime();
```

```
        MPI_Bcast(buffer, size, MPI_CHAR, 0, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        total_time += (MPI_Wtime() - start_time);
    }

    if (rank == 0) {
        printf("Average time for broadcast : %f secs\n", total_time /
100);
    }

    MPI_Finalize();
}
```

**Program 3.5.1b.c**

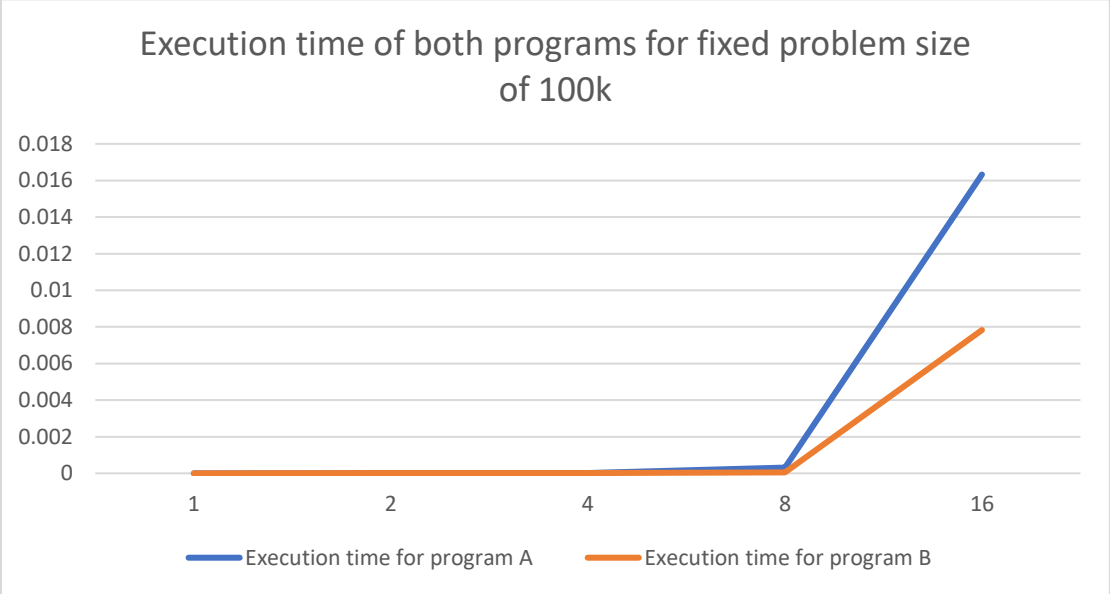
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Usage : bcast message_size\n");
        return 1;
    }

    int rank;
    int size = atoi(argv[1]);
    char* buffer = new char[size];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int i;
    if (rank == 0) {
        srand(time(NULL));
        for (i = 0; i < size; i++)
            buffer[i] = rand() % 256;
    }
```

```
    MPI_Bcast(buffer, size / 2, MPI_CHAR, 0, MPI_COMM_WORLD);

    MPI_Bcast(buffer + size / 2, size / 2, MPI_CHAR, 0,
MPI_COMM_WORLD);

    double total_time = 0.0;
    double start_time = 0.0;
    for (i = 0; i < 100; i++) {
        MPI_Barrier(MPI_COMM_WORLD);
        start_time = MPI_Wtime();

        MPI_Bcast(buffer, size / 2, MPI_CHAR, 0, MPI_COMM_WORLD);

        MPI_Bcast(buffer + size / 2, size / 2, MPI_CHAR, 0,
MPI_COMM_WORLD);

        total_time += (MPI_Wtime() - start_time);
    }

    if (rank == 0) {
        printf("Average time for broadcast (two steps): %f secs\n",
total_time / 100);
    }

    MPI_Finalize();
}
```

**Result Analysis for a fixed problem size of 100k:**

| Number of Processors | Execution time for program A | Execution time for program B |
|---|---|---|
| 1 | 0.000005 | 0.000000 |
| 2 | 0.0000010 | 0.000018 |
| 4 | 0.000024 | 0.000021 |
| 8 | 0.000320 | 0.000058 |
| 16 | 0.016326 | 0.007830 |

## Charts &Screenshots:




## Program 3.5.1.c:

```
C:\Users\Dell>cd C:\Users\Dell\source\repos\MatrixVector\Debug

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 1 MatrixVector.exe 100000
Average time for broadcast : 0.000005 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 2 MatrixVector.exe 100000
Average time for broadcast : 0.000010 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 3 MatrixVector.exe 100000
Average time for broadcast : 0.000015 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 4 MatrixVector.exe 100000
Average time for broadcast : 0.000024 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 8 MatrixVector.exe 100000
Average time for broadcast : 0.000320 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 16 MatrixVector.exe 100000
Average time for broadcast : 0.016326 secs
```

**Program 3.5.1b.c:**

```
C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 1 MatrixVector.exe 100000
Average time for broadcast (two steps): 0.000000 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 2 MatrixVector.exe 100000
Average time for broadcast (two steps): 0.000018 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 4 MatrixVector.exe 100000
Average time for broadcast (two steps): 0.000021 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 8 MatrixVector.exe 100000
Average time for broadcast (two steps): 0.000058 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>mpiexec -n 16 MatrixVector.exe 100000
Average time for broadcast (two steps): 0.007830 secs

C:\Users\Dell\source\repos\MatrixVector\Debug>
```

**Observations:**

The MPI implementation (Program 3.5.1.c) shows good scalability with lower execution times as the number of processors increases. The split message implementation (Program 3.5.1b.c) has lower overhead for a small number of processors but becomes less efficient with larger-scale parallelism. MPI is recommended for better scalability, while the split message approach may be suitable for smaller-scale parallelism.

# System Configuration on which programs are executed