**Class:** Final Year (Computer Science and Engineering)

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

**Course:** High Performance Computing Lab

**Practical No. 11**

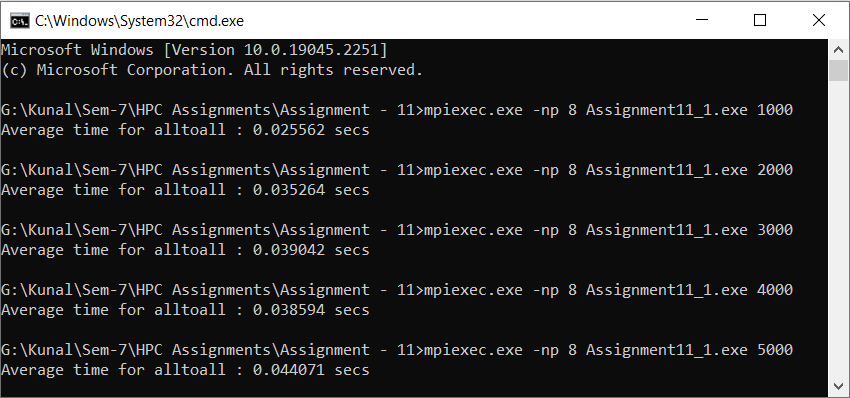
**Exam Seat No: 2019BTECS00064**

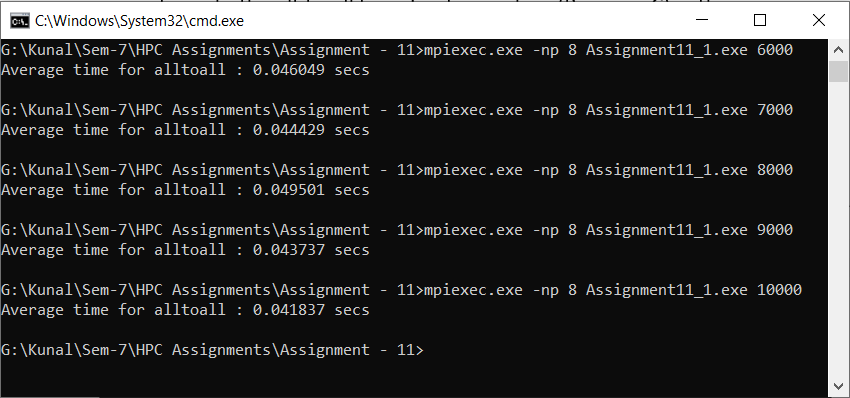
**Name – Kunal Santosh Kadam**

**Problem Statement 1:**

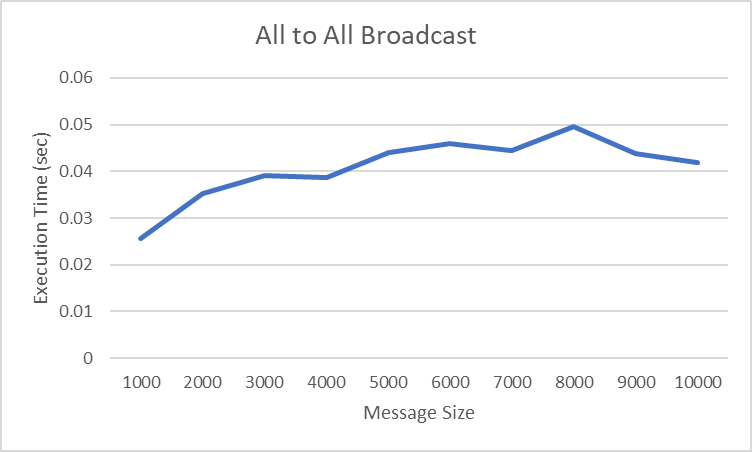
Execute the all-to-all broadcast operation (Program C) with varying message sizes. Plot the performance of the operation with varying message sizes from 1K to 10K (with constant number of processes, 8). Explain the performance observed.

**Screenshot #:**





|  |  |
| --- | --- |
| **Message Size (Processor 8 )** | **Execution Time (sec)** |
| 1000 | 0.025562 |
| 2000 | 0.035264 |
| 3000 | 0.039042 |
| 4000 | 0.038594 |
| 5000 | 0.044071 |
| 6000 | 0.046049 |
| 7000 | 0.044429 |
| 8000 | 0.049501 |
| 9000 | 0.043737 |
| 10000 | 0.041837 |



**Information #:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <mpi.h>

int main(int argc, char \*argv[])

{

if (argc != 2)

{

printf("Usage : alltoall message\_size\n");

return 1;

}

int rank;

int num\_procs;

int size = atoi(argv[1]);

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_procs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

int i;

char input\_buffer[size\*num\_procs];

char recv\_buffer[size\*num\_procs];

srand(time(NULL));

for (i = 0; i < size; i++)

input\_buffer[i] = rand() % 256;

int j;

for (j = 1; j < num\_procs; j++)

{

int k = 0;

for (i = j\*size; i < j\*size + size; i++)

{

input\_buffer[i] = input\_buffer[k];

k++;

}

}

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\_Alltoall(input\_buffer, size, MPI\_CHAR, recv\_buffer, size, MPI\_CHAR, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

total\_time += (MPI\_Wtime() - start\_time);

}

if (rank == 0)

{

printf("Average time for alltoall : %f secs\n", total\_time/100);

}

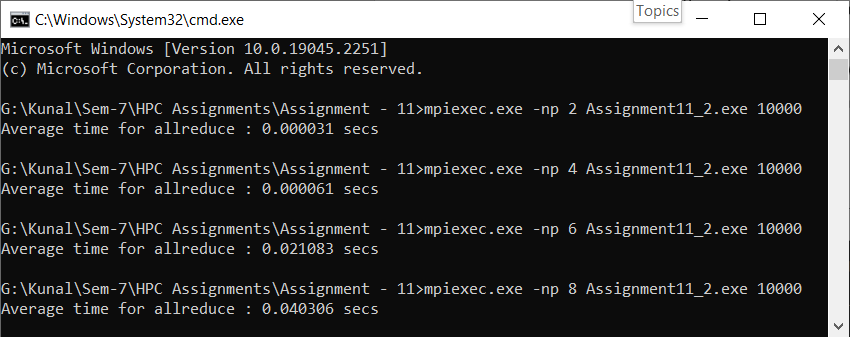
MPI\_Finalize();

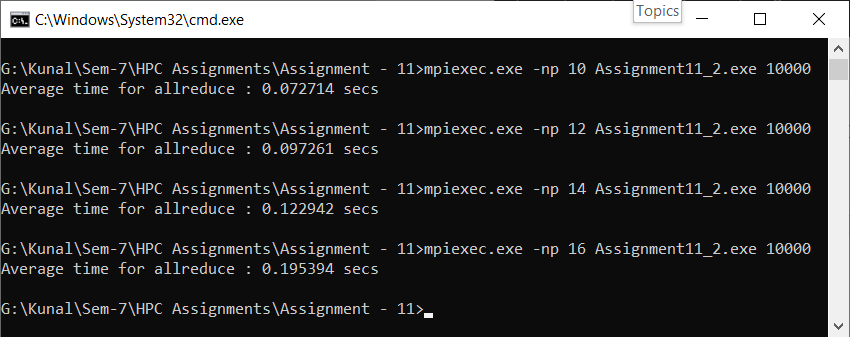
}

**Problem Statement 2:**

Execute the all-reduce operation (Program D) with varying number of processes (1 to 16) and fixed message size of 10K words. Plot the performance of the operation with varying number of processes (with constant message size). Explain the performance observed.

**Screenshot #:**





|  |  |
| --- | --- |
| **Processors (Message Size 10000)** | **Execution Time (sec)** |
| 2 | 0.000031 |
| 4 | 0.000061 |
| 6 | 0.021083 |
| 8 | 0.040306 |
| 10 | 0.072714 |
| 12 | 0.097261 |
| 14 | 0.122942 |
| 16 | 0.195394 |

**Information #:**

#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[size];

char recv\_buffer[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();

}