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

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

**Course:** High Performance Computing Lab

**Practical No. 11**

**Exam Seat No:**

2019BTECS00051-Atharv Inamdar

**Title of practical:**

Analysis of MPI Programs

**Problem Statement 1:**

Execute the MPI program (Program A) with a fixed size broadcast. Plot the performance of the broadcast with varying numbers of processes (with constant message size). Explain the performance observed.

**Screenshot 1:**

#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[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)

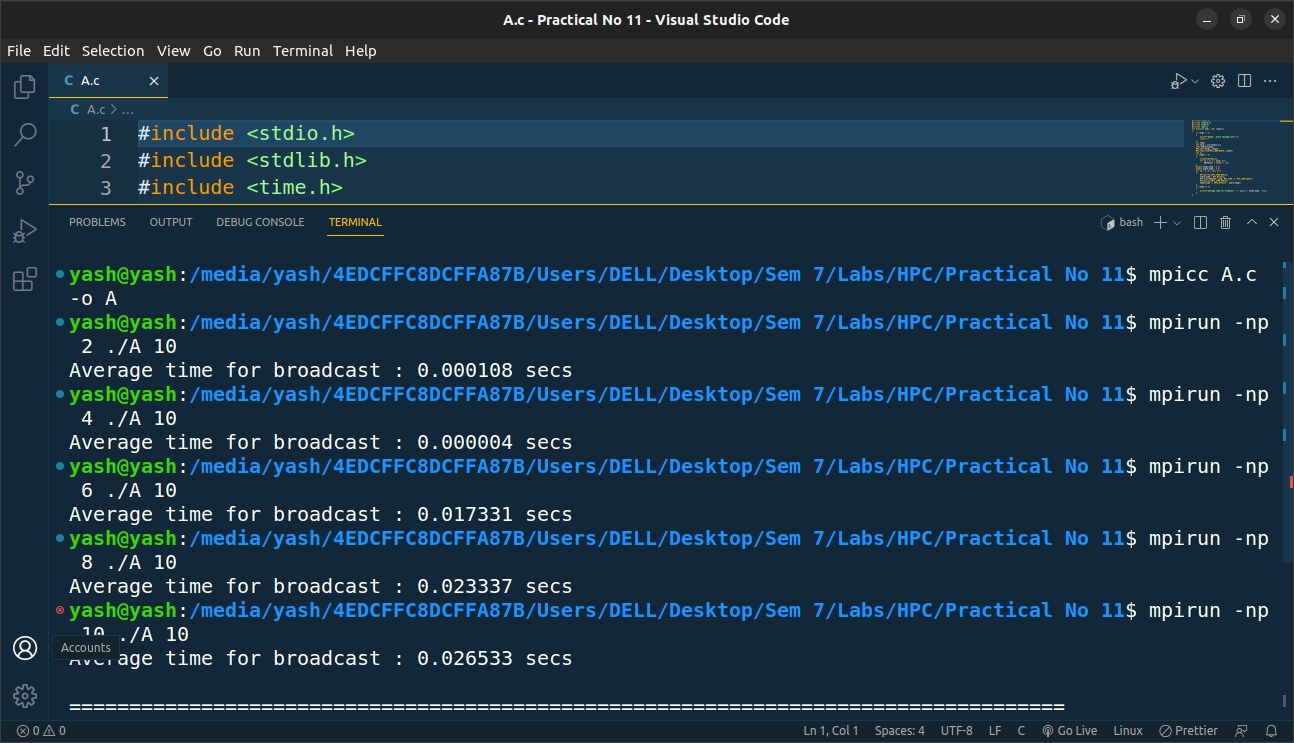
    {

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

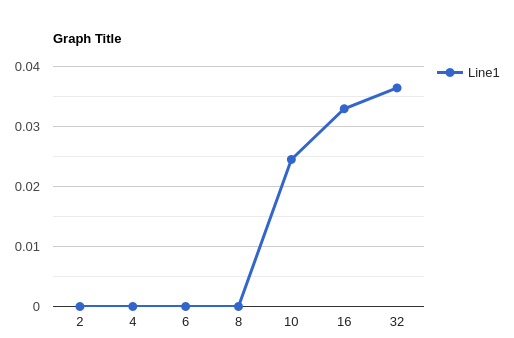
    }

}

**Screenshot 2:**

****

**Screenshot 3:**

****

**Problem Statement 2:**

Repeat problem 2 above with varying message sizes for reduction (Program B). Explain the observed performance of the reduction operation.

**Screenshot 4:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <mpi.h>

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

{

    if (argc != 2)

    {

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

        return 1;

    }

    int rank;

    int size = atoi(argv[1]);

    char input\_buffer[size];

    char output\_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\_Reduce(input\_buffer, output\_buffer, size, MPI\_BYTE, MPI\_BOR, 0, MPI\_COMM\_WORLD);

        MPI\_Barrier(MPI\_COMM\_WORLD);

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

    }

    if (rank == 0)

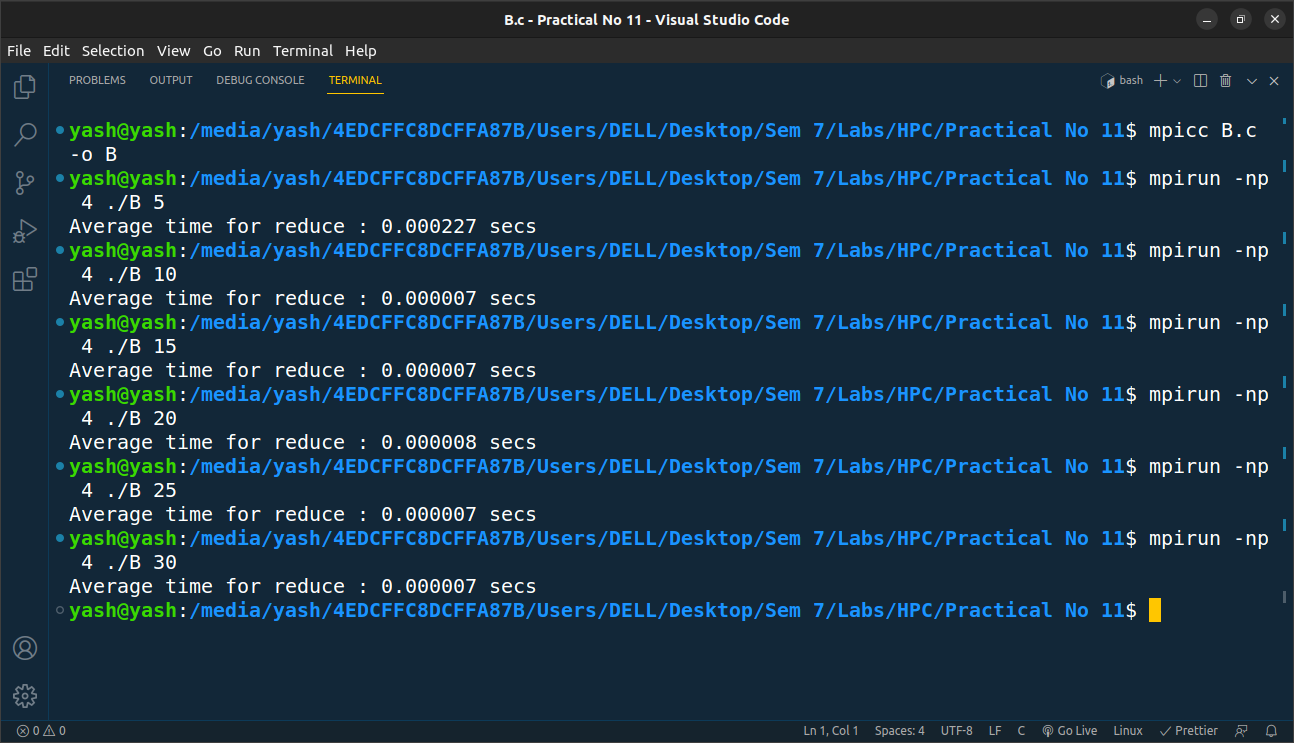
    {

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

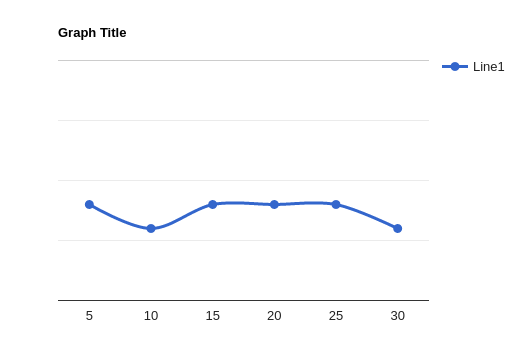
    }

}

**Screenshot 5:**

****

**Screenshot 6:**

****