Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2024-25 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 10

Exam Seat No :

Full Name :

Title of practical : Analysis of MPI Programs

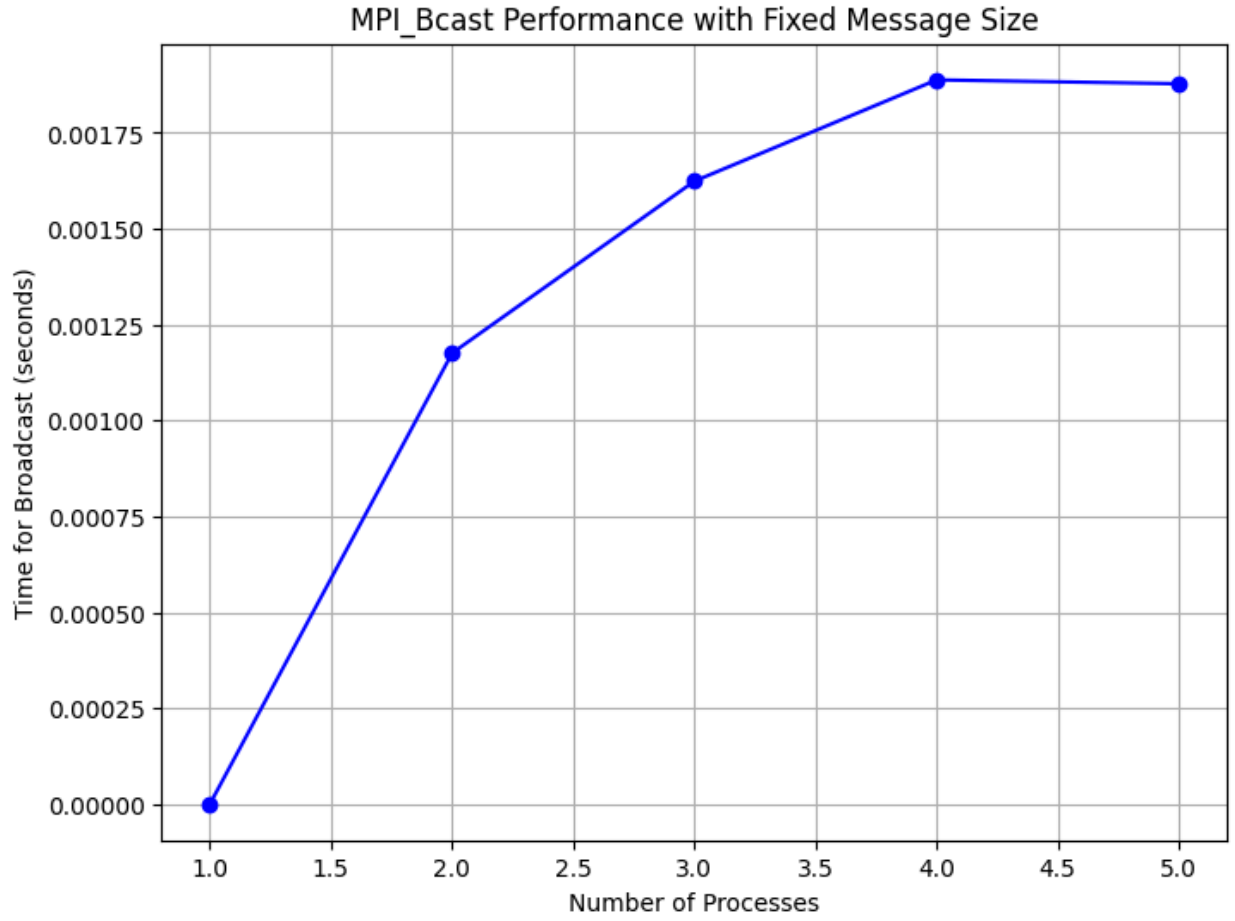
Problem Statement 1:
Execute the MPI program (Program A) with a fixed size broadcast. Plot the performance of the broadcast with varying numbers of processes (with constant messagesize). Explain the performance observed.

Code:

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

#define MSG_SIZE 1024

int main(int argc, char *argv[]) {
   int rank, size;
   int message[MSG_SIZE];
   double start_time, end_time;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   if (rank == 0) {
       for (int i = 0; i < MSG_SIZE; i++) {
           message[i] = i;
       }
```

```
        printf("Root process (Rank %d) broadcasting
message...\n", rank);
    }

    start_time = MPI_Wtime();
    MPI_Bcast(message, MSG_SIZE, MPI_INT, 0, MPI_COMM_WORLD);
    end_time = MPI_Wtime();

    printf("Process %d received message. First value: %d\n",
rank, message[0]);

    if (rank == 0) {
        printf("Number of processes: %d, Time taken for
broadcast: %f seconds\n", size, end_time - start_time);
    }

    MPI_Finalize();
    return 0;
}
```

Analysis:



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

Code:

```c
#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>



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

   int rank, size;
```

```
    int *message, *reduced_result;
    int message_size;
    double start_time, end_time;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    for (message_size = 256; message_size <= 4096; message_size
*= 2) {
        message = (int*) malloc(message_size * sizeof(int));
        reduced_result = (int*) malloc(message_size *
sizeof(int));

        for (int i = 0; i < message_size; i++) {
            message[i] = rank + i;
        }

        start_time = MPI_Wtime();
        MPI_Reduce(message, reduced_result, message_size,
MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        end_time = MPI_Wtime();

        if (rank == 0) {
```

```
            printf("Message size: %d, Number of processes: %d,
Time taken for reduction: %f seconds\n", message_size, size,
end_time - start_time);

            printf("Reduction result: First element = %d, Last
element = %d\n", reduced_result[0],
reduced_result[message_size-1]);

        }


        free(message);

        free(reduced_result);

    }


    MPI_Finalize();

    return 0;

}
```

Analysis:

```
*[main][~/acad/hpc-lab/as10]$ mpirun --oversubscribe -np 4 ./2
Message size: 256, Number of processes: 4, Time taken for reduction: 0.000014 seconds
Reduction result: First element = 6, Last element = 1026
Message size: 512, Number of processes: 4, Time taken for reduction: 0.000004 seconds
Reduction result: First element = 6, Last element = 2050
Message size: 1024, Number of processes: 4, Time taken for reduction: 0.000030 seconds
Reduction result: First element = 6, Last element = 4098
Message size: 2048, Number of processes: 4, Time taken for reduction: 0.000017 seconds
Reduction result: First element = 6, Last element = 8194
Message size: 4096, Number of processes: 4, Time taken for reduction: 0.000032 seconds
Reduction result: First element = 6, Last element = 16386
```