**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);**

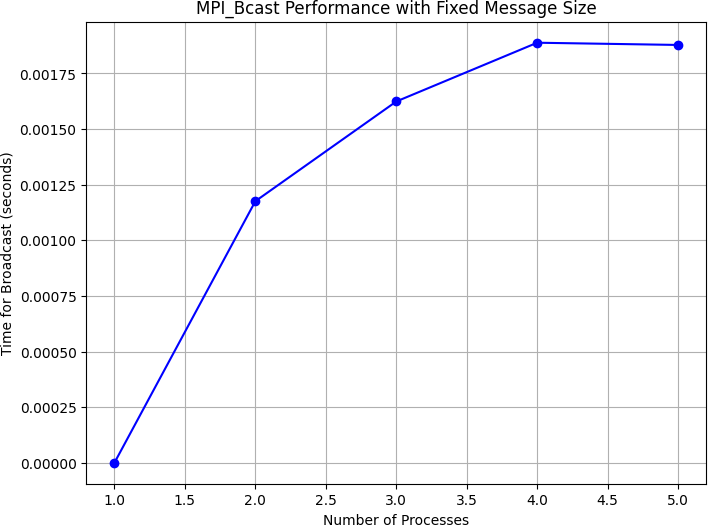
**}**

**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:

*#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);

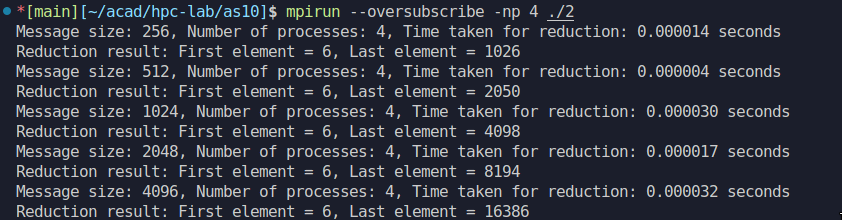
free(reduced\_result);

}

MPI\_Finalize();

*return* 0;

}

**Analysis:**