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 : 21510042**

**Full Name :** Omkar Rajesh Auti

**Title of practical :** Analysis of MPI Programs

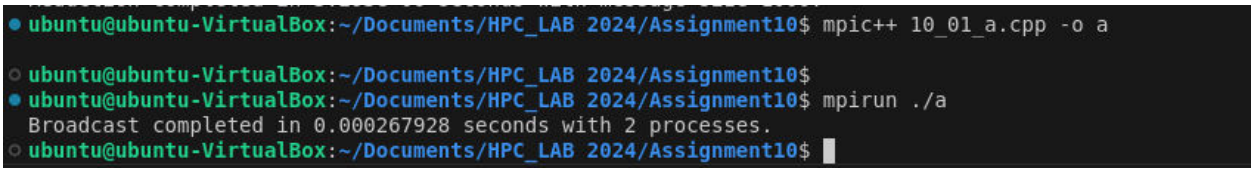
**Problem Statement 1:**
Execute the MPI program (Program A) with a fixed size broadcast. Plot the performance of the broadcast with varying numbers of processes (with constant messagesize). Explain the performance observed.

Code:

```
#include <mpi.h>
#include <iostream>
#include <vector>

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

int world_size, world_rank;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

const int message_size = 1000; // Fixed size
std::vector<int> message(message_size);

if (world_rank == 0) {
// Initialize message in root process
for (int i = 0; i < message_size; i++) {
message[i] = i;
}
}

double start_time = MPI_Wtime();
// Broadcast the message from root (process 0) to all other processes
MPI_Bcast(message.data(), message_size, MPI_INT, 0, MPI_COMM_WORLD);
```

```
double end_time = MPI_Wtime();

if (world_rank == 0) {
std::cout << "Broadcast completed in " << end_time - start_time << "
seconds with " << world_size << " processes.\n";
}

MPI_Finalize();
return 0;
}
```

**Screenshot:**




**Analysis:**

For a fixed message size, as the number of processes increases, the time for broadcasting the message might increase due to the overhead of sending the message to more processes.

However, the increase should be relatively small for a small number of processes. With larger numbers of processes, communication overhead and network congestion may become more noticeable.

**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 <iostream>
#include <vector>

int main(int argc, char *argv[]) {
MPI_Init(&argc, &argv);
```

```
int world_size, world_rank;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

// Varying message size
const int message_size = (world_rank + 1) * 1000; // Message size varies
with rank
std::vector<int> message(message_size, world_rank);

std::vector<int> result(message_size, 0);

double start_time = MPI_Wtime();
// Perform reduction: sum the values across all processes
MPI_Reduce(message.data(), result.data(), message_size, MPI_INT, MPI_SUM,
0, MPI_COMM_WORLD);
double end_time = MPI_Wtime();

if (world_rank == 0) {
std::cout << "Reduction completed in " << end_time - start_time << "
seconds with message size " << message_size << ".\n";
}

MPI_Finalize();
return 0;
}
```

**Screenshot:**

```
ubuntu@ubuntu-VirtualBox:~/Documents/HPC_LAB 2024/Assignment10$ mpirun -np 1 ./a
Reduction completed in 2.18e-06 seconds with message size 1000.
ubuntu@ubuntu-VirtualBox:~/Documents/HPC_LAB 2024/Assignment10$
```

**Analysis:**

For small message sizes, the reduction operation is expected to be fast.

As the message size increases, the time taken for the reduction operation will increase linearly due to the higher communication costs and the amount of data that needs to be transferred and summed across processes.

The overall network bandwidth and latency of the system will also impact the reduction performance.