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

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

**Course:** High Performance Computing Lab

**Practical No. 10**

**Exam Seat No : 21510073**

**Full Name** **:** Sakshi Dhamapurkar

**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.

**Screenshot:**

**#include <stdio.h>**

#include <math.h>

#include "mpi.h"

#define N 8

void transpose(double \*data, int n) {

int i, j;

double tmp;

for (i = 0; i < n; i++) {

for (j = i + 1; j < n; j++) {

tmp = data[i \* n + j];

data[i \* n + j] = data[j \* n + i];

data[j \* n + i] = tmp;

}

}

}

void four1(double \*data, int nn, int isign) {

int n, mmax, m, j, istep, i;

double wtemp, wr, wpr, wpi, wi, theta;

double tempr, tempi;

n = nn << 1;

j = 1;

for (i = 1; i < n; i += 2) {

if (j > i) {

tempr = data[j - 1];

data[j - 1] = data[i - 1];

data[i - 1] = tempr;

tempr = data[j];

data[j] = data[i];

data[i] = tempr;

}

m = n >> 1;

while (m >= 2 && j > m) {

j -= m;

m >>= 1;

}

j += m;

}

mmax = 2;

while (n > mmax) {

istep = mmax << 1;

theta = isign \* (6.28318530717959 / mmax);

wtemp = sin(0.5 \* theta);

wpr = -2.0 \* wtemp \* wtemp;

wpi = sin(theta);

wr = 1.0;

wi = 0.0;

for (m = 1; m < mmax; m += 2) {

for (i = m; i <= n; i += istep) {

j = i + mmax;

tempr = wr \* data[j - 1] - wi \* data[j];

tempi = wr \* data[j] + wi \* data[j - 1];

data[j - 1] = data[i - 1] - tempr;

data[j] = data[i] - tempi;

data[i - 1] += tempr;

data[i] += tempi;

}

wr = (wtemp = wr) \* wpr - wi \* wpi + wr;

wi = wi \* wpr + wtemp \* wpi + wi;

}

mmax = istep;

}

}

int main(int argc, char \*\*argv) {

int rank, numnodes, namelen, i, j, rc;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

double dataR[N \* N], dataI[N \* N];

MPI\_Status status;

rc = MPI\_Init(&argc, &argv);

if (rc != MPI\_SUCCESS) {

MPI\_Abort(MPI\_COMM\_WORLD, rc);

}

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numnodes);

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

MPI\_Get\_processor\_name(processor\_name, &namelen);

if (rank == 0) {

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

dataR[i \* N + j] = i \* N + j;

dataI[i \* N + j] = 0.0;

}

}

}

MPI\_Bcast(dataR, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(dataI, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double start\_time = MPI\_Wtime();

for (i = rank; i < N; i += numnodes) {

four1(&dataR[i \* N], N, 1);

four1(&dataI[i \* N], N, 1);

}

MPI\_Gather(&dataR[rank \* N], N \* N / numnodes, MPI\_DOUBLE, dataR, N \* N / numnodes, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Gather(&dataI[rank \* N], N \* N / numnodes, MPI\_DOUBLE, dataI, N \* N / numnodes, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

transpose(dataR, N);

transpose(dataI, N);

}

MPI\_Bcast(dataR, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(dataI, N \* N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

for (i = rank; i < N; i += numnodes) {

four1(&dataR[i \* N], N, 1);

four1(&dataI[i \* N], N, 1);

}

MPI\_Gather(&dataR[rank \* N], N \* N / numnodes, MPI\_DOUBLE, dataR, N \* N / numnodes, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Gather(&dataI[rank \* N], N \* N / numnodes, MPI\_DOUBLE, dataI, N \* N / numnodes, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double end\_time = MPI\_Wtime();

if (rank == 0) {

printf("Result after 2D FFT (Real part):\n");

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

printf("%2.0f ", dataR[i \* N + j]);

}

printf("\n");

}

printf("Execution time with %d processes: %f seconds\n", numnodes, end\_time - start\_time);

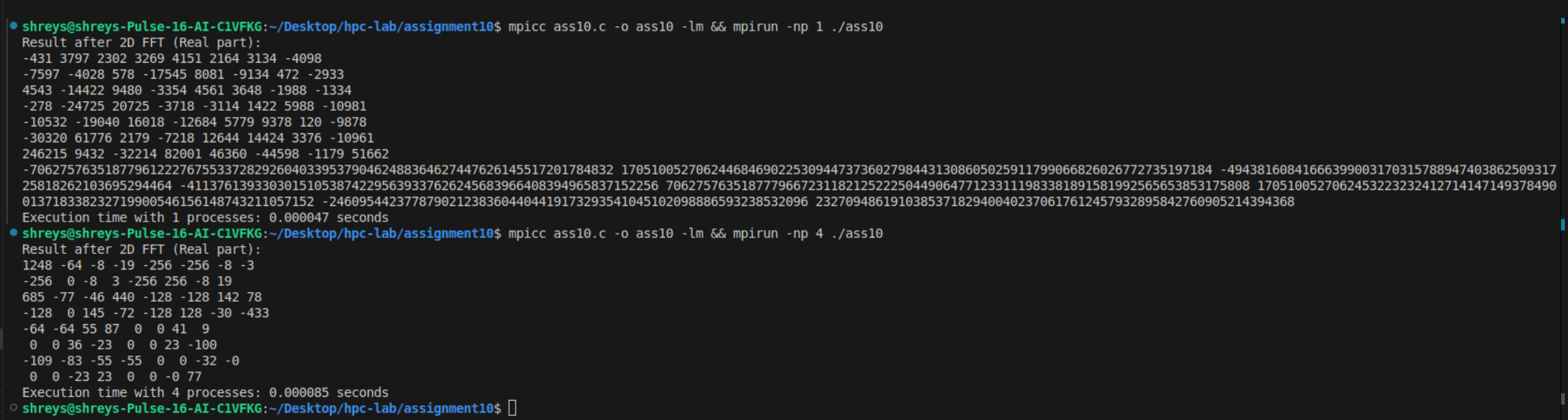
}

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.

**Screenshot:**

#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);

}

MPI\_Finalize();

}

**Analysis:**

