A picture containing text, clipart, vector graphics

Description automatically generated

**CSC580**

**PARALLEL PROCESSING**

**INDIVIDUAL ASSIGNMENT**

**PREPARED FOR:**

HASLIZATUL FAIRUZ BINTI MOHAMED HANUM

**GROUP:**

CS2305C

**PREPARED BY:**

AZRI MUSTAQIM BIN MOHD ROSLAN

(2020620724)

**Task 1:**

Describe the four variations of processor architecture as described by Flynn.

a. Draw appropriate diagrams to support your answer.

b. Explain the components and functions of each architecture.

c. Describe with an example the advantages of each architecture.

**Task 2:**

Describe the phases to transform an algorithm so that it can be executed in parallel machine.

a. Identify one search or sort algorithm and describe the instructions.

b. Draw diagram(s) to show the steps to transform the algorithm.

c. Discuss the performance of the parallel algorithm.

d. Discuss two issues that may affect the performance of the algorithm.

**Task 3:**

The objective of this exercise is to demonstrate the collective communication routines

using MPI.COMM\_WORLD.Scatter() and MPI.COMM\_WORLD.Gather() methods.

The following MPI program is using the scattering and gathering techniques to perform the

following tasks :

• Generate an array of n numbers (generated randomly) on the root process (process 0).

• Scatter the numbers to p processes, given each process an equal amount of numbers.

• Each process computes the TOTAL of their subset of the numbers.

• Gather all the TOTAL to the root process.

• The root process then computes the TOTAL of these numbers.

1. Compile and run the following MPI program. Observe and discuss the program and the output from the program.

Coding:

import java.util.\*;

import mpi.\*;

public class MPISumArray {

public static void main(String args[]) throws Exception

{

MPI.Init(args);

int rank= MPI.COMM\_WORLD.Rank(); **// store the rank#**

int size = MPI.COMM\_WORLD.Size(); **//# of processes ie. -np 4**

int master =0;

int z=1;

int unitSize=25; **// the length for each unit**

int inputSize= unitSize\*size; **// the length of input**

double sendbuf[]= new double[inputSize]; **// send buffer**

double recvbuf[]=new double[unitSize]; **// receive buffer**

double sum[]= new double[unitSize]; **// array to store sum**

double global\_sum= 0;

long startTime=0; **// start execution time**

long elapsedTime=0; **// elapsed time**

if(rank==master)

{

startTime=System.nanoTime(); **// measure start time**

**//generate a random array of numbers**

Random random = new Random();

System.out.println("Random Number produced by Process :"+ master);

for(int i=0; i<sendbuf.length; i++)

{

**// it will generate random number from zero to 100**

sendbuf[i]= random.nextInt(100);

System.out.print((i+1)+") "+sendbuf[i]+"\t");

if(i==(sendbuf.length-1))

{

System.out.print(".");

System.out.println("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

break;

}

if(z==5) **// format output, 5 data in a line**

{

System.out.println(" ");

z=0;

}

z++;

}

}

**//distribute number to all processes to get summation of numbers**

MPI.COMM\_WORLD.Scatter(sendbuf, 0, unitSize, MPI.DOUBLE, recvbuf, 0, unitSize, MPI.DOUBLE,

master);

for(int j=0;j<unitSize;j++)

{

sum[0]+=recvbuf[j]; **// calculate total**

}

for(int x=0;x<size;x++)

{

if(rank==x)

{

System.out.println("\n\n");

**//to display the numbers that the processes receive**

System.out.println("\n\nProcess "+rank+ " received : \t");

for(int j=0;j<recvbuf.length;j++)

{

System.out.print(rank+":"+recvbuf[j]);

if(j==(recvbuf.length)-1)

{

System.out.print(". ");

break;

}

else

System.out.print(", ");

}

System.out.println("\nsum for rank "+rank+": "+sum[0]);

break;

}

}

**// to gather the sum each process to a master**

MPI.COMM\_WORLD.Gather(sum, 0, 1, MPI.DOUBLE , sendbuf, 0, 1, MPI.DOUBLE, master);

if(rank==master)

{

for(int i=0;i<size;i++)

global\_sum+=sendbuf[i];

System.out.println("Total sum of all produced numbers = "+global\_sum);

}

if(rank==master)

{

elapsedTime=System.nanoTime()-startTime; **// measure time**

System.out.println("Total Execution Time : "+(elapsedTime/1000000.0)+"

milliseconds ");

}

MPI.Finalize();

}

}

Output :

Graphical user interface, application

Description automatically generated with medium confidence

1. Produce a serial program to calculate and display the total and average of n (range of 100 -500) numbers in an array of data (integer or double).

Coding:

import java.util.\*;

public class SerialProgramming {

public static void main(String[] args) {

int inputSize = 100;

double[] n = new double[inputSize];

double sum = 0;

double average = 0;

long startTime, elapsedTime;

int z = 1;

startTime = System.nanoTime();

Random random = new Random();

System.out.println("Random Number produced by Process: ");

for(int i=0; i<n.length; i++){

n[i]=random.nextInt(100);

System.out.print((i+1)+")"+n[i]+"\t\t");

if(i==(n.length-1)){

System.out.println();

System.out.println("---------------------------------------------------------------------------");

break;

}

if(z == 5){

System.out.println();

z = 0;

}

z++;

}

for(int i=0; i<n.length; i++){

sum = sum + n[i];

}

average = sum/n.length;

System.out.println();

System.out.println("Sum: "+sum);

System.out.println("Average: "+average);

elapsedTime = System.nanoTime()-startTime;

System.out.println("Total execution time: "+(elapsedTime/1000000.0)+" milliseconds");

}

}

Output:

Table

Description automatically generated

1. Modify the parallel program which uses the scattering and gathering

techniques (refer to slides) to perform the following tasks given below:

• Generate an array of n numbers on the root process (process 0).

(Note: specify the value of n in the range of 100 – 50000)

• Scatter the numbers to p processes, given each process an equal amount

of numbers.

(Note: specify the value of p in the range of 4 - 10)

• Each process computes the AVERAGE of their subset of the numbers.

• Gather all the AVERAGES to the root process.

• The root process then computes the AVERAGE of these numbers to get

the final average.

1. Measure, record and analyze the execution time for both the serial and parallel programs. Discuss the finding from this measurement.

Coding:

package mpisumarray;

import java.util.\*;

import mpi.\*;

public class ParallelProgramming {

public static void main(String[] args) throws Exception {

MPI.Init(args);

int rank = MPI.COMM\_WORLD.Rank();

int size = MPI.COMM\_WORLD.Size();

int master = 0;

int z = 1;

int inputSize = 100;

int unitSize = inputSize/size;

double sendbuf[] = new double[inputSize];

double recvbuf[] = new double[unitSize];

double sum[] = new double[unitSize];

double average[] = new double[unitSize];

double global\_sum = 0;

double global\_average = 0;

long startTime = 0;

long elapsedTime = 0;

if(rank==master)

{

startTime = System.nanoTime();

Random random = new Random();

System.out.println("Random Number produced by Process: " +

master);

for(int i=0; i<sendbuf.length; i++){

sendbuf[i] = random.nextInt(100);

System.out.print((i+1)+")"+sendbuf[i]+"\t\t");

if(i==(sendbuf.length-1))

{

System.out.println();

System.out.println("---------------------------------------------------------------------------");

break;

}

if(z==5){

System.out.println();

z=0;

}

z++;

}

}

**//distribute number to all processes to get summation of numbers**

MPI.COMM\_WORLD.Scatter(sendbuf, 0, unitSize, MPI.DOUBLE,

recvbuf, 0, unitSize, MPI.DOUBLE, master);

for(int j=0; j<unitSize; j++)

{

sum[0]+=recvbuf[j**]; // calculate total**

}

average[0] = sum[0];**//inputSize**

for(int x=0; x<size; x++)

{

if(rank==x)

{

System.out.println("\n\nProcess "+rank+" received :");

for(int j=0; j<recvbuf.length; j++)

{

System.out.print(rank+":"+recvbuf[j]);

if(j==(recvbuf.length)-1)

{

System.out.println(". ");

break;

}

else

System.out.print(", ");

}

System.out.println("\nSum for rank "+rank+": "+sum[0]);

System.out.println("\nAverage for rank "+rank+": "+average[0]);

break;

}

}

**// to gather the sum each process to a master**

MPI.COMM\_WORLD.Gather(sum, 0, 1, MPI.DOUBLE, sendbuf, 0, 1,

MPI.DOUBLE, master);

if(rank == master){

for(int i=0; i<size; i++)

global\_sum+=sendbuf[i];

System.out.println("Total sum of all produced numbers = "+global\_sum);

}

**// to gather the average each process to a master**

MPI.COMM\_WORLD.Gather(average, 0, 1, MPI.DOUBLE, sendbuf, 0, 1,

MPI.DOUBLE, master);

if(rank==master){

for(int i=0; i<size; i++)

global\_average+=sendbuf[i];

System.out.println("Total average of all produced numbers = "+global\_average);

}

if(rank==master){

elapsedTime = System.nanoTime()-startTime;

System.out.println("Total Execution Time: "+(elapsedTime/1000000.0)+" milliseconds");

}

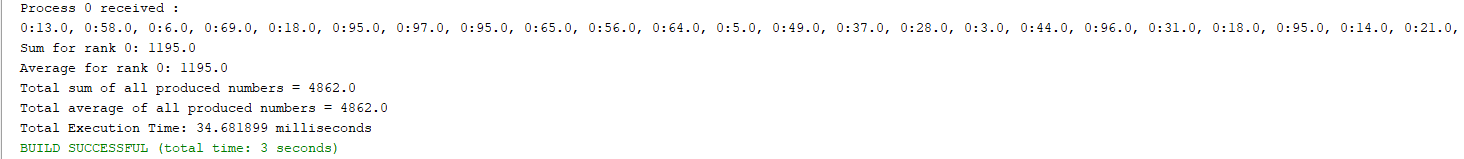
MPI.Finalize();

}

}

Output:

Graphical user interface, table

Description automatically generated  


**Execution Analysis :**

Based on the 3 tests for each programming:

|  |  |
| --- | --- |
| **Parallel Programming** | **Serial Programming** |
| A screenshot of a computer  Description automatically generated with medium confidence | A picture containing graphical user interface  Description automatically generated |
| Text  Description automatically generated | Graphical user interface, text  Description automatically generated with medium confidence |
| A screenshot of a computer  Description automatically generated with medium confidence | A picture containing text  Description automatically generated |

* Execution time taken by serial programming is less than parallel programming and it shows that serial programming is faster than parallel programming.
* It is because serial programming makes less communication that lead to less processing compared to parallel programming, thus make the execution time shorter.