CSC580 Parallel Processing
Assignment 1 (Individual)

Complete the following tasks.

Submit all answers in a .pdf file, named as <yourname_group>

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 Scatter and Gather

Consider the addition of 100 numbers on a distributed memory 4-processor computer. For simplicity of coding, we sum the first one hundred positive integers and compute $S = \sum_{i=1}^{100} i$.

A parallel algorithm to sum 100 numbers proceeds in four stages: (1) distribute 100 numbers evenly among the 4 processors; (2) Every processor sums 25 numbers; (3) Collect the 4 sums to the manager node; and (4) Add the 4 sums and print the result.

Scattering an array of 100 number over 4 processors and gathering the partial sums at the 4 processors to the root is displayed in Figure 1.

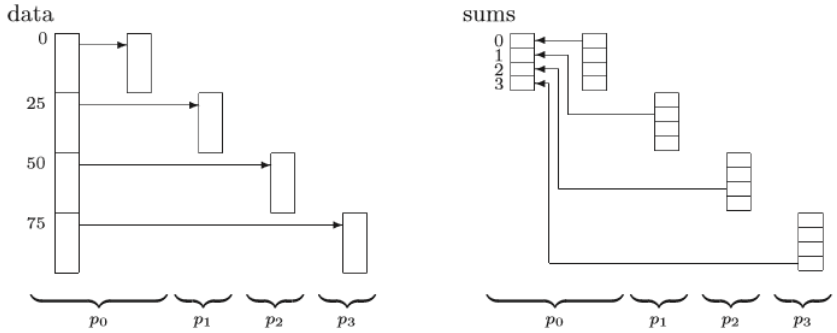

Figure 1: Scattering data and gathering results.

a. Compile and run the following MPI program. Observe and discuss the program and the output from the program.
b. 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).

c. 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.

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

e. Results to be submitted which consists of:
   - Source code (with comments/ descriptions)
   - Screenshots of sample input & output (with explanation)

```
/*
 * This MPI program is using scattering and gathering techniques
 * to calculate and display the total of 100 numbers in an array
 * @ Modified by Zulaile Mabni
 */

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();
    }

}
```