/\*

**Assignment 2**

Design a distributed application using MPI to carry out the calculation in parallel (i.e. by dividing the work up evenly among P processors) which counts the number of primes between 1 and N.

\*/

**// mpiprogram.java**

import mpi.MPI;

public class mpiprogram {

public static void main(String args[]) {

//Initialize and finalize

MPI.Init(args); //takes a reference to the command line arguments

int root = 0;

//Rank and Size

int size = MPI.COMM\_WORLD.Size(); //Whenever program is ran in mpi all processes are grouped in an communicator.

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

if(rank == root)

{

System.out.println("Size of MPI Communicator : "+size);

}

System.out.println("\nThis process has rank "+rank);

// Initializing the send buffer

int sendbuf[] = new int[size];

if(rank == root) {

sendbuf[0] = 10;

sendbuf[1] = 20;

sendbuf[2] = 30;

sendbuf[3] = 40;

System.out.println("Data to be scattered by process "+rank+": ");

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

{

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

}

}

int recvbuf[] = new int[1]; //Declaring receive buffer

//scatter (it sends chunk of array to different processes)

MPI.COMM\_WORLD.Scatter(sendbuf, 0, 1, MPI.INT, //(Send buffer, Start of the data in buffer, chunk size,datatype)

recvbuf, 0, 1, MPI.INT,

root);

//double the data

System.out.println("Process "+rank+" has data: "+recvbuf[0]);

recvbuf[0] = recvbuf[0]\*2;

//gather(it takes each process and gathers them to the root process)

MPI.COMM\_WORLD.Gather(recvbuf, 0, 1, MPI.INT,

sendbuf, 0, 1, MPI.INT,

root);

//display the doubled data

if(rank == root) {

System.out.println("The root process "+rank +" has data: ");

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

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

}

MPI.Finalize();

}

}

**// Prime.java**

import mpi.\*;

import java.util.ArrayList;

public class Prime{

final static int numbers = 20; // End point should be N

static int size, position; // Total number of size

final static ArrayList<Integer> primes = new ArrayList<>();;

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

MPI.Init(args);

position = MPI.COMM\_WORLD.Rank(); // Position [0 / 1]

size = MPI.COMM\_WORLD.Size(); // size [2]

int begin = position \* (numbers / size);

int end = begin + (numbers / size);

ArrayList<Integer> temporaryStorage = new ArrayList<>();

for(int i = begin; i < end; i++){

if(isNumberPrime(i)){

temporaryStorage.add(i);

}

}

// Send back the length the storage currently is

if(position > 0){

int[] sendBuf = new int[]{temporaryStorage.size()};

MPI.COMM\_WORLD.Send(sendBuf, 0, 1, MPI.INT, 0, 1);

System.out.println("SendBuf Completed");

for(int i = 0; i < temporaryStorage.size(); i++){

int[] send = new int[]{temporaryStorage.get(i)};

MPI.COMM\_WORLD.Send(send, 0, 1, MPI.INT, 0, 0);

System.out.println("Send ["+i+"] : " + temporaryStorage.get(i));

}

} else {

for(int i = 0; i < temporaryStorage.size(); i++){

primes.add(temporaryStorage.get(i));

}

int[] recvBuf = new int[1];

for (int src = 1; src < size; src++) {

MPI.COMM\_WORLD.Recv(recvBuf, 0, 1, MPI.INT, src, 1);

System.out.println("recvBuf Completed");

for (int ind = 1; ind <= recvBuf[0]; ind++) {

int[] recv = new int[1];

MPI.COMM\_WORLD.Recv(recv, 0, 1, MPI.INT, src, 0);

System.out.println("Recv ["+ind+"] : " + recv[0]);

primes.add(recv[0]);

}

}

}

MPI.Finalize();

}

// checking prime or not

public static boolean isNumberPrime(int number) {

if(number <= 1) return false;

for(int i = 2; i <= Math.sqrt(number); ++i)

{

if(number % i == 0) return false;

}

return true;

}

}

**// OUTPUT**

Text, letter

Description automatically generated