TERM PROJECT REPORT

ON

Parallel and Distributed Programming (CMPE523)

TOPIC

"IMPLENTATION OF ODD-EVEN PARALLEL PREFIX PROBLEM"

SUBMITTED BY

BAHRAM LAVI SEFIDGARI 125799
AMIR NARIMANI 135144
ZAHRA TAGHIZADEH ABAS ABAD 135164

TO

ASSOC. PROF. Dr. ALEXANDER.G. CHEFRANOV

DEPARTMENT OF COMPUTER ENGINEERING,
FACULTY OF ENGINEERING

EASTERN MEDITERRENEAN UNIVERSITY, NORTHERN CYPRUS

Spring 2014

OUTLINE

INTRODUCTION

TASK FORMULATION

PROPOSE STRUCTURE AND ALGORITHM

BRIEF DISCRIPTION ON THE SYSTEM

TEST OF PROGRAM AND RESULTS

SUMMARY AND CONCOLUSION

REFERENCES

INTRODUCTION

Parallel computing is the simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads. Ideally, parallel processing makes programs run faster because there are more engines (CPUs or cores) running it. In practice, it is often difficult to divide a program in such a way that separate CPUs or cores can execute different portions without interfering with each other. Most computers have just one CPU, but some models have several, and multi-core processor chips are becoming the norm. There are even computers with thousands of CPUs.

With single-CPU, single-core computers, it is possible to perform parallel processing by connecting the computers in a network. However, this type of parallel processing requires very sophisticated software called distributed processing software.

Note that parallelism differs from concurrency. Concurrency is a term used in the operating systems and databases communities which refers to the property of a system in which multiple tasks remain logically active and make progress at the same time by interleaving the execution order of the

tasks and thereby creating an illusion of simultaneously executing instructions.[1][2] Parallelism, on the other hand, is a term typically used by the supercomputing community to describe executions that physically execute simultaneously with the goal of solving a problem in less time or solving a larger problem in the same time. Parallelism exploits concurrency.[1]

Parallel processing is also called parallel computing. In the quest of cheaper computing alternatives parallel processing provides a viable option. The idle time of processor cycles across network can be used effectively by sophisticated distributed computing software. The term parallel processing is used to represent a large class of techniques which are used to provide simultaneous data processing tasks for the purpose of increasing the computational speed of a computer system.

The aim of parallelism could be occurred by implementing an algorithm. An algorithm is a sequence of steps designed to solve a problem. For traditional serial algorithms, the steps run one at a time in a well-defined order. Not surprisingly, when concurrent and parallel algorithms are considered, things get a bit more complicated.

Concurrency in an algorithm implies that instead of a single sequence of steps. These steps are interleaved in different ways depending on how the tasks are scheduled for execution. This means the order of memory access operations will vary between runs of a program. If those memory access operations mix reads and writes to the same location, results can vary from one run of a program to the next [3].

In the case, at this project work, the Odd-Even prefix algorithm is implemented in parallelism between up to 10 computers. In the section we will explain and discuss our implantation.

TASK FORMULATION

As said earlier, in this term project we will be implementing a parallelism algorithm of Odd-Even prefix with 10 distributed clients as processor and one server in the network. We shall be using C# to create the interface for interacting with other computers in network while Socket programming will be engaged in our implementation.

PROPOSED STRUCTURE AND ALGORITHM

The figure below shows the proposed structure on which we are going to design and implement of Odd-Even parallel prefix in distributed systems. Also, figure two shows the proposed algorithm of this work.

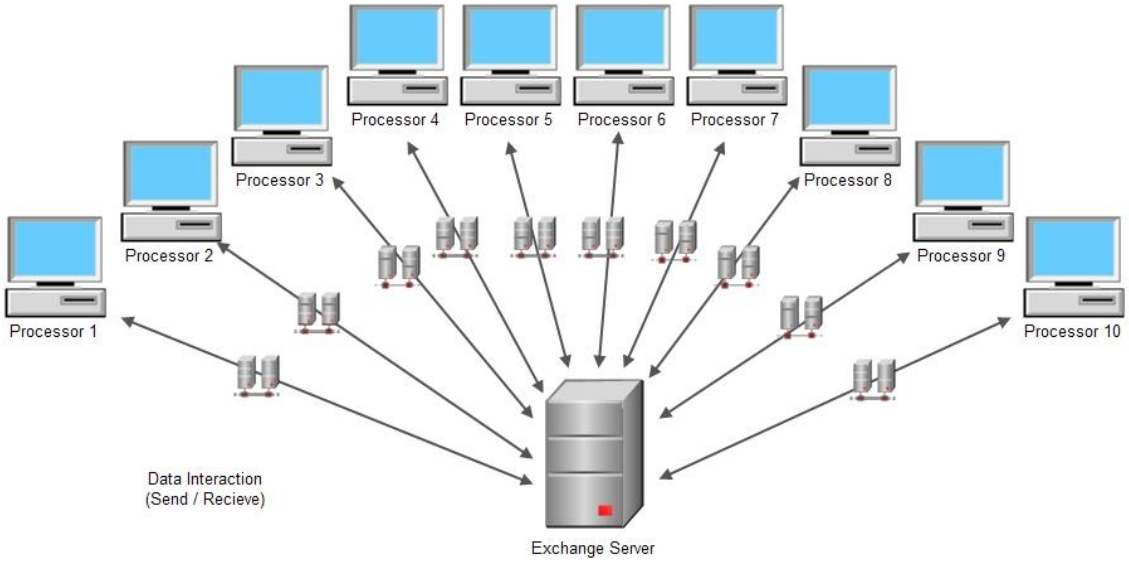

Figure 1 whole structure of proposed method

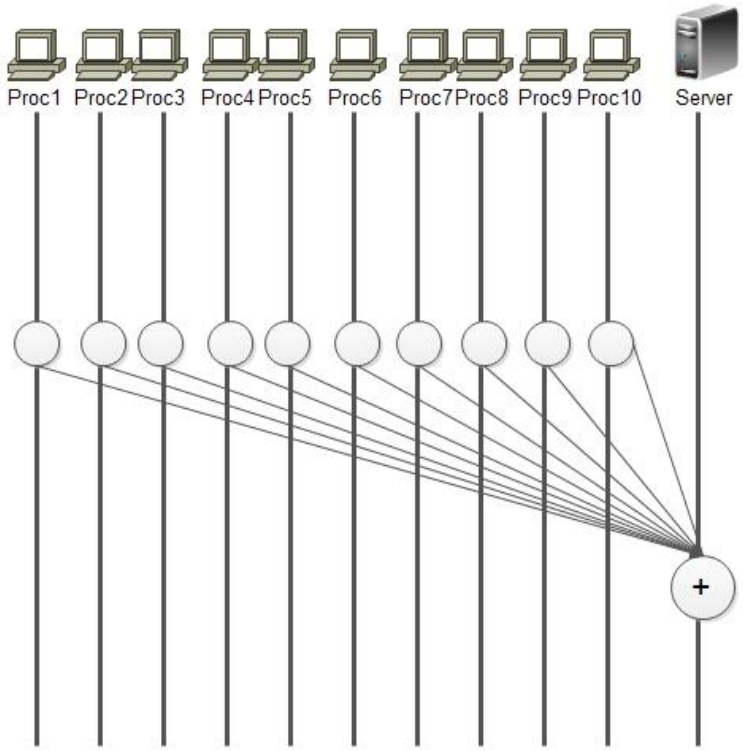

Figure 2 Proposed algorithm

BRIEF DESCRIPTION ON THE SYSTEM

The system consist of a server side computer, 10 client side computers as processor, and two C# applications which are employed in the server and clients. The computers are interacted with Socket programming techniques to have a secure transaction between server and clients. In the case of Odd-Even prefix parallelism, we will like to point out that server separate a big number which is impossible or heavy in calculation cost to calculate in single computer, and by employing the processor of other clients, distribute the separated number to others.

The server side program, firstly, implement socket programming for providing distribution systems. Then, we define a function which is run for each of clients as thread. In the other word, the function still alive until transaction is terminated.

After distribution of separated number to client, server side is waited to get back from the clients. After gathering data, the server side is going to addition all of the data and print last result of algorithm. Below we give C# code which is implementing for the server side of our work.

```
//Initializing the listener.
        static TcpListener tcpListener = new TcpListener(13000);
        static long last_result = 0;
        //The function which is executed for each of the Clients. This Function
is run as Thread.
        static void Listeners()
        {

            Socket socketForClient = tcpListener.AcceptSocket();
            if (socketForClient.Connected)
            {
                Console.WriteLine("Client:" + socketForClient.RemoteEndPoint + "
now connected to server.");
                NetworkStream networkStream = new NetworkStream(socketForClient);
                System.IO.StreamWriter streamWriter =
                new System.IO.StreamWriter(networkStream);
                System.IO.StreamReader streamReader =
                new System.IO.StreamReader(networkStream);
                string theString;
                //here we send message to client
                streamWriter.WriteLine(start_index.ToString() + "," +
end_index.ToString());
                streamWriter.Flush();

                long tmp = start_index;
                start_index = end_index+1;
                end_index += (end_index-tmp)+1;
                //Console.WriteLine("new s:"+start_index.ToString()+"  new
end:"+end_index.ToString());

                while (true)
                {
```

```
                    theString = streamReader.ReadLine();
                    Console.WriteLine("Message recieved by client:" + theString);
                    if (theString == "exit")
                        break;
                    else
                    {
                        last_result += Convert.ToInt64(theString);
                        //Console.WriteLine("lllast=" + last_result.ToString());
                    }

                }

                socketForClient.Close();
                streamReader.Close();
                networkStream.Close();
                streamWriter.Close();

            }

        }

        static long n_sep_pc;

        static long start_index = 0, end_index, tmp;
        // Main Function
        public static void Main(string[] args)
        {
            int servPort = 13000;
            const int Num_PCs = 10; // the number of client

            //user insert a big number for calculation in parallelism.
            Console.WriteLine("Please enter N : ");
            long N = Int64.Parse(Console.ReadLine());
            Console.WriteLine("-------------------------------------------------
");

            if (N % 2 == 0)
                n_sep_pc = N / Num_PCs;
            else
                n_sep_pc = Convert.ToInt64((Math.Floor(Convert.ToDecimal(N /
Num_PCs))));

            start_index = 1;
            end_index = n_sep_pc;

             try
            {

                tcpListener.Start();
                 // this loop will be executing for the number of alive clients.
                for (int i = 0; i < Num_PCs; i++)
                {
                    try
                    {
                        // Create new thread for new client and pendding the
Listeners on it.
                        Thread newThread = new Thread(new
ThreadStart(Listeners));
```

```
                    newThread.Start();

                    System.Threading.Thread.Sleep(500);
                    tmp = start_index;
                    start_index = end_index + 1;
                    end_index += (end_index - tmp) + 1;

                }
                catch (SocketException ex)
                {
                    Console.WriteLine(String.Format("Unable to register
Process {0}. Error:{1}", i, ex.Message));
                    Console.ReadLine();
                }
            }

Console.WriteLine("================================================================
===");
            Console.WriteLine("Last Result is: " + last_result.ToString());

            Console.WriteLine("Press any key to exit from server program");
            Console.ReadKey();

        }
        catch (SocketException ex)
        {
            Console.WriteLine(ex.ErrorCode + " : " + ex.Message);
        }

        Console.ReadLine();
    }
```

The clients receive the separated number from server and calculate Odd-Even prefix by joining parallelism and feedback last result to the server. Below we give C# code which is implementing for the each of client of our work.

```
static IList<long> V = new List<long>();
    static int i;

    // The method used for calculating Odd-Even Prefix Parallelism
    static long ParallelOddEvenPrefix(long NumberStart, long NumberEnd)
    {

        int level = 0;
        int p = 1;
```

```
            int NP2 = 0;
            int NP = 10;
            long Number = NumberEnd - NumberStart + 1;
            long j = NumberStart;
            try
            {
                for (long i = 0; i <= Number; i++)
                    V.Add(j++);
            }
            catch (Exception ex)
            {
                throw;
            }
            try
            {
                level = 2;
                NP2 = NP;

                while (level <= Number)
                {
                    for (int i = level + ((p - 1) * level); i <= Number; i = i +
(level))
                    {
                        V[i] = Convert.ToInt64(V[i]) + Convert.ToInt64(V[i -
(level / 2)]);
                        if (p <= NP2) p++;
                    }
                    //Wait()
                    level = 2 * level;
                    p = 1;
                    NP2 = NP2 / 2;
                }

                level = level / 2;
                p = 1;
                NP2 = NP;
                if (level == Number) { level = level / 2; }

                while (level > 1)
                {
                    for (int i = level + (level / 2) + (p - 1) * level; i <=
Number; i = i + (level))
                    {
                        V[i] = Convert.ToInt64(V[i]) + Convert.ToInt64(V[i -
(level / 2)]);
                        if (p <= NP) p++;
                    }
                    //wait();
                    level = level / 2;
                    p = 1;
                    NP2 = NP2 / 2;

                }

            }
            catch (Exception ex)
            {
                Console.WriteLine("Error ::" + ex.Message);
```

```
            }
            Console.WriteLine(V[V.Count - 1].ToString());
            return V[V.Count - 1];
        }

        //Intializing socket for communication with Server.
        static TcpClient socketForServer;
        static bool conncet=false;
        //Main Method
        static void Main(string[] args)
        {

            while (!conncet)
            {
                try
                {
                    // Define server IP and Port
                    socketForServer = new TcpClient("192.168.181.241", 13000);
                    conncet = true;
                    Console.Clear();
                    break;
                }
                catch
                {
                    conncet = false;
                    Console.WriteLine(
                    "Failed to connect to server at {0}:999", "localhost");


                }
            }

            NetworkStream networkStream = socketForServer.GetStream();
            System.IO.StreamReader streamReader =
            new System.IO.StreamReader(networkStream);
            System.IO.StreamWriter streamWriter =
            new System.IO.StreamWriter(networkStream);
            Console.WriteLine("*******This is client program who is connected to
localhost on port No:13000");

            string[] rec_dev;

            long result_seq = 0;



                try
                {
                    string outputString=null;
            // read the data from the server and display it

                    outputString = streamReader.ReadLine();

                    Console.WriteLine("Message Recieved by server:" +
outputString);
                    string[] recArray = outputString.Split(',');
                    result_seq = ParallelOddEvenPrefix(Int64.Parse(recArray[0]),
Int64.Parse(recArray[1]));
```

```
                // write the data to server
                streamWriter.WriteLine(result_seq.ToString());
                streamWriter.Flush();
                streamWriter.WriteLine("exit");
                streamWriter.Flush();


            }
            catch (Exception ex) { throw; }

            networkStream.Close();
            Console.WriteLine("Press any key to exit from client program");
            Console.ReadKey();

    }
```

TEST OF PROGRAM AND RESULTS

Here we present a test of our system. We defined 10 computers as processor in the network by joining a server. Out experimental result is tested in Computer Laboratory of Department of Computer Engineering and we get the good result from our implementation and proposed method for a big number. Below we will show the figures of our test for a big number while it is 100000000.

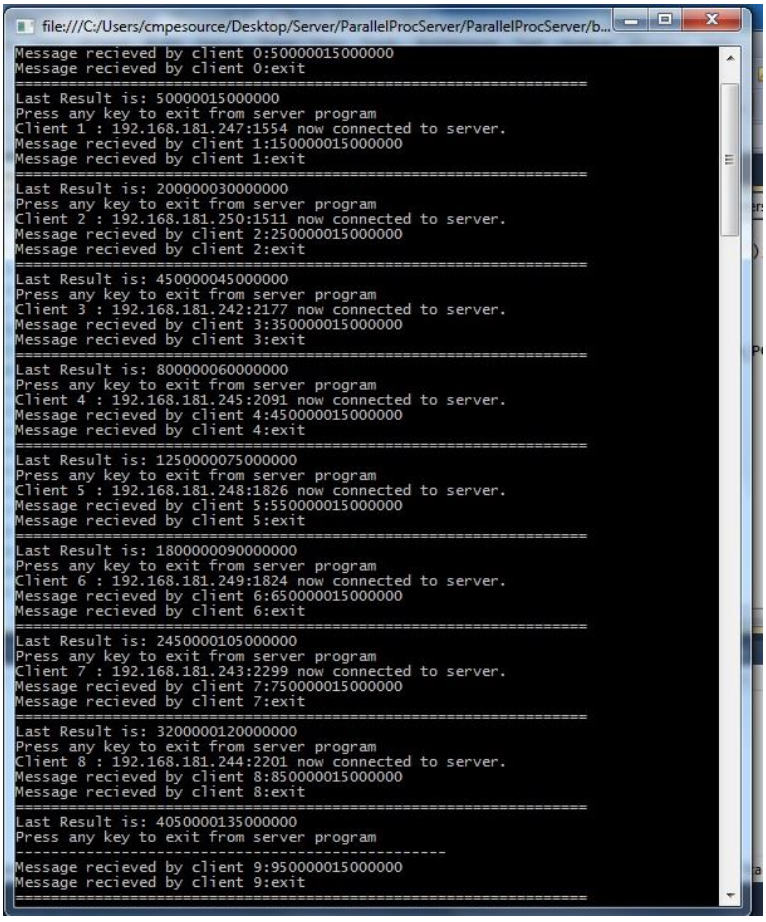

Figure 3 Result of server

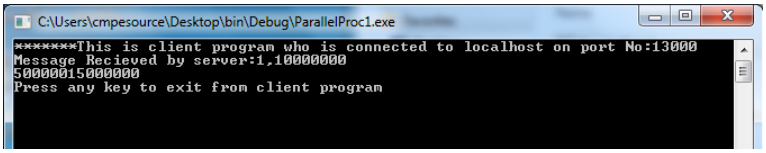

Figure 4 Result of Processor 1

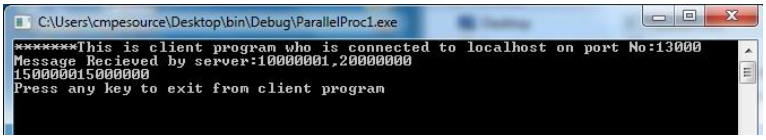

Figure 5 Result of Processor 2

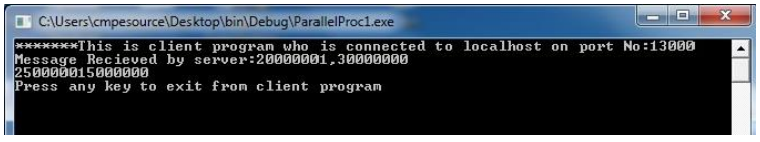

Figure 6 Result of Processor 3

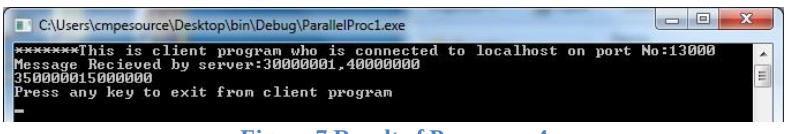

Figure 7 Result of Processor 4

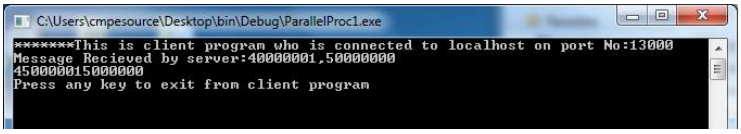

Figure 8 Result of Processor 5

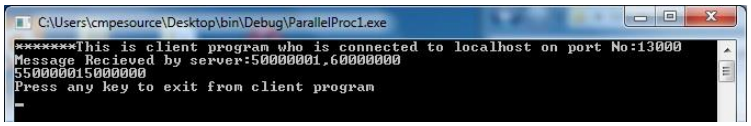

Figure 9 Result of Processor 6

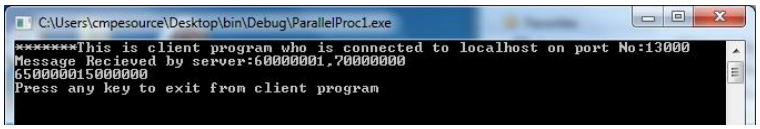

Figure 10 Result of Processor 7

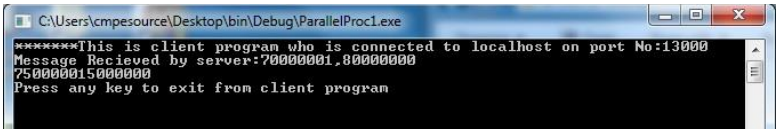

Figure 11 Result of Processor 8

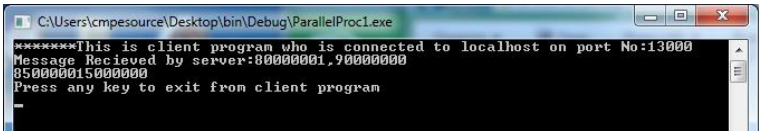

Figure 12 Result of Processor 9

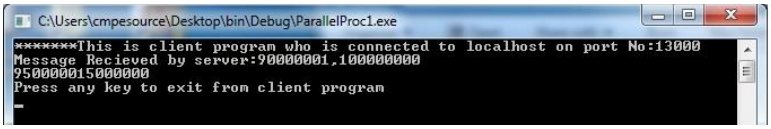

Figure 13 Result of Processor 10

SUMMARY AND CONCLUSION

Parallelism in distributed systems can be defined as the powerful in networking programming. In this term project, we designed a distributed system with a server and 10 clients as processors. We tested our design and the implementation was prefect. Parallelism has been a point of research and development which was based on the distribution system concept. This concept has help to extend to calculate a big number which is heavy to calculation in weak processors or spend a lot of time for getting result.

REFERENCES

[1] How to sound like a Parallel Programming Expert by Timothy Mattson.

[2] Principles of Parallel Programming by Lin C. and Snyder L.

[3] Patterns for Parallel Programming by Mattson, Sanders, and Massingill