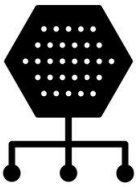
PARALLEL & DISTRIBUTED COMPUTING

Assignment # 1



Multi Threading

Student Name:
Muhammad Farrukh Naveed

Enrollment:
01-134211-056

Class:
BS(CS)-7A

Table of Contents

Problem Overview

The purpose of this assignment was to solve a system of linear equations using c#. A total of three methods/approaches had to be applied in this assignment:

1. Sequential execution
2. Multithreaded execution
3. Threadpool execution

We had to compare the results of each technique.

Objective

The objective of this assignment was to learn the basics of threading in parallel computing. This assignment would serve me as a crucial foundation in learning to optimize any program I would build in future. This assignment also serves to teach the pros and cons of using multithreading in a program.

Methods Overview

Sequential method of execution

Any program that does not utilize multiple threads is sequential in nature. Whenever we execute a piece of code, the system creates a main thread for it's execution, such that every step is sequential and instructions are executed one by one by the CPU.

Multithreaded method of execution

In this method of execution, we divide the task into independent chunks and then assign those tasks/chunks to individual thread objects created by us. Here we must note that multithreading can only be applied to those program/applications which have the scope of being divided, there are some programs whose iterations depend on the result of the previous iterations.

Threadpool method of execution

In this method we do not create individual thread objects ourselves instead we create a thread pool and let the system handle the allocation and priority of queue (Though an argument for priority can pe passed), this method is particularly useful when we don't know how many threads we may require.

Jacobi Method

I used Jacobi's method to solve the system of linear equations, this is a iterative technique in which we first assume all the variables to be zero, we then continuously replace and update the values of variables until we reach a desired iteration, decimal point or convergence.

$$
\begin{aligned}
a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} &= b_1 \\
a_{21}x_1^{(k)} + a_{22}x_2^{(k+1)} + \cdots + a_{2n}x_n^{(k)} &= b_2 \\
\vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots & \\
a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \cdots + a_{nn}x_n^{(k+1)} &= b_n
\end{aligned}
$$

My code works for any number of 'n' variables provided the given equations are perfectly diagonal (in terms of the largest absolute value).

The code is read from input file in the following format:

```
34  -12  6  0
12  -14  8  22
3    6  -2  2
```

Source Code

Jacobi sequential

```
//Author: Muhammad Farrukh Naveed
//StartDate: 28/02/2024
//StartTime: 1:24 PM

//Class: BS(CS)-7A
//Enrollment: 01-134211-056
//Group: None


//*****************************************  PDC ASSIGNMENT NO.1
*****************************************


//Problem: Solving system of linear equationn using sequential,multithreading and
threadpool techniques

//                                        --- Solving Sequentially ---

// ...

using System;
using System.Diagnostics;
using System.IO;
using System.Threading;

class Program
{
    static void Main()
    {
        //creating a timer varibale to keep track of the execution time
        var timer = new Stopwatch();
        timer.Start();

        //creatin a variable to store the main thread's info
        Thread mainSeq = Thread.CurrentThread;

        //Naming the main thread
        mainSeq.Name = "Sequential Main Thread";

        if (mainSeq.ThreadState == System.Threading.ThreadState.Running)
        {
            Console.WriteLine("Main thread started...");
        }


        // since or only change occur at the number value of IN test case
        string commonPath =
"C:\\Users\\farru\\source\\repos\\assignmentPDC\\testcases\\IN";

        // Number of files to be fetched and read
```

```
        int numFiles = 10;

        //Sequentially execute all the files
        for (int i = 0; i <= numFiles; i++)
        {
            performJacobi($"{commonPath}{i}.txt");
        }


        //we prit this once the main thread has finished execution using the
sequential method
        Console.WriteLine(mainSeq.Name + " has completed execution");
        timer.Stop();
        Console.WriteLine("Execution time: " + timer.ElapsedMilliseconds + "ms");
    }

    static void performJacobi(string filePath)
    {
        //No of iterations for performing jacobi are set here
        //more iterations means more accurate resultd
        int iter = 100;

        if (File.Exists(filePath))
        {
            string[] lines = File.ReadAllLines(filePath);
            float[] varval = new float[lines.Length];

            Console.WriteLine(varval.Length);

            //In jacobi initially we assume all the variables to be 0
            for (int i = 0; i < varval.Length; i++)
            {
                varval[i] = 0;
            }

            // Convert the tokens to integers and store them in a array
            float[,] eqtk = new float[lines.Length, lines[0].Split(' ').Length];

            for (int i = 0; i < lines.Length; i++)
            {
                // Spliting each line into individual numbers
                string[] numbers = lines[i].Split(' ');

                for (int j = 0; j < numbers.Length; j++)
                {
                    // typecasting tofloat and storing in the 2d array
                    eqtk[i, j] = float.Parse(numbers[j]);
                }
            }


            /*considering the last element of the equation is that present on the
other side of the "=" sign,
             *hence when we want to perform operations on it we want it on the
other side of the equation. so I
             *change the sign for that element*/
            for (int i = 0; i < eqtk.GetLength(0); i++)
            {
                eqtk[i, eqtk.GetLength(1) - 1] *= -1;
            }


            /*we will divide each respective row with the diagonal element of the
```

```
             matrix assuming that is the largest absolute value amongst the
variables*/

            //This will only be performed once to calculate the euation for each
variable
            HandleDivision(eqtk);

            //Printing the value of variables at each iteration
            for (int i = 0; i <= iter; i++)
            {
                Console.WriteLine("\nITERATION: " + i);
                for (int j = 0; j < varval.Length; j++)
                {
                    Console.WriteLine("x" + (j + 1) + ": " + varval[j]);
                }
                HandleMultiplication(eqtk, varval);
            }
        }
        else
        {
            Console.WriteLine("File not found: " + filePath);
        }
    }

    //Function used for debugging purposes
    static void PrintArray(float[,] array)
    {
        for (int i = 0; i < array.GetLength(0); i++)
        {
            for (int j = 0; j < array.GetLength(1); j++)
            {
                Console.Write(array[i, j] + " ");
            }
            Console.WriteLine();
        }
    }

    /*In this function what we do is we divide the row by the diagonal element of
that matrix
     present in that particular row and we also see if that element is positive
then we have to change
    sign for the rest of the equation because when that element moves across the
= sign it becomes negative
    which is then negated by multiplying both sides by -1*/
    static void HandleDivision(float[,] matrix)
    {
        int rows = matrix.GetLength(0);
        int cols = matrix.GetLength(1);

        for (int i = 0; i < rows; i++)
        {
            // finding the diagonal element
            float diagonalElement = matrix[i, i];

            //since this element moves on the other side of the equation we do
not need it in the final sum
            matrix[i, i] = 0;

            // if the element is positive then that means that we have to change
the sign for the rest of the equation
            for (int j = 0; j < cols; j++)
            {
                if (diagonalElement >= 0)
```

```
                {
                    matrix[i, j] *= -1;
                }

                //Console.WriteLine("\n\n" + matrix[i, j] + "/" + diagonalElement
+ "\n\n");
                matrix[i, j] /= diagonalElement;
            }
        }
    }

    static void HandleMultiplication(float[,] eq, float[] varval)
    {
        //creating a duplicate 2d array to store the manipulations and keep the
original one intact
        float[,] temp = new float[eq.GetLength(0), eq.GetLength(1)];
        for (int i = 0; i < eq.GetLength(0); i++)
        {
            for (int j = 0; j < eq.GetLength(1); j++)
            {
                temp[i, j] = eq[i, j];
            }
        }

        //First we need to replace the values of variables in each equation
        for (int i = 0; i < varval.Length; i++)
        {
            for (int j = 0; j < temp.GetLength(0); j++)
            {
                //Console.WriteLine(temp[j, i] + " * " + varval[i]);
                temp[j, i] *= varval[i];
            }
        }


        //we nned to clear the previous values stored for variables
        for (int i = 0; i < varval.Length; i++)
        {
            varval[i] = 0;
        }

        //Sum of each line of the matrix is stored in the varval array as
variable values
        for (int i = 0; i < temp.GetLength(0); i++)
        {
            for (int j = 0; j < temp.GetLength(1); j++)
            {

                varval[i] += temp[i, j];
                //Console.WriteLine(temp[i, j] + " + " + varval[i]);

            }
        }
    }
}

// ...

// Time to completion: 6.3 hrs
// Actual Lines of code: 112
// This code has been pushed to origin on 02/02/2024 at 10:37pm
```

Jacobi Multithreaded

```
//Author: Muhammad Farrukh Naveed
//StartDate: 28/02/2024
//StartTime: 1:24 PM

//Class: BS(CS)-7A
//Enrollment: 01-134211-056
//Group: None


//*****************************************  PDC ASSIGNMENT NO.1
*****************************************


//Problem: Solving system of linear equationn using sequential,multithreading and
threadpool techniques

//                                          --- Solving Multithreaded ---

// ...

using System;
using System.Diagnostics;
using System.IO;
using System.Threading;

class Program
{
    static void Main()
    {
        //creating a timer varibale to keep track of the execution time
        var timer = new Stopwatch();
        timer.Start();

        //creatin a variable to store the main thread's info
        Thread mainSeq = Thread.CurrentThread;

        //Naming the main thread
        mainSeq.Name = "Parent Main Thread";

        if (mainSeq.ThreadState == System.Threading.ThreadState.Running)
        {
            Console.WriteLine("Main thread started...");
        }


        // since or only change occur at the number value of IN test case
        string commonPath =
"C:\\Users\\farru\\source\\repos\\assignmentPDC\\testcases\\IN";

        // Number of files to be fetched and read
        int numFiles = 10;

        // creating an array of threads such that each thread corresponds to
solving one testcase
        Thread[] threads = new Thread[numFiles + 1];
        for (int i = 0; i <= numFiles; i++)
        {
            string filePath = $"{commonPath}{i}.txt";
```

```
            threads[i] = new Thread(() => performJacobi(filePath));
            threads[i].Start();
        }

        // we will wait so that all threads have finished job
        // This is done to prevent the termination of main thread before hand
        foreach (Thread thread in threads)
        {
            thread.Join();
        }


        //we prit this once the main thread has finished execution using the
sequential method
        Console.WriteLine(mainSeq.Name + " has completed execution");
        timer.Stop();
        Console.WriteLine("Execution time: " + timer.ElapsedMilliseconds + "ms");
    }

    static void performJacobi(string filePath)
    {
        //No of iterations for performing jacobi are set here
        //more iterations means more accurate resultd
        int iter = 100;

        if (File.Exists(filePath))
        {
            string[] lines = File.ReadAllLines(filePath);
            float[] varval = new float[lines.Length];

            Console.WriteLine(varval.Length);

            //In jacobi initially we assume all the variables to be 0
            for (int i = 0; i < varval.Length; i++)
            {
                varval[i] = 0;
            }

            // Convert the tokens to integers and store them in a array
            float[,] eqtk = new float[lines.Length, lines[0].Split(' ').Length];

            for (int i = 0; i < lines.Length; i++)
            {
                // Spliting each line into individual numbers
                string[] numbers = lines[i].Split(' ');

                for (int j = 0; j < numbers.Length; j++)
                {
                    // typecasting tofloat and storing in the 2d array
                    eqtk[i, j] = float.Parse(numbers[j]);
                }
            }


            /*considering the last element of the equation is that present on the
other side of the "=" sign,
             *hence when we want to perform operations on it we want it on the
other side of the equation. so I
             *change the sign for that element*/
            for (int i = 0; i < eqtk.GetLength(0); i++)
            {
                eqtk[i, eqtk.GetLength(1) - 1] *= -1;
            }
```

```
            /*we will divide each respective row with the diagonal element of the
             matrix assuming that is the largest absolute value amongst the
variables*/

            //This will only be performed once to calculate the euation for each
variable
            HandleDivision(eqtk);

            //Printing the value of variables at each iteration
            for (int i = 0; i <= iter; i++)
            {
                Console.WriteLine("\nITERATION: " + i);
                for (int j = 0; j < varval.Length; j++)
                {
                    Console.WriteLine("x" + (j + 1) + ": " + varval[j]);
                }
                HandleMultiplication(eqtk, varval);
            }
        }
        else
        {
            Console.WriteLine("File not found: " + filePath);
        }
    }

    //Function used for debugging purposes
    static void PrintArray(float[,] array)
    {
        for (int i = 0; i < array.GetLength(0); i++)
        {
            for (int j = 0; j < array.GetLength(1); j++)
            {
                Console.Write(array[i, j] + " ");
            }
            Console.WriteLine();
        }
    }

    /*In this function what we do is we divide the row by the diagonal element of
that matrix
     present in that particular row and we also see if that element is positive
then we have to change
    sign for the rest of the equation because when that element moves across the
= sign it becomes negative
    which is then negated by multiplying both sides by -1*/
    static void HandleDivision(float[,] matrix)
    {
        int rows = matrix.GetLength(0);
        int cols = matrix.GetLength(1);

        for (int i = 0; i < rows; i++)
        {
            // finding the diagonal element
            float diagonalElement = matrix[i, i];

            //since this element moves on the other side of the equation we do
not need it in the final sum
            matrix[i, i] = 0;

            // if the element is positive then that means that we have to change
the sign for the rest of the equation
```

```
            for (int j = 0; j < cols; j++)
            {
                if (diagonalElement >= 0)
                {
                    matrix[i, j] *= -1;
                }

                //Console.WriteLine("\n\n" + matrix[i, j] + "/" + diagonalElement
+ "\n\n");
                matrix[i, j] /= diagonalElement;
            }
        }
    }

    static void HandleMultiplication(float[,] eq, float[] varval)
    {
        //creating a duplicate 2d array to store the manipulations and keep the
original one intact
        float[,] temp = new float[eq.GetLength(0), eq.GetLength(1)];
        for (int i = 0; i < eq.GetLength(0); i++)
        {
            for (int j = 0; j < eq.GetLength(1); j++)
            {
                temp[i, j] = eq[i, j];
            }
        }

        //First we need to replace the values of variables in each equation
        for (int i = 0; i < varval.Length; i++)
        {
            for (int j = 0; j < temp.GetLength(0); j++)
            {
                //Console.WriteLine(temp[j, i] + " * " + varval[i]);
                temp[j, i] *= varval[i];
            }
        }


        //we nned to clear the previous values stored for variables
        for (int i = 0; i < varval.Length; i++)
        {
            varval[i] = 0;
        }

        //Sum of each line of the matrix is stored in the varval array as
variable values
        for (int i = 0; i < temp.GetLength(0); i++)
        {
            for (int j = 0; j < temp.GetLength(1); j++)
            {

                varval[i] += temp[i, j];
                //Console.WriteLine(temp[i, j] + " + " + varval[i]);

            }
        }
    }
}

// ...

// Time to completion: 6.3 hrs
// Actual Lines of code: 121
```

```
// This code has been pushed to origin on 02/02/2024 at 10:42pm
```

Jacobi Thread Pool

```
//Author: Muhammad Farrukh Naveed
//StartDate: 28/02/2024
//StartTime: 1:24 PM

//Class: BS(CS)-7A
//Enrollment: 01-134211-056
//Group: None


//*****************************************  PDC ASSIGNMENT NO.1
*****************************************


//Problem: Solving system of linear equationn using sequential,multithreading and
threadpool techniques

//                                          --- Solving ThreadPooling ---

// ...

using System;
using System.Diagnostics;
using System.IO;
using System.Threading;

class Program
{
    // we use a countdown event to keep track of the threads that hae signalled
completion
    static CountdownEvent? countdownEvent;

    static void Main()
    {
        //creating a timer varibale to keep track of the execution time
        var timer = new Stopwatch();
        timer.Start();

        //creatin a variable to store the main thread's info
        Thread mainSeq = Thread.CurrentThread;

        //Naming the main thread
        mainSeq.Name = "Parent Main Thread";

        if (mainSeq.ThreadState == System.Threading.ThreadState.Running)
        {
            Console.WriteLine("Main thread started...");
        }


        // since or only change occur at the number value of IN test case
        string commonPath =
"C:\\Users\\farru\\source\\repos\\assignmentPDC\\testcases\\IN";
```

```
        // Number of files to be fetched and read
        int numFiles = 10;

        countdownEvent = new CountdownEvent(numFiles);

        // using threadpool to assign each testcase to a thread
        for (int i = 0; i <= numFiles; i++)
        {
            string filePath = $"{commonPath}{i}.txt";
            ThreadPool.QueueUserWorkItem(_ => performJacobi(filePath));
        }

        countdownEvent.Wait();


        //we prit this once the main thread has finished execution using the
sequential method
        Console.WriteLine(mainSeq.Name + " has completed execution");
        timer.Stop();
        Console.WriteLine("Execution time: " + timer.ElapsedMilliseconds + "ms");
    }

    static void performJacobi(string filePath)
    {
        //No of iterations for performing jacobi are set here
        //more iterations means more accurate resultd
        int iter = 100;

        if (File.Exists(filePath))
        {
            string[] lines = File.ReadAllLines(filePath);
            float[] varval = new float[lines.Length];

            Console.WriteLine(varval.Length);

            //In jacobi initially we assume all the variables to be 0
            for (int i = 0; i < varval.Length; i++)
            {
                varval[i] = 0;
            }

            // Convert the tokens to integers and store them in a array
            float[,] eqtk = new float[lines.Length, lines[0].Split(' ').Length];

            for (int i = 0; i < lines.Length; i++)
            {
                // Spliting each line into individual numbers
                string[] numbers = lines[i].Split(' ');

                for (int j = 0; j < numbers.Length; j++)
                {
                    // typecasting tofloat and storing in the 2d array
                    eqtk[i, j] = float.Parse(numbers[j]);
                }
            }


            /*considering the last element of the equation is that present on the
other side of the "=" sign,
             *hence when we want to perform operations on it we want it on the
other side of the equation. so I
             *change the sign for that element*/
```

```
            for (int i = 0; i < eqtk.GetLength(0); i++)
            {
                eqtk[i, eqtk.GetLength(1) - 1] *= -1;
            }


            /*we will divide each respective row with the diagonal element of the
             matrix assuming that is the largest absolute value amongst the
variables*/

            //This will only be performed once to calculate the euation for each
variable
            HandleDivision(eqtk);

            //Printing the value of variables at each iteration
            for (int i = 0; i <= iter; i++)
            {
                Console.WriteLine("\nITERATION: " + i);
                for (int j = 0; j < varval.Length; j++)
                {
                    Console.WriteLine("x" + (j + 1) + ": " + varval[j]);
                }
                HandleMultiplication(eqtk, varval);
            }

            countdownEvent.Signal();
        }
        else
        {
            Console.WriteLine("File not found: " + filePath);
        }
    }

    //Function used for debugging purposes
    static void PrintArray(float[,] array)
    {
        for (int i = 0; i < array.GetLength(0); i++)
        {
            for (int j = 0; j < array.GetLength(1); j++)
            {
                Console.Write(array[i, j] + " ");
            }
            Console.WriteLine();
        }
    }

    /*In this function what we do is we divide the row by the diagonal element of
that matrix
     present in that particular row and we also see if that element is positive
then we have to change
    sign for the rest of the equation because when that element moves across the
= sign it becomes negative
    which is then negated by multiplying both sides by -1*/
    static void HandleDivision(float[,] matrix)
    {
        int rows = matrix.GetLength(0);
        int cols = matrix.GetLength(1);

        for (int i = 0; i < rows; i++)
        {
            // finding the diagonal element
            float diagonalElement = matrix[i, i];
```

```
            //since this element moves on the other side of the equation we do
not need it in the final sum
            matrix[i, i] = 0;

            // if the element is positive then that means that we have to change
the sign for the rest of the equation
            for (int j = 0; j < cols; j++)
            {
                if (diagonalElement >= 0)
                {
                    matrix[i, j] *= -1;
                }

                //Console.WriteLine("\n\n" + matrix[i, j] + "/" + diagonalElement
+ "\n\n");
                matrix[i, j] /= diagonalElement;
            }
        }
    }

    static void HandleMultiplication(float[,] eq, float[] varval)
    {
        //creating a duplicate 2d array to store the manipulations and keep the
original one intact
        float[,] temp = new float[eq.GetLength(0), eq.GetLength(1)];
        for (int i = 0; i < eq.GetLength(0); i++)
        {
            for (int j = 0; j < eq.GetLength(1); j++)
            {
                temp[i, j] = eq[i, j];
            }
        }

        //First we need to replace the values of variables in each equation
        for (int i = 0; i < varval.Length; i++)
        {
            for (int j = 0; j < temp.GetLength(0); j++)
            {
                //Console.WriteLine(temp[j, i] + " * " + varval[i]);
                temp[j, i] *= varval[i];
            }
        }


        //we nned to clear the previous values stored for variables
        for (int i = 0; i < varval.Length; i++)
        {
            varval[i] = 0;
        }

        //Sum of each line of the matrix is stored in the varval array as
variable values
        for (int i = 0; i < temp.GetLength(0); i++)
        {
            for (int j = 0; j < temp.GetLength(1); j++)
            {

                varval[i] += temp[i, j];
                //Console.WriteLine(temp[i, j] + " + " + varval[i]);

            }
        }
    }
```

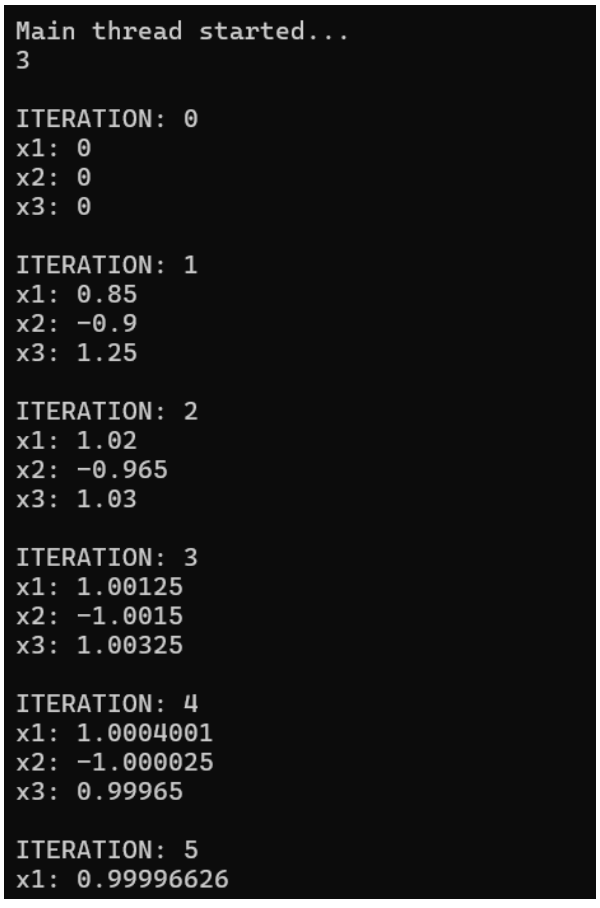
```
}

// ...

// Time to completion: 6.3 hrs
// Actual Lines of code: 120
// This code has been pushed to origin on 02/02/2024 at 11:07pm
```

Outputs

Jacobi sequential



```
x1: 2.425476
x2: 3.5730157
x3: 1.925954

ITERATION: 97
x1: 2.425476
x2: 3.5730157
x3: 1.925954

ITERATION: 98
x1: 2.425476
x2: 3.5730157
x3: 1.925954

ITERATION: 99
x1: 2.425476
x2: 3.5730157
x3: 1.925954

ITERATION: 100
x1: 2.425476
x2: 3.5730157
x3: 1.925954
Sequential Main Thread has completed execution
Execution time: 180ms
```

Jacobi Multithreaded

```
Main thread started...
3
3
2
4
3
4
4
3
3

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0
3
```

```
x2: -0.31013307
x1: 1
x3: -0.1258956

ITERATION: 100
x2: 2
x3: 3
x4: 5.9604645E-08
x1: 0.11975436
x2: -0.31013307
x3: -0.1258956

ITERATION: 99
x1: 1
x2: 2
x3: 3
x4: 5.9604645E-08

ITERATION: 100
x1: 1
x2: 2
x3: 3
x4: 5.9604645E-08
Parent Main Thread has completed execution
Execution time: 220ms
```

Jacobi Thread pool

```
Main thread started...
4
3
3
2
3
4

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0

ITERATION: 0
x1: 2.614266
x2: -1.892055
x3: 0.40691888
x4: -0.327691

ITERATION: 100
x1: 2.6140394
x2: -1.892152
x3: 0.4068604
x4: -0.32765672
x4: 0.96618354

ITERATION: 81
x1: 3.992754
x2: 2.954106
x3: 2.1618357
x4: 0.96618354
Parent Main Thread has completed execution

ITERATION: 82
x1: 3.992754
x2: 2.954106
x3: 2.1618357
x4: 0.96618354

ITERATION: 83
x1: 3.992754
x2: 2.954106
x3: 2.1618357
Execution time: 178ms
```

Comparison Table

#testcases	Sequential	Multithreded	Threadpool
1	8ms	12ms	9ms
5	21ms	32ms	19ms
10	180ms	220ms	178ms

* Please Note that last 10 testcases were ran with 100 iterations per question

Conclusion

I had a very good experience solving this assignment, The quality of question was exceptionally good. The assignment made me think logically about the system resources.

I learned about creating and using threads and parallelizing my task, similarly I also learned about the drawbacks of using threads in certain cases, i.e., it can add a significant performance overhead by creating and managing multiple threads at a time can eat up system resources.

I also learned about using thread pool, in which instead of creating threads individually we assign this task to the scheduler.