KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

CONCURRENT PROGRAMMING
INDIVIDUAL PROJECT

ALDRIC SHAUN RAJESH IFU-0

2022

1. Description:

Goal — The aim of my program is to simplify and make a word-processing task from my object-oriented programming module using the concurrency tools and methods we learned this semester.

2. Task:

I have created a program that downloads text from 8 different websites locally to 8 different textiles. we then execute the code in parallel to read these textiles and count the number of vowels in each of them. We are simultaneously calculating the time taken to execute this code by varying the number of threads being used to find out the most optimum number of threads to use for executing this code.

The tool we are using to achieve this is C# parallel LINQ.

3. Testing Methods

- There are no specific requirements needed for running this project.
- The code uses Pre-existing C# tools to perform the count.
- The website links are placed inside string variables inside the code which can be changed to the user's requirements.
- We are performing speed tests with every iteration to calculate the optimum number of threads to use.

4. Source Code:

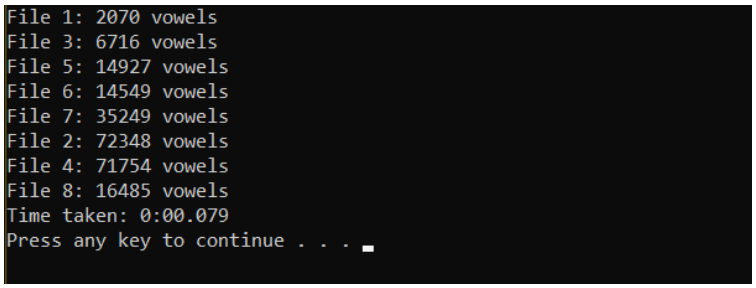
```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Threading;
using System.Diagnostics;

namespace L1_A
{
    class Program
    {
        const string read1 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text1.txt";
        const string read2 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text2.txt";
        const string read3 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text3.txt";
        const string read4 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text4.txt";
        const string read5 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text5.txt";
        const string read6 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text6.txt";
        const string read7 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text7.txt";
        const string read8 = @"C:\Users\foxtr\OneDrive\Desktop\All Uni
softwares\concurrent code\Final Project\Rajesh_Lab1\Text8.txt";

        static void Main(string[] args)
        {
            string website_1="https://moodle.ktu.edu/course/view.php?id=6837";
            string website_2="https://www.fighterpilotpodcast.com/glossary/";
            string website_3="https://learn.microsoft.com/en-us/visualstudio";
            string website_4="https://www.tunefind.com/";
            string website_5="https://www.britannica.com/technology/concurrent-
programming";
            string
website_6="https://wiki.esn.org/pages/viewpage.action?pageId=7045162";
            string website_7="https://combatace.com/files/file/9401-super-hornet-
package-for-sf2-v41/";
            string
website_8="https://wiki.esn.org/display/WEL/Welcome+to+ESN+Wiki+-
+Guide+for+beginners";

            // Create a list of file paths
            List<string> filePaths = new List<string>(){ read1, read2, read3 ,
read4, read5 , read6, read7, read8 };

            // Create a list of websites
            List<string> websites = new List<string>() { website_1, website_2,
website_3, website_4, website_5, website_6, website_7, website_8 };
```

```
            // Download the text from each website and save it to the
corresponding text file
            for (int i = 0; i < websites.Count; i++)
            {
                string text = DownloadText(websites[i]);
                SaveTextToFile(text, filePaths[i]);
            }

            var timer = new Stopwatch();
            timer.Start();

            // Read the text from each file and count the number of vowels
            int numThreads = 5;
            ParallelEnumerable.Range(0,
filePaths.Count).WithDegreeOfParallelism(numThreads).ForAll(i => {
                string text = ReadTextFromFile(filePaths[i]); int vowelCount =
text.Count(c => "aeiouAEIOU".Contains(c));
                Console.WriteLine($"File {i + 1}: {vowelCount} vowels");
            });
            timer.Stop();

            TimeSpan timeTaken = timer.Elapsed;
            string foo = "Time taken: " + timeTaken.ToString(@"m\:ss\.fff");
            Console.WriteLine(foo);

        }

        // Function to download the text from a website
        static string DownloadText(string website)
        {
            using (WebClient client = new WebClient())
            {
                return client.DownloadString(website);
            }
        }

        // Function to save text to a file
        static void SaveTextToFile(string text, string filePath)
        {
            File.WriteAllText(filePath, text);
        }

        // Function to read text from a file
        static string ReadTextFromFile(string filePath)
        {
            return File.ReadAllText(filePath);
        }

        // Function to count the number of vowels in a string
        static int CountVowels(string text)
        {
            return text.Count(c => "aeiouAEIOU".Contains(c));
        }
    }
}
```

Output:

Iteration 1:

Iteration 2:

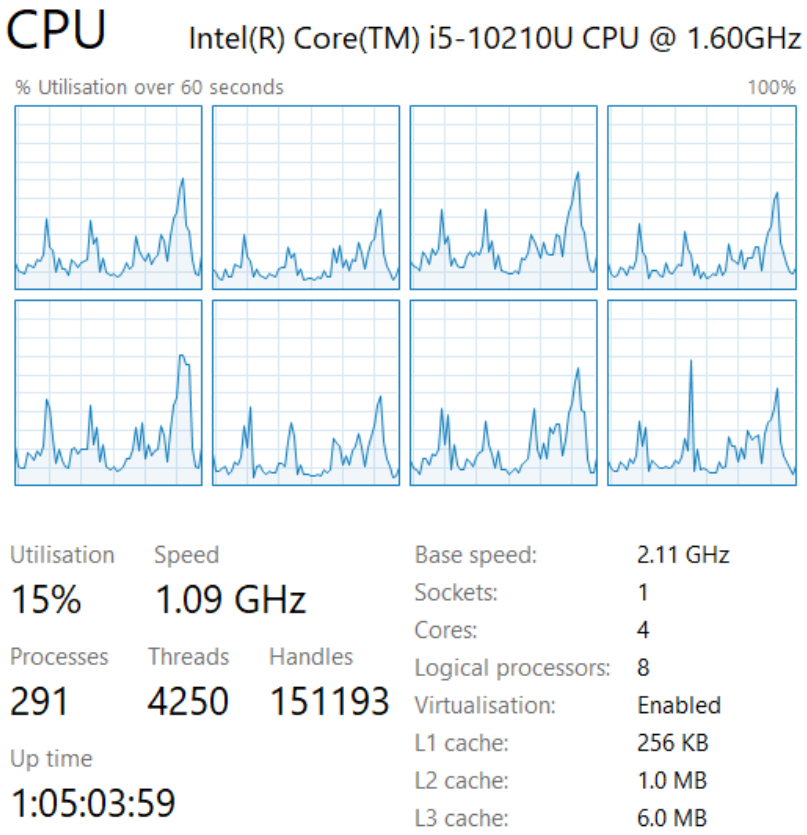


Iteration 3:

5. Performance Analysis:

Below I have mentioned all the speed tests conducted to determine the optimum number of threads to use to perform this task.

Computer Specifications:



Device name **Aldys-Laptop**

Processor **Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz**

Installed RAM **8.00 GB (7.83 GB usable)**

Device ID **9EAA6D97-2B63-4A91-B054-792A0E75467D**

Product ID **00327-35833-05467-AAOEM**

System type64-bit operating system, x64-based processor

Pen and touch **No pen or touch input is available for this display**

The number of threads the processor can run concurrently depends on the number of physical cores and the number of threads per core that your processor supports. The Intel Core i5-10210U processor has 4 cores and 8 threads, so it can run up to 8 threads concurrently.

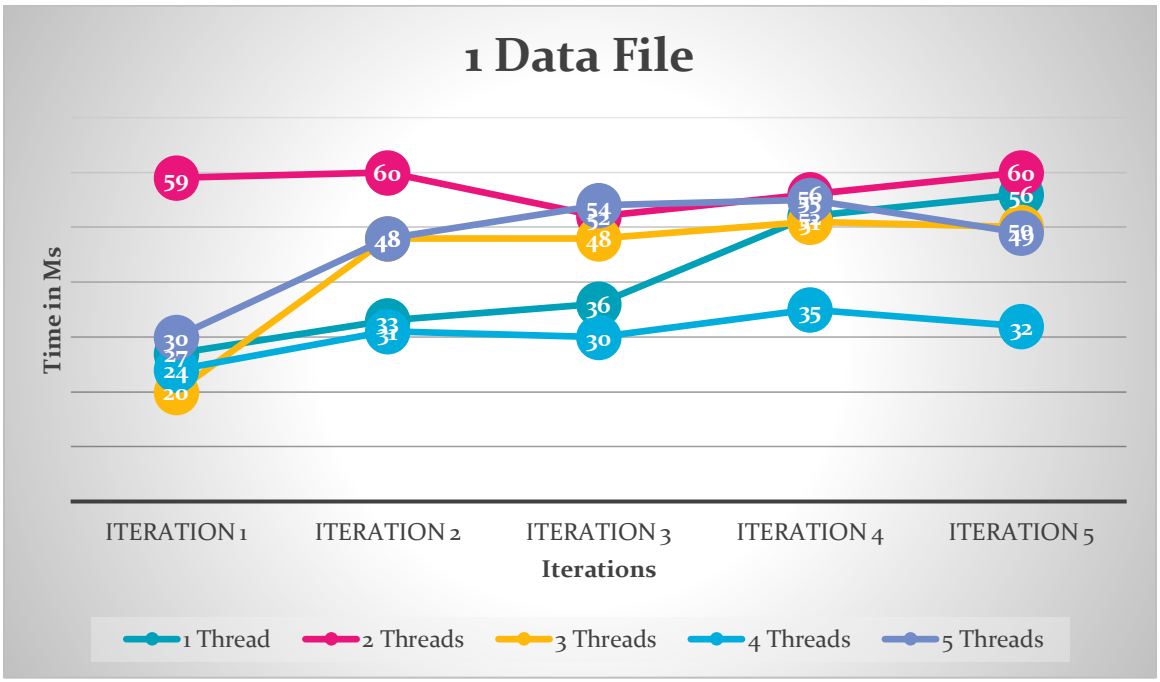
The amount of memory (RAM) that the device has can also affect processing time, especially if the program requires a lot of memory. With 8 GB of RAM, I should be able to run multiple programs at the same time without running into memory issues.

The type of tasks that your program is performing can also impact processing time. Some tasks, such as those that involve a lot of math or data manipulation, can be more CPU-intensive and take longer to complete.

Finally, the overall speed and performance of your device can also affect processing time. The Intel Core i5-10210U processor is a mid-range processor with a base clock speed of 1.60 GHz and a boost speed of 4.20 GHz, so it should be able to handle most tasks fairly quickly.

1st Test:

Processing 1 Data File:



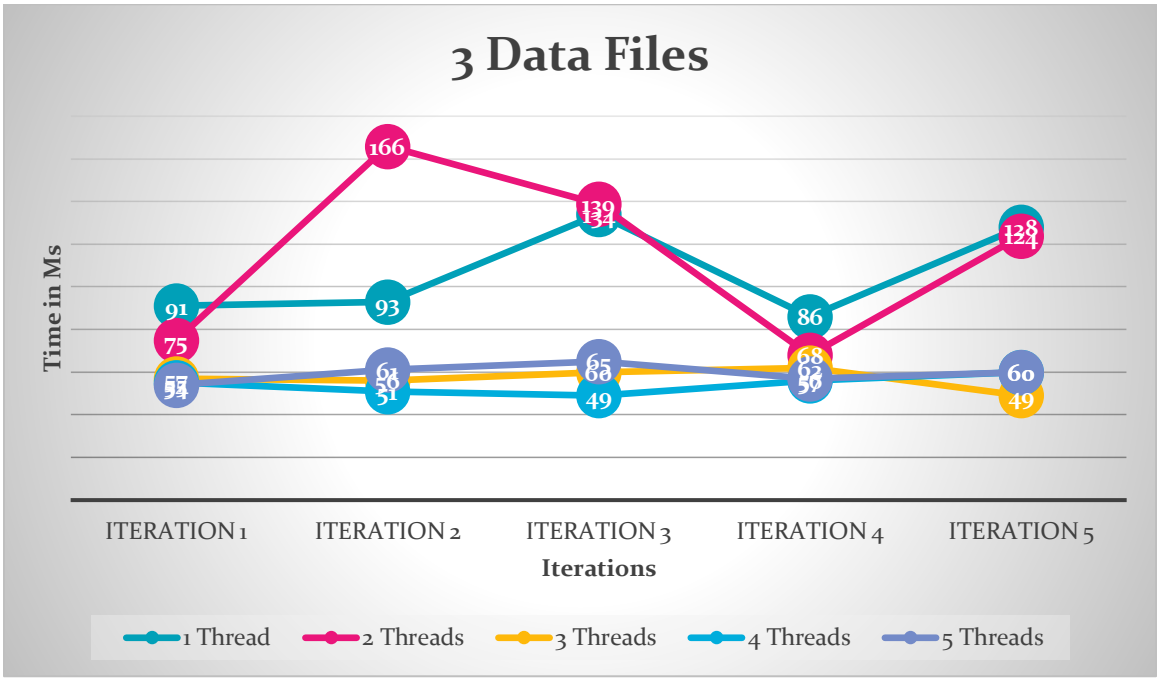
	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
Iteration 1	27	59	20	24	30
Iteration 2	33	60	48	31	48
Iteration 3	36	52	48	30	54
Iteration 4	52	56	51	35	55
Iteration 5	56	60	50	32	49
Average(ms)	38.4	57.4	39.8	38	41.2

We can see that when the code is executed with just one thread it tends to finish faster than when executing it with 2 threads.

We can also observe from the above graph that with 4 threads we are able to achieve the most consistent times while being the most efficient in terms of performance.

2nd Test:

Processing 3 Data Files:



	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
Iteration 1	91	75	57	55	54
Iteration 2	93	166	56	51	61
Iteration 3	134	139	60	49	65
Iteration 4	86	68	62	56	57
Iteration 5	128	124	49	60	60

Average(ms)	106.4	114.4	56.8	54.2	60

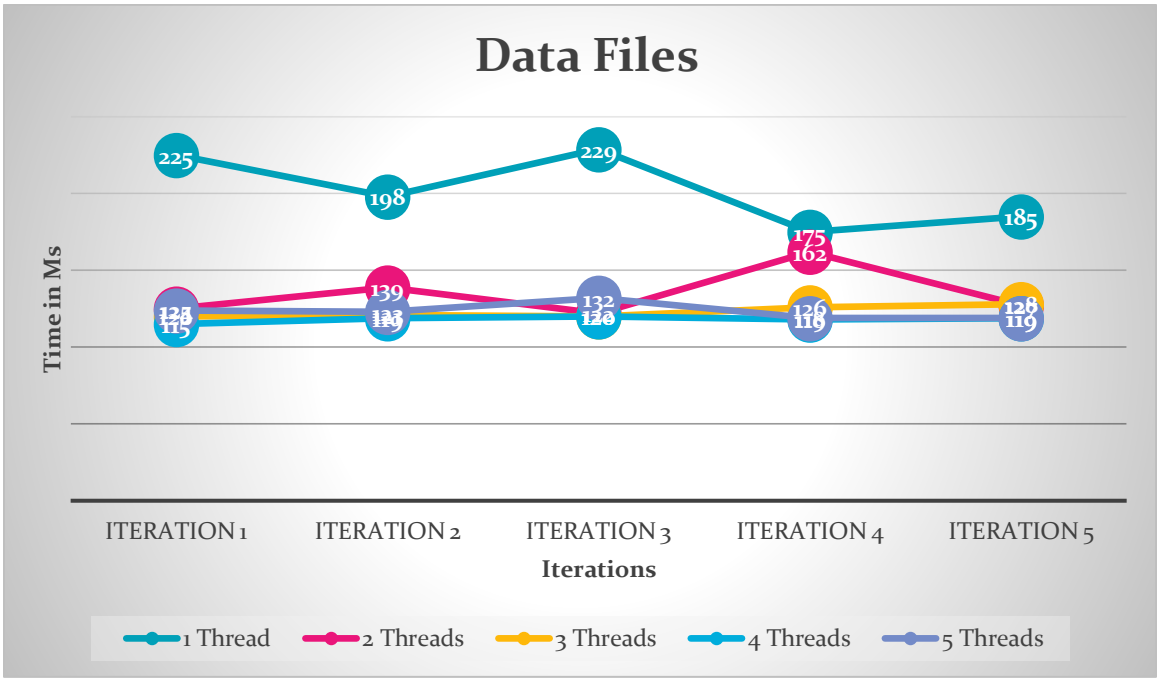
When we increase the number of files in the process we can observe changes in the processing times when the code executes with 1 and 2 threads, they take longer to execute.

This curve flattens when we increase the number of threads to 3 and more. We are able to observe a flatter curve and more consistent times

Again we are able to observe that the most efficient times are when the code is executed with 4 threads.

3rd Test:

Processing 5 Data Files:



	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
Iteration 1	225	125	120	115	124
Iteration 2	198	139	121	119	123

Iteration 3	229	122	120	120	132
Iteration 4	175	162	126	118	119
Iteration 5	185	127	128	119	119
Average(ms)	202.4	135	123	118.2	120.6

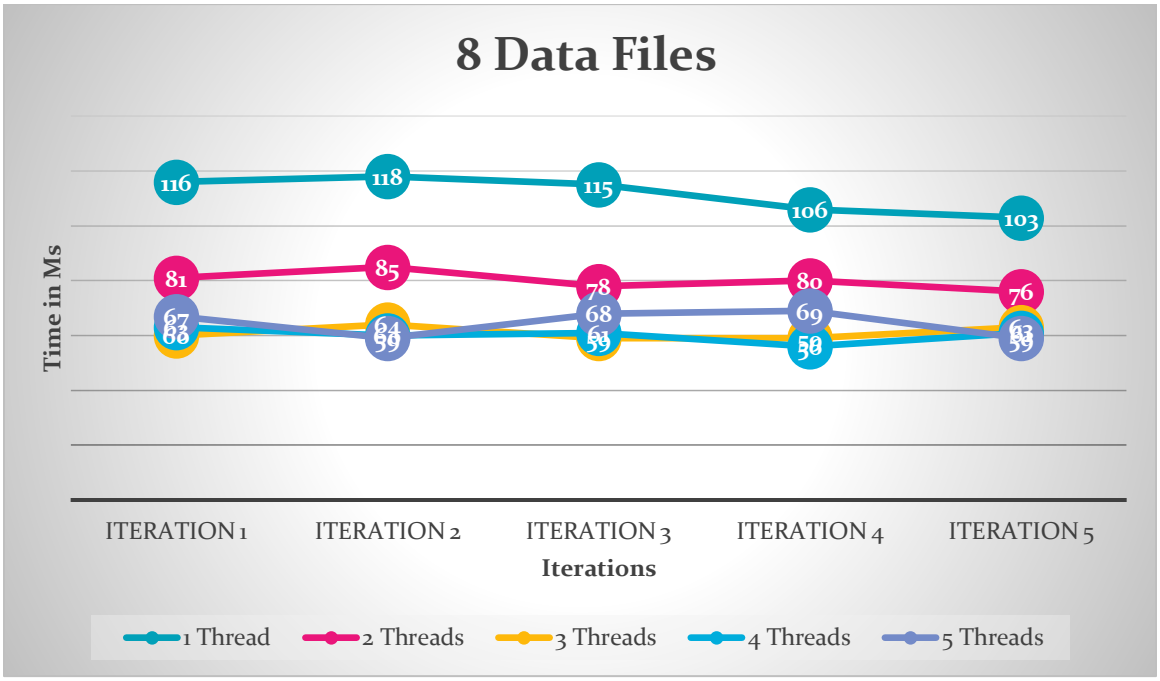
When we execute 5 files the processing load increases and we can see the CPU struggles to execute the code with 1 thread with reasonable times.

This curve flattens when we increase the number of threads to 3 and more. We are able to observe efficient performance and more consistent times

Again, we are able to observe that the most efficient times are when the code is executed with 4 threads.

4th Test:

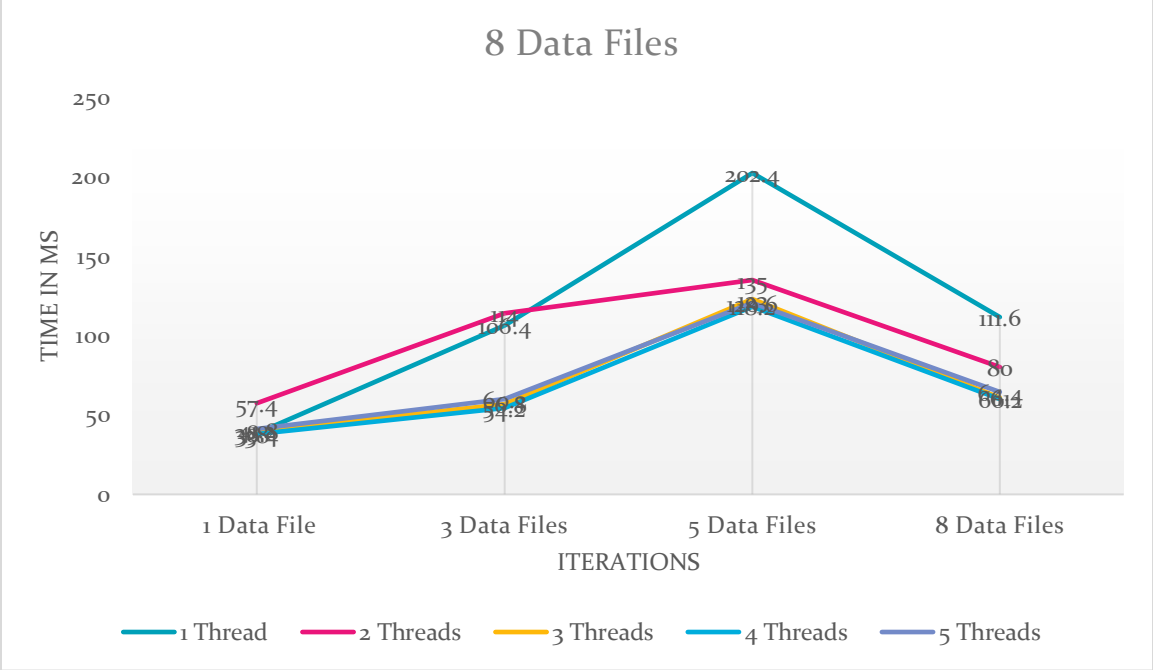
Processing 8 Data Files:



	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
Iteration 1	116	81	60	63	67
Iteration 2	118	85	64	60	59
Iteration 3	115	78	59	61	68
Iteration 4	106	80	59	56	69
Iteration 5	103	76	63	61	59
Average(ms)	111.6	80	61	60.2	64.4

Average graph:

Processing 5 Data Files:



	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
1 Data File	38.4	57.4	39.8	38	41.2
3 Data Files	106.4	114	56.8	54.2	60.2

5 Data Files	202.4	135	123	118.2	120.6
8 Data Files	111.6	80	61	60.2	64.4

Based on the average execution timetable, it looks like using 4 threads may be the optimal number of threads to use for this code.

Here are a few observations based on the data:

- For the case with 1 data file, using 4 threads seems to be the fastest option, with an average execution time of 38 seconds.
- For the case with 3 data files, using 4 threads also seems to be the fastest option, with an average execution time of 54.2 seconds.
- For the case with 5 data files, using 4 threads is the second-fastest option, with an average execution time of 118.2 seconds. The fastest, in the case of thi,s case is using 2 threads, with an average execution time of 135 seconds
- For the case with 8 data files, using 4 threads is the second-fastest option, with an average execution time of 60.2 seconds. The fastest, in this case, is the case is using 1 thread, with an average execution time of 111.6 seconds.
- Overall, it appears that using 4 threads is the optimal number of threads for this code, as it provides the best performance in 3 out of 4 cases.

6. Conclusion:

This is a program that downloads text from 8 websites and writes it to 8 text files, and then uses Parallel LINQ to count the number of vowels in each text file.

We are performing performance analysis on this code to determine which conditions and how many threads are required for the program to execute with moss efficient execution time.

Using more threads can potentially improve the performance of the program by allowing it to complete tasks in parallel. However, it's also possible that using more threads can slow down using more threads can potentially improve the performance of your program by allowing it to complete tasks in parallel. However, it's also possible that using more threads can slow down the processing time in certain situations. A few reasons why this might happen:

Overhead: Creating and managing additional threads can introduce overhead, which can decrease overall performance. This is especially true if the threads are very short-lived or if there are a large number of them.

Contention: If multiple threads are trying to access the same shared resource, such as a file or a database, it can cause contention and lead to slower performance.

Context switching: If the processor has to frequently switch between different threads, it can lead to decreased performance. This can happen if the threads are not well-balanced in terms of the amount of work they're doing.

Limited resources: If the program requires more resources (e.g. CPU, memory, etc.) than are available on the device, using more threads can actually slow down the processing time.

It's important to keep in mind that the optimal number of threads to use can depend on a variety of factors and can vary from one program to another.