Multi-threaded programming

Summary

This thesis describes different multithreading techniques and how they are realized in modern environment, specifically C# and .NET. The goal of this thesis is to research how a modern application could run on multiple threads, what are the tools that are necessary for creating a multithreaded application and what are the benefits that threading can provide. These principles are demonstrated on a C# application, which provides a benchmark for selected implementation scenarios on a set of complex tasks.

Nowadays the term programming has a lot of meanings. To know how to program does not mean only to have the knowledge of programming language and its syntax, but also the ability to use a lot of other important tools, such as frameworks, software, hardware, etc. Among such tools are the different program execution methods. Apart from the "classical" line-after-line code execution, there exist another way to execute the code – multithreaded execution.

The objective of this thesis is to determine what is multithreaded programming and where it is appropriate to use. It is generally assumed, especially among less experienced developers, that creating multithreaded application is an extremely challenging task which requires thorough knowledge of the programming language and frameworks which are used for threading. By writing this thesis I want to investigate how threading is implemented in a modern .NET framework. Also, I want to research is there are cases when code written to be executed by multiple threads will improve the performance and user experience of the application, and hence may replace the classic code that is written for a single threaded execution.

In order to achieve the set objectives, I used multiple books and Microsoft documentation website for researching and obtaining information about how threading is implemented in .NET. Practical part of the thesis is focused on how the theoretical concepts of threading could be implemented in a real-world application and how exactly they affect the application. For practical part I created a WPF application using Visual Studio 2019 along with .NET framework for implementing multithreading in the application. The app consists of 4 execution modes – Synchronous execution, Asynchronous execution, Parallel execution and Asynchronous + Parallel execution. When a user clicks the respective to the execution mode button, the app starts its execution in the selected execution mode. and create multiple web requests. During each cycle of execution, the app is performing 3 different tests which are designed to load CPU and create multiple web requests.

Additionally, the app allows to cancel current execution cycle and display the results of all executions.

In order to collect an accurate statistic of the application performance, every execution mode has been run for 50 times each. Benchmark was executed on my personal computer. It has the following technical specifications:

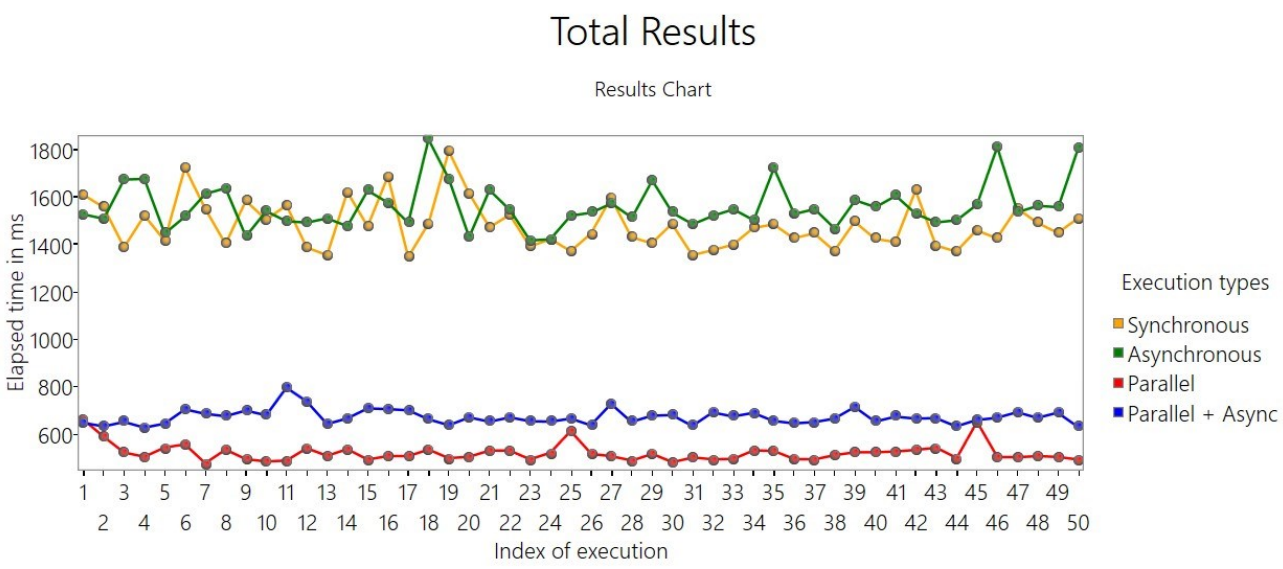
- Operating System: Windows 10
- CPU: AMD Ryzen 3 2200G 3.5 GHz
- Motherboard: ASUS ROG Strix B450-F
- GPU: AMD Vega 56
- RAM: G.Skill RipjawsV 8GB DDR4 3200 MHz

Because tests are designed to load the computer, it would take more time to execute the tests on a lower-end machine (such as laptop), and oppositely, computer with better technical specifications then my machine could execute the tests faster. While running the tests there was no processes running in background, since additional processes may require resources from the computer and thus may affect the results of the benchmark.

The time results of the executions are following:

- Synchronous execution: 14.75 seconds
- Asynchronous execution: 15.54 seconds
- Parallel execution: 5.13 seconds
- Parallel + Asynchronous execution: 6.66 seconds

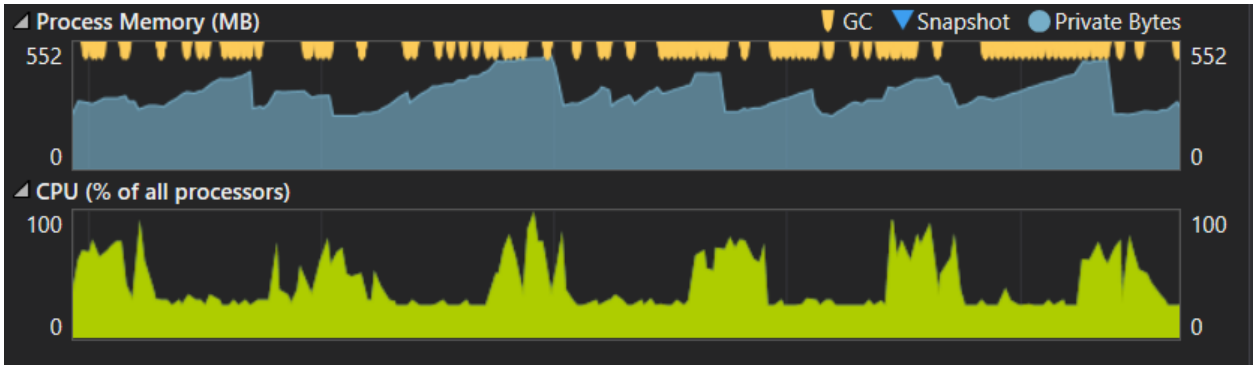
Below is the chart that visually represents the results:



Total results chart

Both parallel execution modes greatly outperform sequential execution modes in terms of total execution time, however, while application is running in parallel mode it requires more resources from the CPU and slightly more operating memory, as shown on figures below.

Benchmark execution in Asynchronous (sequential) mode



Benchmark execution in Parallel + Asynchronous mode

As a result, while app takes much less time to complete all tasks, it appears "slow" or "less responsive" for the user and thus is worse for overall user experience. Another issue are potential

failures during the execution of tests because of high CPU load – app has a higher chance to crash while performing tasks in parallel than sequential mode (although the chance is still low).

Another trend which is observed from the total results is that the execution in asynchronous modes takes on average more time to complete than non-asynchronous modes. It is expected since the app also has to continuously update the UI of the app and keep the app responsive. However, when looking at the app from user experience perspective, the advantages of asynchronous execution modes outweigh the increase of overall execution time by approx. 1 second.

From the results of the tests, it is possible to conclude that asynchronous programming significantly improves user experience at a slight performance cost related to regular UI updates, meaning that asynchronous execution should be implemented for any operation that is longer than 10-15 seconds in order to not leave the user with an unresponsive application. Parallel execution is also an extremely useful technique, which allows to dramatically improve the application performance, especially in cases where there are multiple independent tasks which could to be executed simultaneously.

Theoretical part of the thesis covers in detail the implementation of multithreading in C# and .NET. In order to gain the necessary information regarding the theoretical topics of the thesis I used multiple books and websites (mainly Microsoft Documentation (https://docs.microsoft.com/en-us/dotnet/csharp/) and Stackoverflow (https://stackoverflow.com/) websites). Despite the fact that threading techniques have a slightly steep learning curve, and require in depth knowledge of the framework to fully utilize them, modern frameworks are greatly optimized for multithreaded development, making it easy to implement basic threading and already improve the application.

Although the app that is shown in the practical part is not nearly as large and complicated as a "real-world" released application, it still demonstrates that threading is a very desired technique for modern applications, especially for those that process large amounts of data, handle multiple web requests and are used by regular people not related to the IT industry.