CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

Department of Information Engineering

Diploma Thesis Abstract

Asynchronous and parallel programming in .NET framework 4 and 4.5 using C#

Author: Milan Manasievski

Supervisor: Ing. Jiří Brožek, Ph.D

Prague 2015

Summary

In this diploma thesis the author will elaborate on asynchronous and parallel programming in the .NET framework version 4 and version 4.5. The aim of this thesis will be to prove and provide better insight on the task-programming model that Microsoft introduced and compare different applications in terms of speed and lines of code used to write then and the differences between them using simple statistics. Using the literature gathered, the author will explain what would be the best ways to achieve parallelism on applications, write about design patterns used, and provide code snippets that will help the reader get better overall understanding of the Task Parallel Library and the benefits it gives in comparison of older methods and sequential programming.

Keywords

Parallel programming, parallel computing, .NET, C#, multicore processors.

Thesis objectives and methodology

In this part of the thesis, the author will elaborate on the methodology used and objectives that the author tries to achieve. A clear intention and aspiration will be presented and also shown how the author is planning to achieve the aim set forth and the methods that will be used in order to achieve this aim. Since this project will be about the task parallel library in .NET framework 4 and 4.5 versions, the focus of the thesis will be on the benefits of adopting the methods that this library offers and also focus on the differences between these methods and the way it was done before, more specifically, comparisons on how parallel programming was done before the release of the task parallel library and what is the difference after will be presented.
The author will try to prove that the methods that are used in the new task parallel library are easier to understand and learn and also prove that they are based on higher abstraction and in programming terms that means that programmers or developers do not need to worry about any low level details such as memory management, task execution or task priority.
The task parallel library does all this internally and most of the implementations are even hidden from the developer. Moreover, the author will try to prove that the task parallel library help us to efficiently and effectively utilize all the physical cores on a multicores architecture and do that in a manner that will not affect the execution of the program, but it will improve its responsiveness and perceived speed. At the end of this project the author will try to prove that with less code, code easier to understand, read and write will be able to write high performing application that are using 100% of all the CPU cores.

The author will also try to prove that there are differences in application designs and the creators of the task parallel library identified this and provided the consumers with different classes for writing parallel code and also making sure that the threads in the application are not sitting idle without doing anything.

A principle called SMART will be used for setting objectives for the thesis and the SMART abbreviation stands for:

- Specific

In order to reach the first point of SMART, the author needs to know exactly what is need to do in order to achieve the goal of the thesis and in this case the author will present two different types of applications, one is done using an older technology for presentation called Windows Forms however, its computationally very expensive and will make sure that the thread that is executing it have no idle time. After that the author will present different versions of the application where he will try to make it run in parallel see the difference in the result and also compare. The second application is a console based application that has no user interface, this application will know how to connect to the internet and download some data about stocks for any company that the user requests. This application does not require high CPU power, however it makes threads to go to sleep and be idle for the time of the downloading is done. With the task parallel library the author will try to prevent this.

- Measurable

The application will have built in mechanisms for timing and also for counting the number of threads that are executing in parallel, the author will try to present the number of CPU time to the reader and provide clear and concise differences between the different versions of the applications and also the benefits from adopting a certain approach over the other.

- Achievable

The first application that will be presented on this thesis will be based on a Mandelbrot set and in the most basic sense the Mandelbrot set is a group of numbers that display a certain unusual properties and as per definition the Mandelbrot set is the visual representation of an iterated function on the complex plane. In this application, deeper levels of the Mandelbrot set will not be explored as it can be quite complex and its computation can take even several minutes. For the application, for each pixel on the screen the program will check if that particular pixel is within the Mandelbrot set or not, as an input the program will give a random number between -2 and 2. The second application will be slightly simpler and the user who will be running the application will only need an Internet connection. There are classes in the .NET framework that know how to connect to the Internet and download some content from a certain website.

- Realistic

The objectives set forth must also be realistic, in this term the author needs to think in terms of time, resources and skills. Both application to write and the actual thesis to write will be time consuming and challenging, however if everything starts on time, probably four or five months before, the author will be able to write the necessary applications and before doing that the theoretic part should be already done. Since the author is working as a web developer in a company in Prague, Czech Republic and also covered most of the material in the Czech University of Life Sciences, safely can be assumed that the necessary skills are available for utilization.

- Time constrained

The literature has to be gathered well before starting the thesis and that should be around a year before starting to actually write anything. After the literature is gathered, which will consist mostly of books, the author will start classifying them and start writing the theoretical part. The theoretical part will be based on parallel programming in the .NET framework and while writing and researching additional knowledge can be accumulated and this will be beneficial for the applications needed for the practical part of this thesis. Last month before due date, everything should be pretty much finished and finalized. This time will be scheduled for last consultations with the thesis mentor to make sure that the author is on track and doing the things right. Last two weeks, the author will go throughout the literature once again and make sure the thesis is formatted correctly, its in presentable form and ready for submission.

Conclusions

Throughout this diploma thesis the author has explained about the new task parallel library in .NET framework 4 and 4.5, the author has also compiled a theory based on newly released material in form of books and articles on the internet. As it was mentioned and seen the task parallel library or the TPL has been around in the .NET framework since version 4 and its design to make it easy for us to work with threads and asynchronous operation so consumers of the TPL can run work in parallel or asynchronously and not to worry about partitioning work and scheduling threads and some of the low level details that they had to worry about in the past.

The task parallel library is both good and provides high level abstractions for parallel programming and also for asynchronous programming and as it was mentioned in the theoretical part these two notions are related to each other but they are different and the applications that the author built during the practical part and the theory explained this. From the task parallel library it was explained how to create a task that represents a unit of work in the .NET framework, how one can start a task for that unit of work to start executing, also it was elaborated on how to create and start a task at the same time without making an explicit call to the Start() method.
On top of that, the author explained how a user can cancel a long-running task in case its taking too long with the support of the task parallel library and in case a task is cancelled, the thread that was assigned to that task is return to the thread pool and is ready for some additional work that will be assigned by the operating system. The author also elaborated on exception handling and saw that there is not much difference in this area when it comes to synchronous code, the easiest way is to wrap the un-safe code into a try-catch block and if something goes wrong the exception will be caught and handled gracefully. Since, it is not guaranteed for the task to start immediately when the user requests, there is a way to specify a priority to a task saying that this task has high priority and the task scheduler will be able to start that particular task first and make all the others wait. In the part where the theory was presented, we have seen the best design techniques to write parallel applicants that are responsive and not hanging or crashing the user interface.

Moreover, the author looked into the most common types of parallelism and also showed implementation of some of them in the practical part. Some built in data structures as presented as well that allow concurrent work and are thread safe and when talking about thread safe operations and data structures, its important to note that there are some dangers of parallel programming and dangers of concurrent work. A developer must be very careful in implementing parallel code since quite few subtle bugs can be introduced in the code and some of the dangers and the bugs were also presented in this thesis. Moreover, the author explained when its good idea to use the task parallel library and when its not and also when it can help and when it cannot, in fact in some cases it can hinder performance.

In the practical part of the diploma thesis, two very different applications were presented that are making use of the parallel and asynchronous programming model. The first application that it was mentioned was the Mandelbrot set application that was drawing the Mandelbrot image on the screen for the user to see. The application is computationally very expensive and as it was shown before without using the task parallel library the application was running on a single core and it was finished with drawing the image in about twenty seconds and after doing small changes and made the application run in parallel it was noticed significant improvement on performance for just few lines of extra code written. The application was using 100% of the CPU power and finished in five seconds whereas in the sequential version it was using only 30% of the CPU power. The measurements are done directly in the application using some of the features of the .NET framework. Also with experimenting the author has ascertained that the best pattern for parallelism in this application was the “Embarrassingly parallel” where the application is firing off tasks for each of the loops and the operating system is assigning threads to those tasks accordingly. In the second application, the history stock application, the same methods for parallelism cannot be employed there and since the computation that the application is doing is not expensive but for best performance it still required creation of additional tasks. Note that in case the .NET framework thinks that no thread is required for particular work it will not assign a new thread, that’s the beauty in the task parallel library, it takes responsibility of the low level code that consumers do not need to write. In the history stock application we created separate tasks for the three methods that were downloading data from the Internet and the first one to finish was the winner. The results from the winner task were then displayed to the user to see. In the previous versions of the application there was not any type of parallelism and the call to the websites was a blocking call where the user had to wait and the thread had to sleep. The previous version did not support multiple users and was not making use of all the cores on the machine that the user might have and even thought the user might have a single core, the .NET framework will coordinate the work in such as way that no extra threads are created and no unnecessary work is being done in the background.

Therefore, bottom line is that the task parallel library can be implemented to multicore but also in single core architectures. In the multicore architectures consumers can take advantage of the parallel programming and on a single core machines, the asynchronous programming model can be used where a thread is not sitting idle waiting for an I/O operation to complete but its listening for new requests and doing some other work.

Bibliography

Watson, B. (2014). *Writing High-Performance .NET code.* Norfolk, USA: Ben Watson.

Campbell, C., Johnson, R., Miller, A., & Toub, S. (2010). *Parallel Programming with Microsoft® .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures.* Redmond, USA: Microsoft Press.

Cleary, S. (2014). *Concurrency in C# Cookbook.* Sebastopol, USA: O'Reilly Media.

Esposito, D., & Saltarello, A. (2014). *Microsoft .NET - Architecting Applications for the Enterprise.* Redmond, USA: Microsoft Press.

Herlihy, M., & Shavit, N. (2012). *The Art of Multiprocessor Programming.* Burlington: Morgan Kaufmann.

Mattson, T., Sanders, B., & Massingill, B. (2013). *Patterns for Parallel Programming* (1st Edition ed.). Boston, USA: Addison-Wesley Professional.

Pacheco, P. (2011). *An Introduction to Parallel Programming* (1st Edition ed.). Burlington, Massachusetts, USA: Morgan Kaufmann.

Sharp, J. (2013). *Microsoft Visual C#.* Redmond, USA: Microsoft Press.

Skeet, J. (2013). *C# in Depth* (3rd Edition ed.). Greenwich, USA: Manning Publications.

Razdan, S. (2014). *Fundamentals of Parallel Computing.* Oxford, UK: Alpha Science International Ltd.