**CAB 401**

**High Performance and Parallel Computing**

**The parallelization of the Program**

**Digital Music Analysis**



**John Wang**

**N9475389**

Contents

[Sequential Application 3](#_Toc528433616)

[Potential Parallelism 6](#_Toc528433617)

[Parallelization Programming Language 9](#_Toc528433618)

[Timing and profiling results 12](#_Toc528433619)

# Sequential Application

General Overview

The purpose of this report is to demonstrate parallelization of the C sharp sequential program “Digital Music analysis” provided on blackboard. The program was designed for the purpose of assisting users to learn new music without the presents of a professional music teacher, where it provides feedback based on the accuracy of the note played by the user. The program starts by prompting a window that allows the user to select the input wav file and xml file, and outputs the frequency, octaves and staff of the selected music piece.

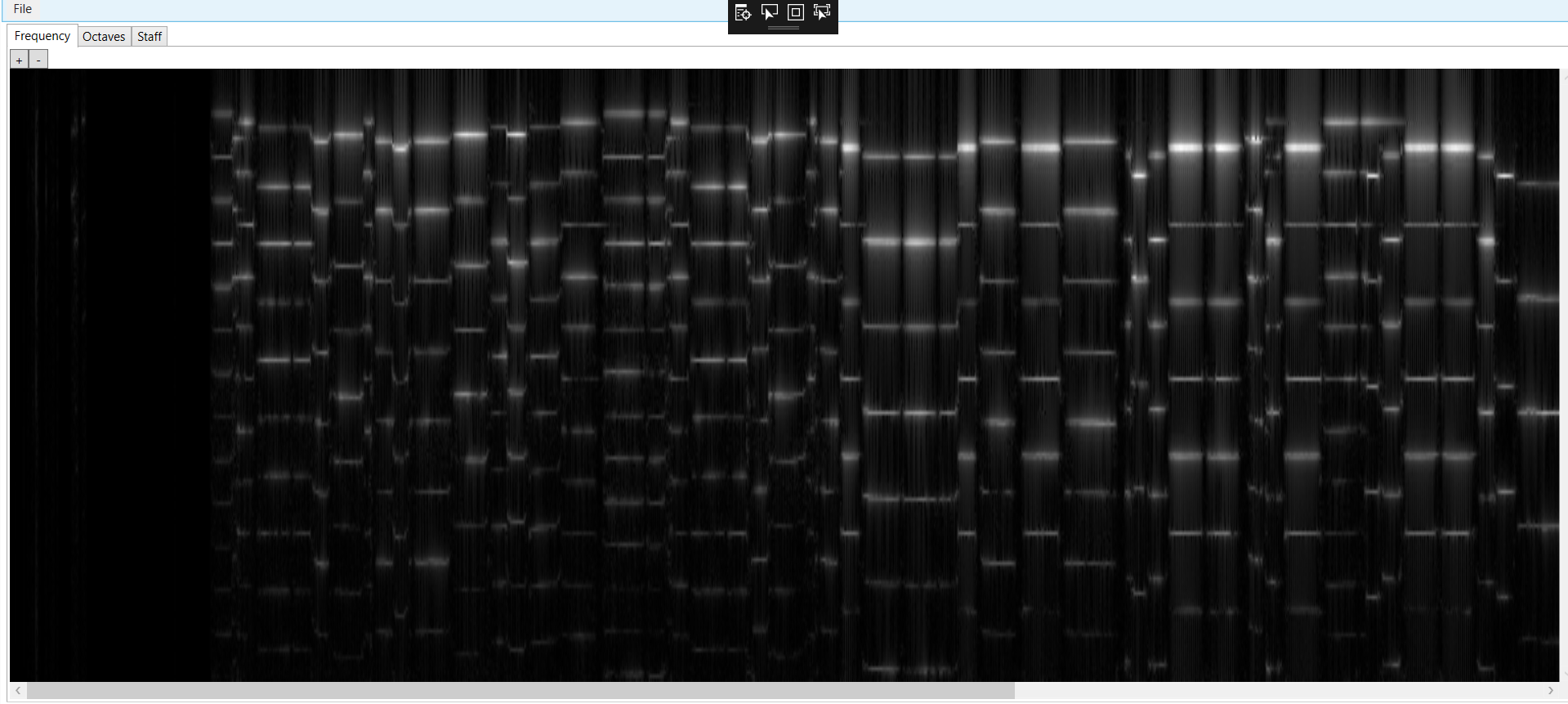
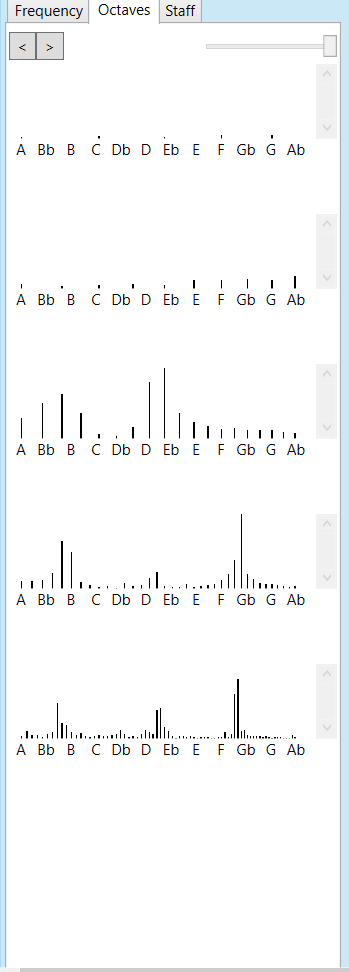
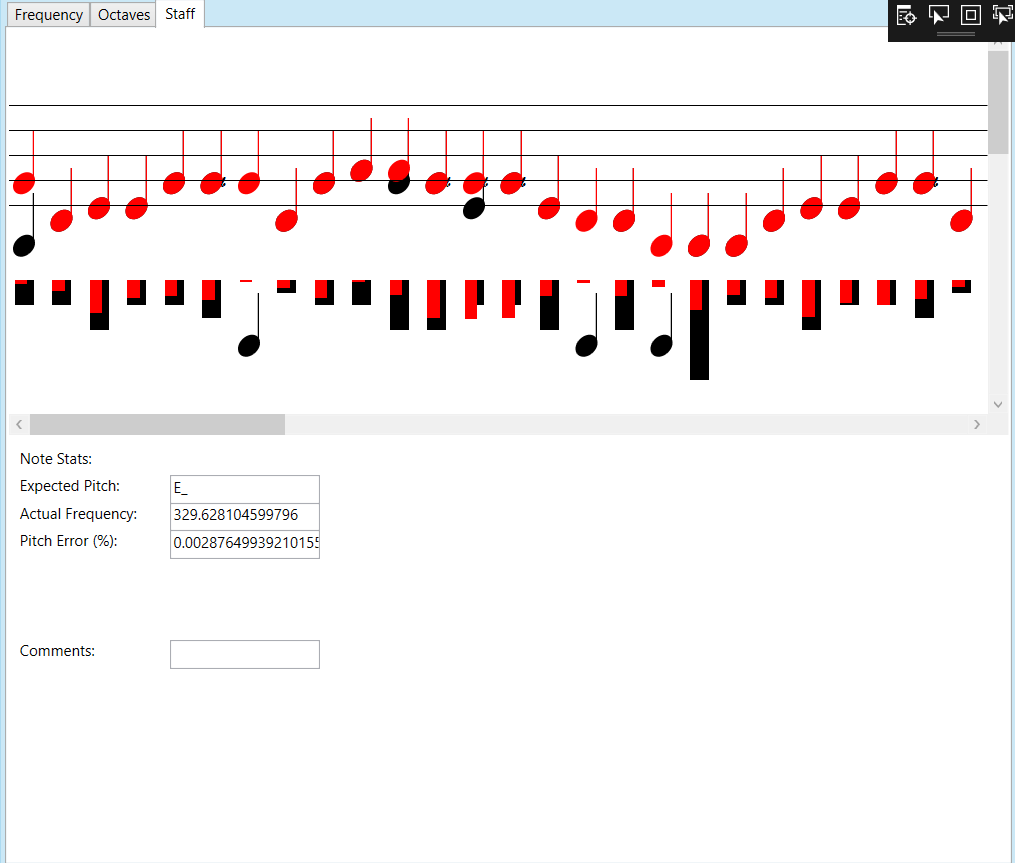


Figure above is the frequency generated by inputting the wav file, the bandwidth is the duration of the note played by the user. The height of where the white line occurs symbolizes the value of frequency, in other words it shows the combination of the keys played. The frequency above is obtained by dividing the input music piece into small intervals, and apply a technique called short term Fourier transform where it allows to convert the time domain data into frequency domain.



The figure on the left, is the outputted octaves of the music piece. It allows user to visualize the note that were played during each point of the time of the music piece inputted. This Octave diagram also allows user to clarify the accuracy of the note they played, by checking if the frequency spike occurred on the note they were meant to be played at.



The figure above is a screenshot the one of the output staff, where black notes is where the note supposed to be played, and the red notes is the note that were played by the user. In the section below, provided a literature feedback for the user, it shows user, the expected pitch, actual frequency that were played, and the pitch error of how far the notes were off. Besides that, it also provides comments whether it tells the user that the note played were too sharp or too flat etc.

Code Analysis

The chosen sequential program Digital Music Analysis uses the technique of object-oriented programming which contains mainly four classes *‘musicNote.cs’*, *‘noteGraph.cs’,* *‘timefreq.cs’, ‘wavefile.cs’* and an Extensible Markup Language (XAML) named *‘MainWindow.xaml.cs’*. Most of the functions from other class are called inside the *‘MainWindow.xaml.cs’*.

*‘MusicNote.cs’*

The class that determines the pitch error, pitch frequency as well as checking for flat notes. Besides that, it also determines the staff positions. The results mentioned are calculated by frequency and duration of the input. The class also contains public enum variable like notePitch, integer pitch, double duration, Boolean flat, double rror, int staffPos, int mullt and double frequency.

*‘noteGraph.cs’*

Class that used to set the height of the histogram in octaves. The class only contained two main function ‘noteGraph’ and ‘setRectHeights’. ‘noteGraph’ is the function to store current base frequency value and heights into the public variable. Furthermore, ’setRectHeights’ is the function that sets the height of the histogram based on the base frequency.

*‘timefreq.cs’*

Class that exist for the purpose to perform Short Term Fourier Transform on the input audio file, it allows the data to be converted from a time domain into frequency domain. It is mainly used to determine the sinusoidal frequency and phase content of local section of a signal as it changes over time. (Allen, 1977) The class contains two function ‘*timefreq’* and ‘*stft’,* ‘*timefreq*’function is used to extract complex array and w sample data to perform Short Term Fourier Transform. ‘*stft*’ function where the calculation of Short-Term Fourier transform happens.

‘*wavefile.cs’*

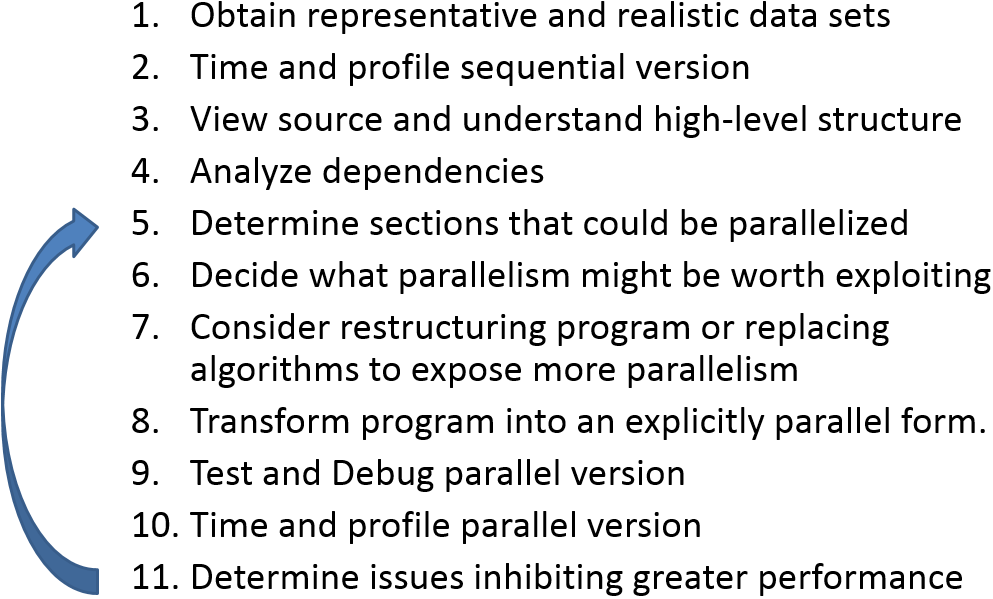
For the purpose of reading data from the wav file, the class mainly uses C# binary reader to obtain data from the input wav file. Data like *‘ChunkID’, ‘ChunkSize’*, Format etc. can be accessed from this class.

*‘MainWindow.xaml.cs’*

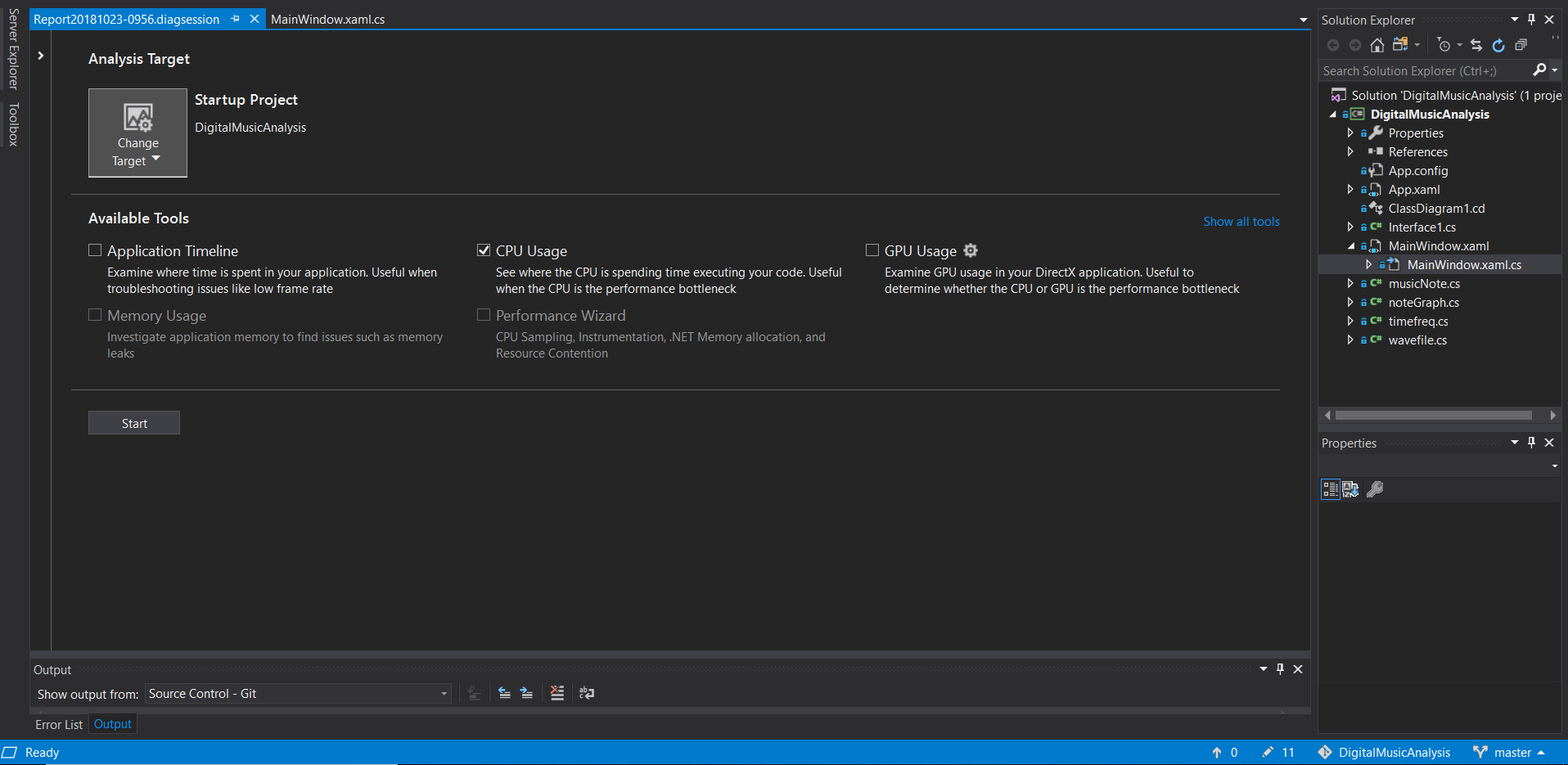
The main class where all the function from other classes and variables were initialized. The public method MainWindow() plays the major role in the class, it controls the overall flow of the program. It starts by initializes the essential components, then it prompts the user to input the wav file and xml file. After that the function initializes a thread to make sure the song is playing while the slider gets updated. It then loads the wav file the user inputted previously, to prepare for performing Short Term Fourier Transform. After, the class goes through a ‘*loadImage*’ function, where it essentially loads a time frequency graph of the input wav file, the image will be displayed onto the graphical user interface frequency tab for users to visualize. After *‘loadImage’*, the *‘loadHistogram’* gets called, the function produces a similar characteristic as the *‘loadImage’* function, instead it produces a histogram, the histogram is used in octaves tap. Lastly, the *‘onsetDetection’* function is used to produces the visualization in Staff.

# Potential Parallelism

The chosen program Digital Music Analysis was built on many nested for loops, this is an indication the program’s execution time and CPU usage can be reduced through parallelization. The parallelism method used in this report is shown below, the frame work was taught within the lecture, it followed an eleven steps process.



The first step of the framework is to obtain representative and realistic data sets, in this case Jupiter.wav and Jupiter.xml from blackboard was chosen as the input dataset. The second step of the framework is to time and profile the sequential version of the program. This step can be complete by using the visual studio built-in tool, the performance profiler. The tool can be found under the Analyse tap at the top of the window.



The visual studio profiler allows user to measure the CPU and GPU usage of each function inside the code. In this case there are no GPU usage will be associated, therefore only CPU usage will be selected in this case.

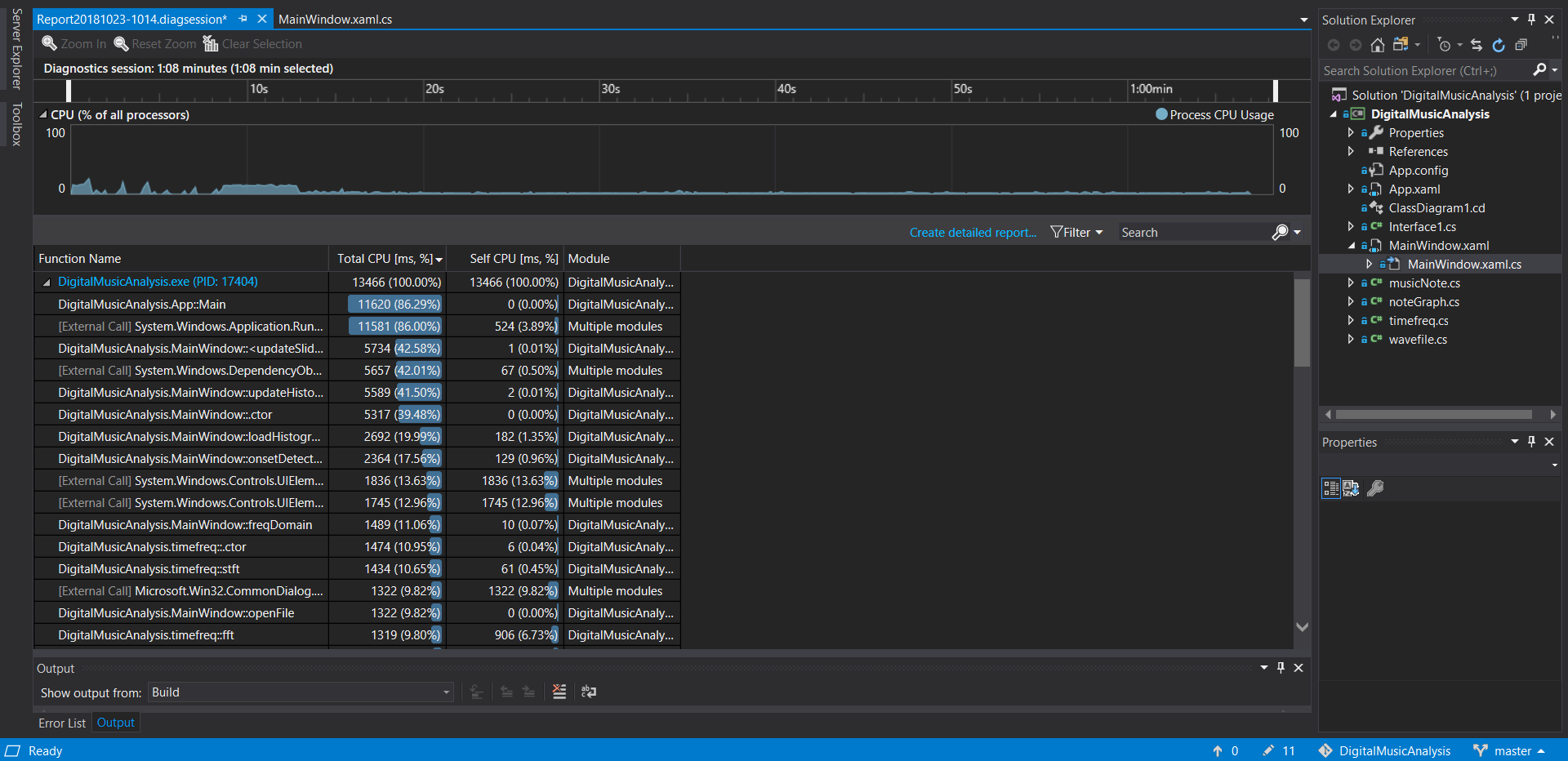
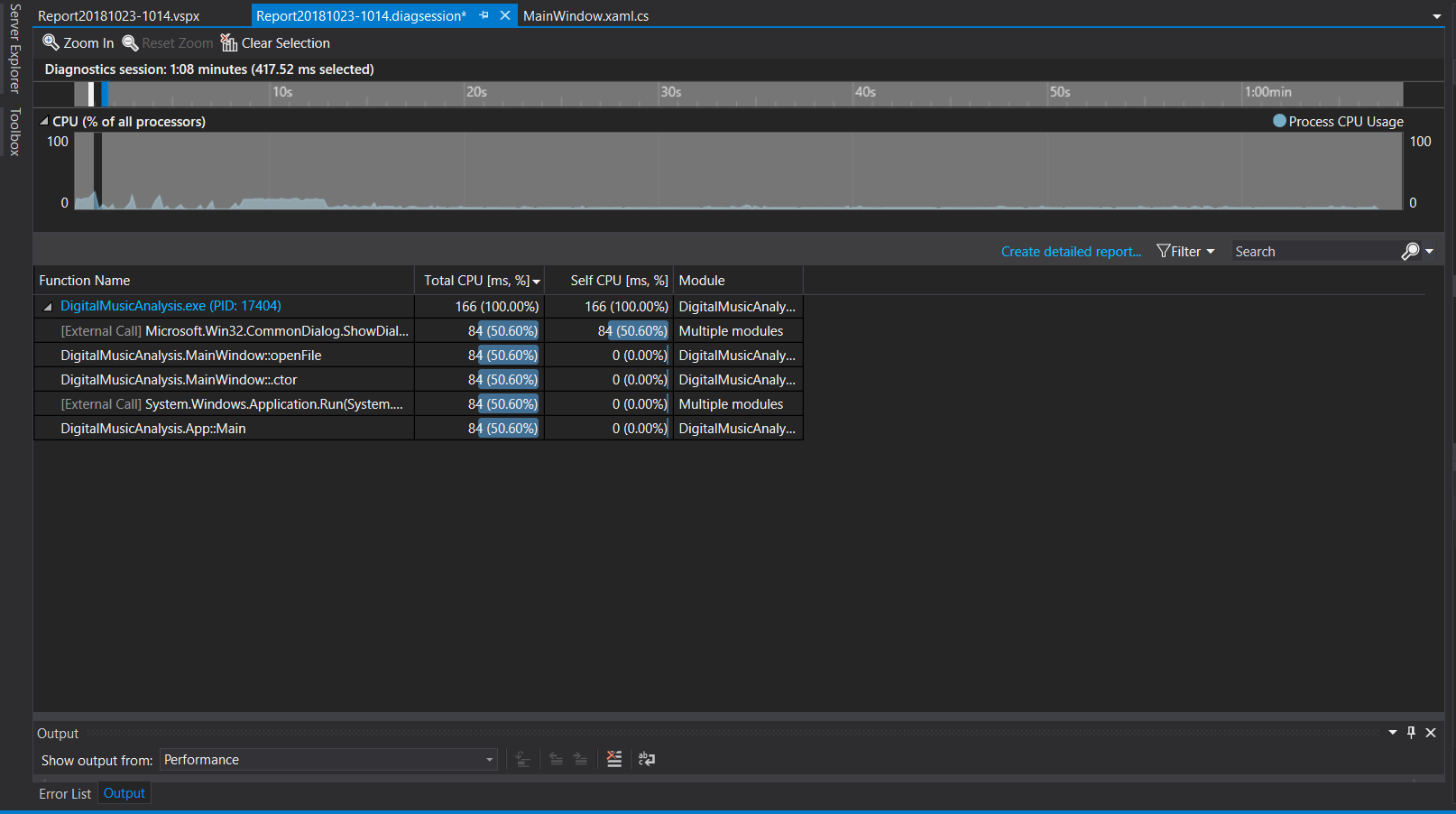
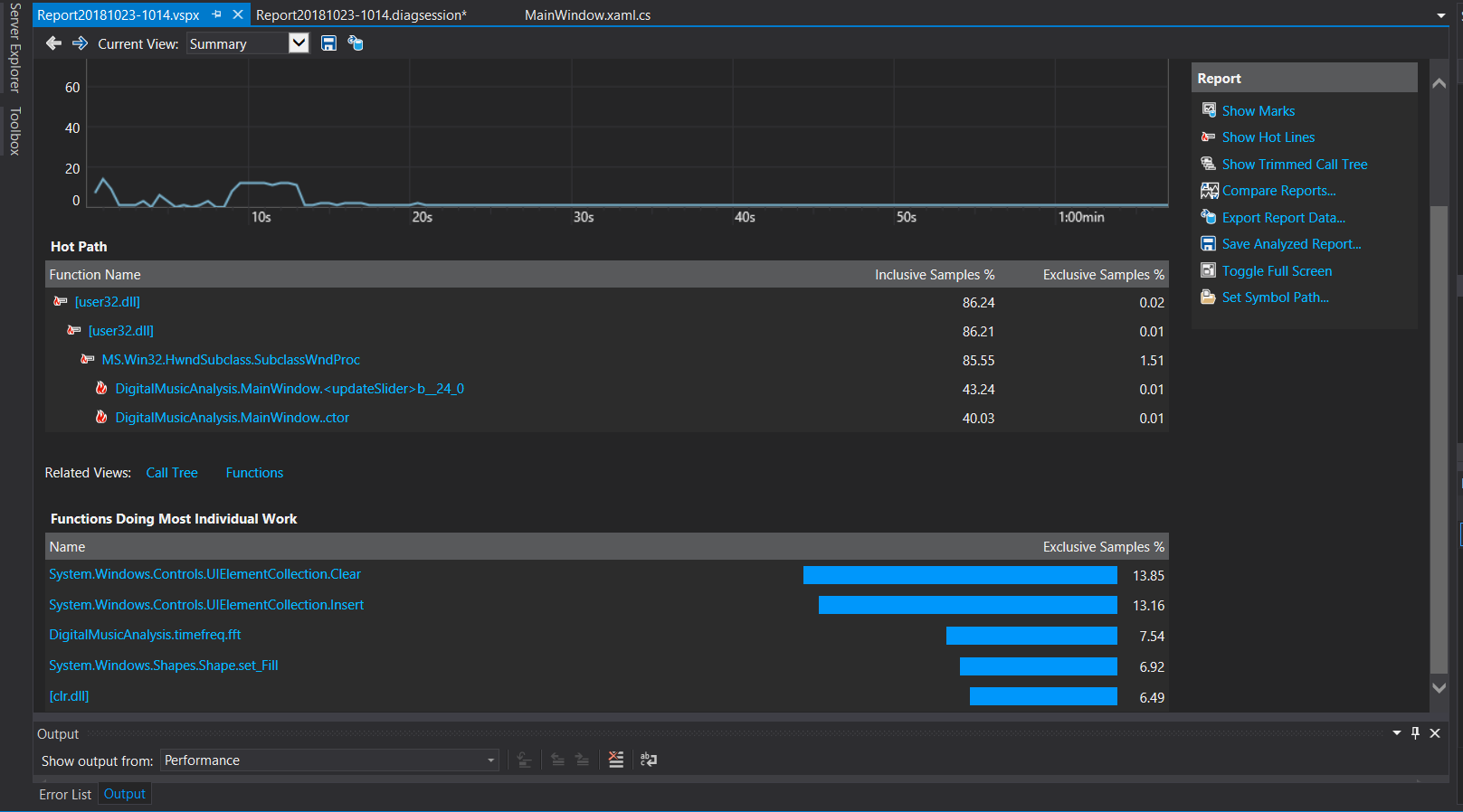


Figure above is the CPU usage report generated from visual studio profiler. Upon observation *‘DigitalMusicAnalysis.App::Main’* and *‘System.Windows.Application.Run’* has the highest total CPU usage both recorded a high usage rate of 86%, total CPU time is the time spent executing code in this function and in function called by this function. *‘DigitalMusicAnalysis.App::Main´* is where all the function from other library gets called, so undoubtedly it will record the highest CPU usage rate. *‘System.Windows.Application.Run’* is pretty much an initialization to GUI, it gives GUI its “life”. Moreover, the functions within the loop that takes the highest usage is *‘updateHistogram’* 41.50%, *‘loadHisotgram’* 19.99%, *‘onsetDectection’* at 17.56% and *‘freqDomian’* at 11.06%.



By toggling Diagnostics session at the peak of the CPU usage occurs at where user selects the input file as *‘openFile’* function was called. Upon on further inspection, a detailed report of the CPU usage was created as shown below.



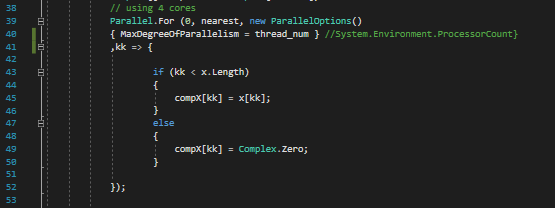
The figure above is the detailed report generated, from the “generated detailed report” function in visual studio, the report offers information about the functions that’s doing most individual work. Base on observation*, ‘fft’* from *‘timefreq’* class does most of the work. Therefore, it can be concluded, *‘updateHistogram’, ‘loadHisotgram’, ‘onsetDectection’, ‘freqDomian’ and ‘fft’* are the place where the parallelisation will be focused on*.*

# Parallelization Programming Language

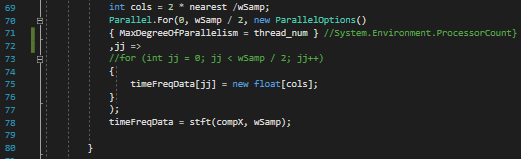
The programming chosen for this report is ‘Task Parallel Library’, is a set of public types and APIs in System.Threading and System.Threading.Tasks namespaces. The purpose of the development of the library is to reduce the complicity of adding parallelism and concurrency to applications. The library is initialized by putting ‘using System.Threading.Tasks.Parallel ‘ on the top of the every code that the user want to parallel. The reason of selecting such language is because is easy to implement, it is built within visual studio, and does not require any additional downloads. In this case, the constructor that will be used is the *‘Parallel.For’* with *‘MaxDegreeOfParallelism = thread\_num’* where it will only parallel through the number of cores specified, in this case It is four.

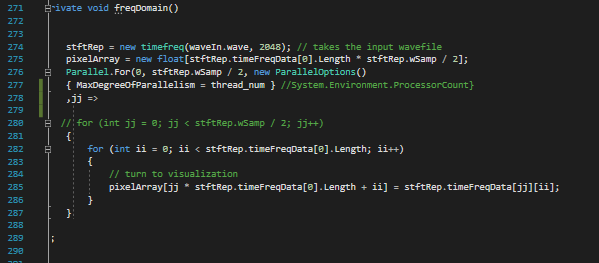


After following the frame work shown previously and try out and error, few of the for loops from the program was parallelized. First for loop that was parallelized was the loop inside the ‘*timefreq.cs*’ under the function ‘*timefreq*’.

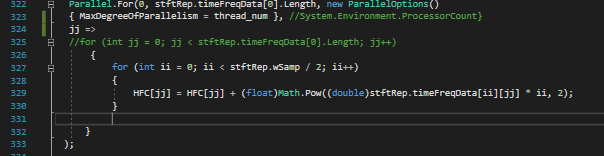


Another parallelized loop is placed within the same function at the bottom.

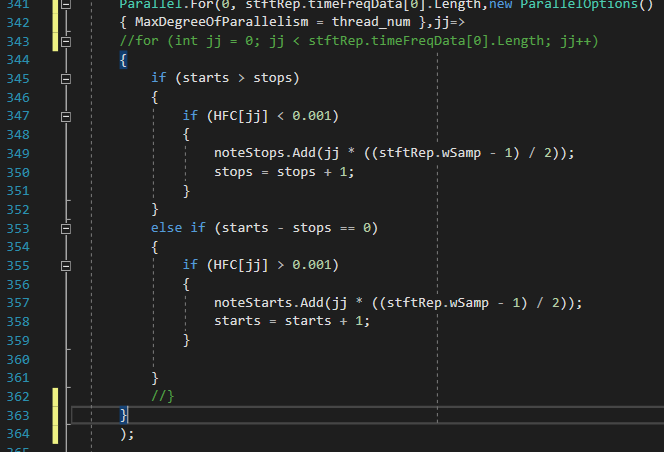




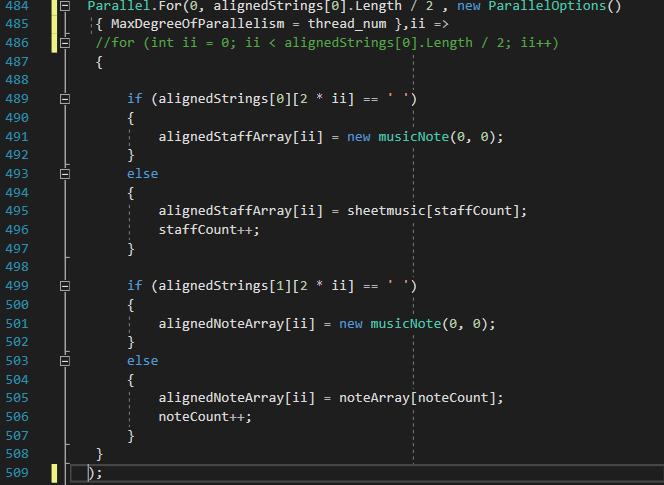
Parallelization within the *‘MainWinodw.xaml.cs’*  under the freqDomain function.



Parallelization within the *‘onsetDectection’* method.



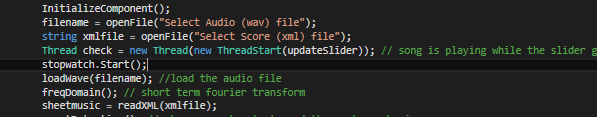
Parallelization within the *‘onsetDectection’* method.



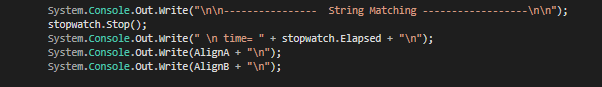
Parallelization within the ‘onsetDectection’ method.

# Timing and profiling results

To measure the execution time, a package must be implemented, this can be achieved by placing *‘using System.Diagnostics’* on the top of the program. By placing this, it will allow the access of the stopwatch class. The timer will be initialized as stopwatch as a global variable as it needs to be accessed throughout the entire program. 



The stopwatch will start counting after the file selection. The reason of this is that, is because during file selection, users takes time to select the input file, this will cause inaccuracy in result. As the selection time for each time will be different. Furthermore, *‘stopwatch.Stop’* will be placed under string matching as shown below.



The reason of stopping the stopwatch at this location, is due to the fact that, from the overall flow of the program this is where the analysis ends. By placing it over there, it enables to obtain a more accurate result.

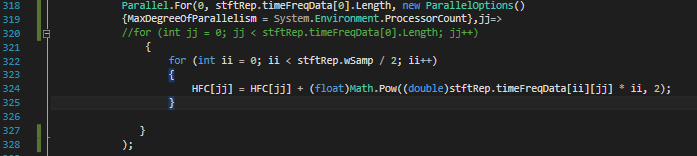
After executing the program, the sequential program recorded 2.69 second execution time.

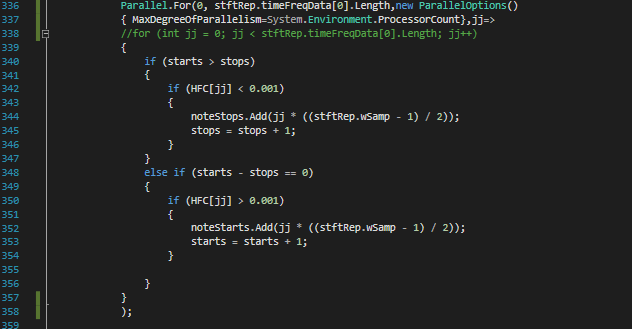


However, the parallelized version of the program returned a result of 4.69 seconds of execution time.



The reason that the sequential version recorder a better execution time was because the sample size in this case is relatively small, therefore the time on executing each iteration is very little. On the other hand, creating and managing different thread takes much longer time. Parallel programming is way more efficient when it comes to large dataset. One way to solve this problem is to eliminate those parallelized for loop that iterates through small dataset. For example, the two shown below:





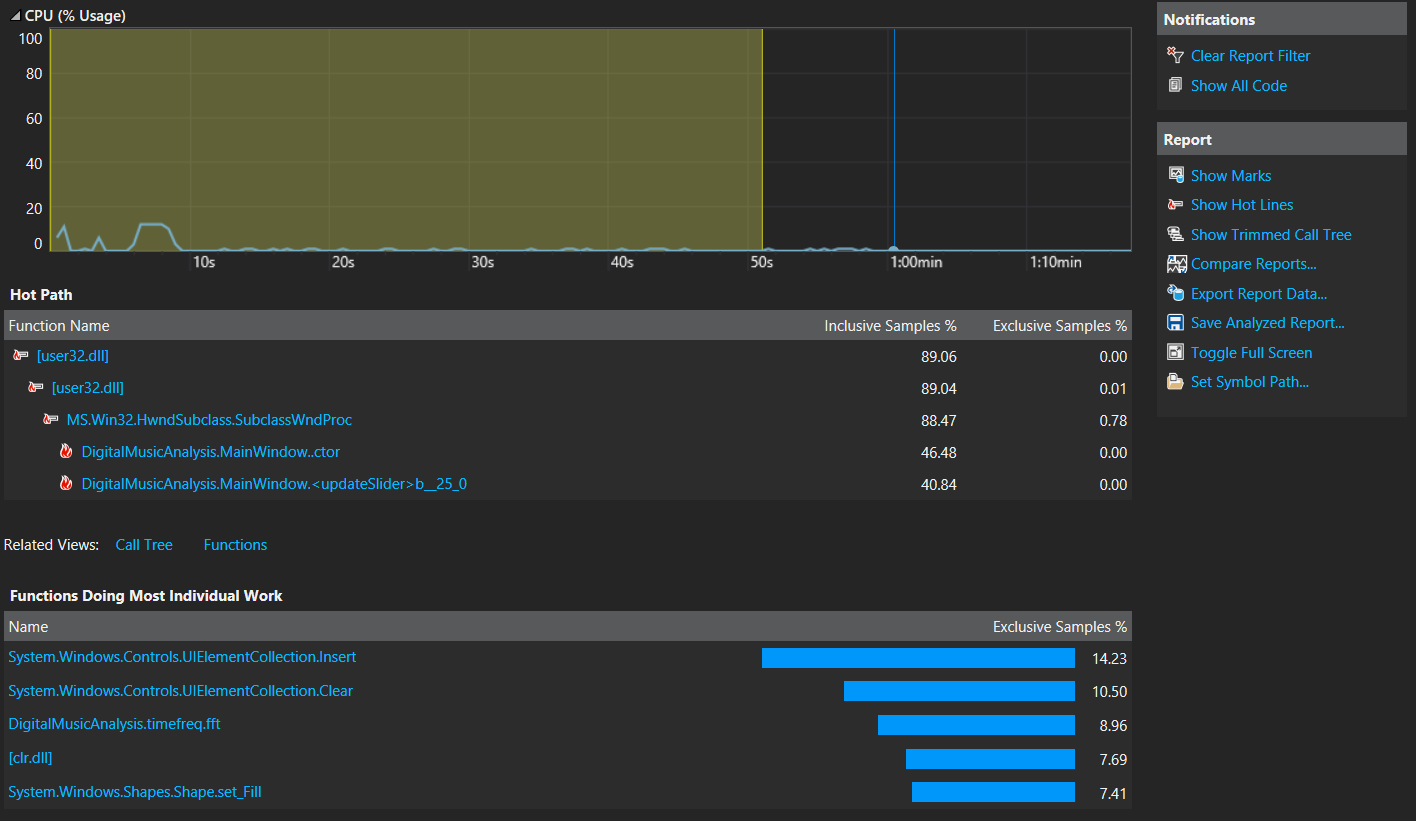
After eliminating those two parallelized for loop the execution time got sped up to 02.56 second.

The result obtained is slightly faster than the sequential program.

The reason that the result of parallelization isn’t as significant is due to the reason that parallelization works better with a larger dataset as mentioned previously. Furthermore, it is worth to mention, FFT function within timefreq can be potentially parallelized as well however it recorded an execution time of



Figure below is a detailed report of CPU usage generated from sequential program for the first 50 seconds.



The figure below is a detailed report generated from running the parallelized version program in the first 50 seconds.



Base on comparison, under the functions doing most individual work section, it is clearly evident that there is slight drop in CPU usage in each of the function. This proves that by implementing parallel programming it will increase the performance of the program.

# Compilers, software, tools and techniques

Upon close inspection of the program, there are redundant existents of two FFT function both within ‘*MainWindow.xmal.cs*’ and ‘*timefreq.cs*’. This will be further mentioned in the upcoming report section performance problem/ barriers.