

1/11/2021

# CAB401

# Parallelization

# Project

Digital Music Analysis

Julian Zhu  
QUT YEAR3

## Contents

Explanation of the original sequential application.....	2
How does Digital Music work .....	2
Potential parallelism.....	3
Parallelism for timefreq.cs .....	4
Parallelism for onsetdetection() .....	6
Abstractions or programming language used in synchronization .....	7
Result .....	7
Timing.....	7
Result comparison.....	7
Compiler, software, tools and techniques. ....	12
Compiler .....	12
Parallelisation framework and Hardware.....	12
Techniques .....	13
Barriers.....	13
Explanation Code.....	14
timefreq.cs.....	14
mainwindow.cs .....	17
Reflection.....	18

## Explanation of the original sequential application

The application that is used for parallelisation for this project is “DigialMusic”. The application is downloaded on the QUT’s blackboard under assessment. The “DigialMusic” application is written in the C# programming language. The purpose of the application is to help beginner users identify which note is currently playing according to uploaded files. The application shows which note the student played and where the note should be playing at. The application also visualises the frequency in black and white on the first page of the GUI.

Once the user runs the application, the first action that the user must perform is to upload two types of files, “.wav” and “.xml”. The “.wav” is the file that the application will play through, and the “.xml” file is the provided file (the file that the user played) for the application to compare. There are in total three tabs from the application, “frequency”, “octaves”, and “staff”. The “frequency” tab shows the audio frequency in black and white. The “octaves” tab shows which note is currently playing. Lastly, the “staff” tab shows whether if the student is playing correctly or not. The red notes mean that the note is incorrect, and the back notes mean the right note. The user can also see the percentage of pitcher error if the user hovers over the notes in the tab.

## How does Digital Music work

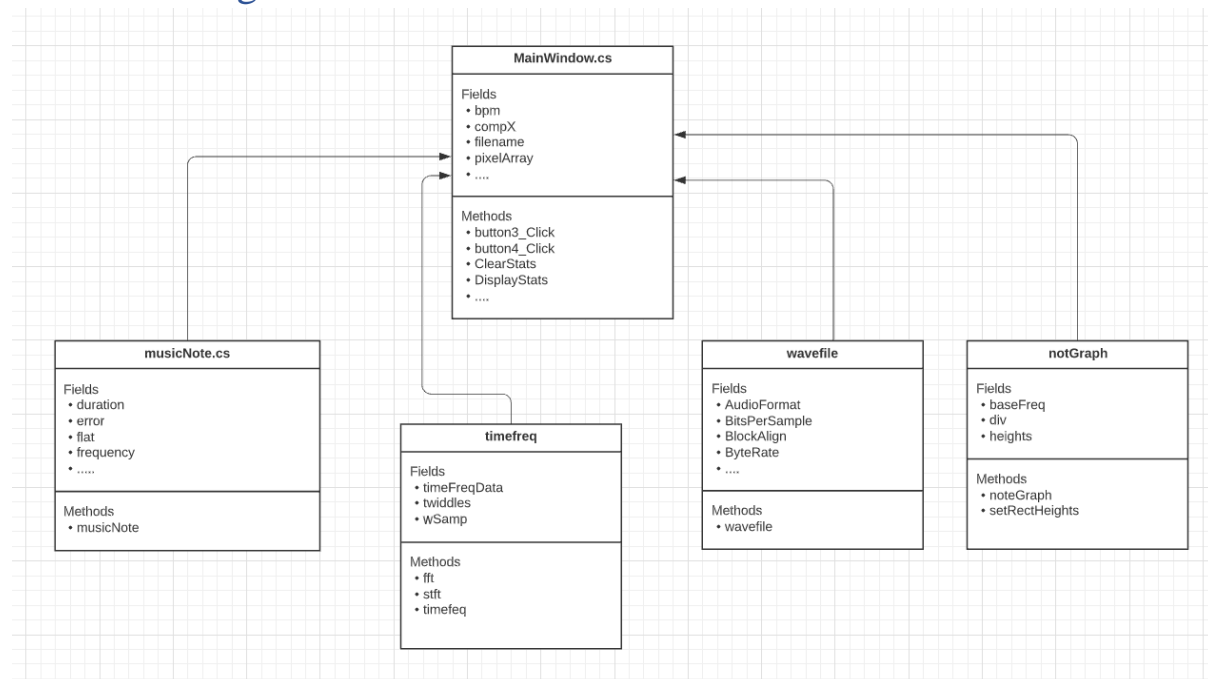


Figure 1 (class diagram)

The application contains multiple classes, those are musicNote.cs, noteGraph.cs, timefreq.cs and wavefile.cs. The musicNote class contains several logical switch cases, which helps to check the note's position; it also includes pitch of the note, duration, flat note, error, staff position, and frequency. The class noteGraph is used to set the height of the note. The third class, timefreq, is used to analysing frequency. Furthermore, The class contain complex algorithms, such as FFT(Fast Fourier transform) and STFT(Short-time Fourier transform). The last class wavefile is binary reader, the class can read and information from the file, including chunk size, format, audio format, number of channels, sample rate, byte rate, bites per sample, and many more. The main function is located in "mainWindow.xaml.cs" file. The file contains multiple functions in including loading file, processing file, and GUI design. The most important function will be analysed below.

1. “loadwave()” function

The first important function that the application will run once the user runs the application is “loadwave”. If the uploaded file is incorrect, the application will return an error message telling the user; else, the function will convert the uploaded file into an array.

2. “freqDomain()” function

The next function that will run after “loadwave()” function is “freqDomain()” function. This function will lead to a class called “timefreq”. The “timefreq” class has two functions called “stft” and “fft”, which stands for Short-time Fourier transform and Fast Fourier transform.

3. “readXML” function

The “readXML” function will read the uploaded xml file, and convert them in to lists.


4. “onsetDetection”

The “OnsetDetection” function will check whether if the note starts playing and when did it stop.

The rest of the function such as “loadImage()”, “loadHistogram()”, and “PlayBack()” can be described by its function name.

## Potential parallelism

The method of parallelism will be used according to the Parallizaion Methodology table. The first four steps have already been done; the starting step will be to determine sections that could be parallelized by using a performance profiler.

1. Obtain representative and realistic data sets
  2. Time and profile sequential version
  3. View source and understand high-level structure
  4. Analyze dependencies
  5. Determine sections that could be parallelized
  6. Decide what parallelism might be worth exploiting
  7. Consider restructuring program or replacing algorithms to expose more parallelism
  8. Transform program into an explicitly parallel form.
  9. Test and Debug parallel version
  10. Time and profile parallel version
  11. Determine issues inhibiting greater performance
- 

*Figure 2(Parallizaion Methodology table from Wayne's Lecture)*

The Performance profiler will capture the CPU Usage and give the user a great view of which part of the program is doing most of the work. Once I ran the performance profiler, it was clear that most calculation and time is on `onsetDetection()` and `freqDomain()`, with 28.06% and 27.68%. According to the profiler, it is believed that paralleling those two classes will significantly reduce the amount of time it takes, which result in speeding up the whole application.

Function Name	Total CPU [unit,...]
DigitalMusicAnalysis.exe (PID: 37988)	7364 (100.00%)
[Native]	7355 (99.88%)
DigitalMusicAnalysis.App.Main()	5908 (80.23%)
[External Call] System.Windows.Application.Run()	5812 (78.92%)
DigitalMusicAnalysis.MainWindow.ctor()	5229 (71.01%)
DigitalMusicAnalysis.MainWindow.onsetDetection()	2066 (28.06%)
DigitalMusicAnalysis.MainWindow.freqDomain()	2038 (27.68%)

Figure 3 (Performance Profiler Capture)

## Parallelism for timefreq.cs

According to the performance profiler, 27.44% is at the class `timefreq`, and if we examine the class, we can see that out of 27.44%, there is 26.82% inside a function called `stft`.

207		
268		<code>private void freqDomain()</code>
269		<code>{</code>
270	2021 (27.44%)	<code>stftRep = new timefreq(waveIn.wave, 2048);</code>
271		<code>pixelArray = new float[stftRep.timeFreqData[0].Length * stftRep.wSamp / 2];</code>
272		<code>for (int jj = 0; jj &lt; stftRep.wSamp / 2; jj++)</code>
273		<code>{</code>

Figure 4(Workload from timefreq.cs)

1975 (26.82%)	50	
	51	<code>timeFreqData = stft(compX, wSamp);</code>
	52	

Figure 5(Workload from stft method)

By viewing inside of `timefreq` class, the first two loops take a bit of time, the first one is the for loop with the if-else statement, and the second one is just a for loop. Since there are no data dependencies for those two loops, it makes those loops safe to parallel. The method that I'll parallel those loops is by using implicit threads because both of them are simple for loop with no data dependencies.

29		<code>Complex[] compX = new Complex[nearest];</code>
30		<code>for (int kk = 0; kk &lt; nearest; kk++)</code>
31		<code>{</code>
32		<code>if (kk &lt; x.Length)</code>
33		<code>{</code>
34		<code>compX[kk] = x[kk];</code>
35	41 (0.67%)	<code>}</code>
36	1 (0.02%)	<code>else</code>
37		<code>{</code>
38		<code>compX[kk] = Complex.Zero;</code>
39		<code>}</code>
40		<code>}</code>
41		<code>}</code>

Figure 6(First loop from timefreq.cs)

45		<code>for (int jj = 0; jj &lt; wSamp / 2; jj++)</code>
46		<code>{</code>
47		<code>timeFreqData[jj] = new float[cols];</code>
48	11 (0.16%)	<code>}</code>
49		

Figure 7(Second loop from timefreq.cs)

Implicit threads are high level libraries and language constructs built on top of more primitive threading libraries.

If we look down a little more, we can see where most of the work is done, which is a line calling the stft method.

1940 (28.86%)	50 51 52	timeFreqData = stft(compX, wSamp);
---------------	----------------	------------------------------------

Figure 8 (Workload for stft method)

To parallel this line of code, we will require to check the method's structure of stft. First, identify whether it is safe to parallel, then check if it requires changing the algorithm or transforming it to speed up.

Inside stft method, there are two for loops, one is a simple for loop, and one is a nested for loop.

```
for (ll = 0; ll < wSamp / 2; ll++)
{
    Y[ll] = new float[2 * (int)Math.Floor((double)N / (double)wSamp)];
}

Complex[] temp = new Complex[wSamp];
Complex[] tempFFT = new Complex[wSamp];

for (ii = 0; ii < 2 * Math.Floor((double)N / (double)wSamp) - 1; ii++)
{
    for (jj = 0; jj < wSamp; jj++)
    {
        temp[jj] = x[ii * (wSamp / 2) + jj];
    }

    tempFFT = fft(temp);

    for (kk = 0; kk < wSamp / 2; kk++)
    {
        Y[kk][ii] = (float)Complex.Abs(tempFFT[kk]);

        if (Y[kk][ii] > fftMax)
        {
            fftMax = Y[kk][ii];
        }
    }
}
```

Figure 9 (loops inside stft method)

The first for loop is similar to the previous for loop; it is safe to parallel by simply using implicit threads. However, the second loop is more complex. The second loop is a loop that calls the method FFT function where the calculation is being done. The function fft is a method that contains complex algorithms, and it is not safe to parallel due to data dependencies. The data output is dependent on the complex number array, which makes this function hard to parallel. The only way to parallel fft function is to transform the algorithms so that the data no longer depend on each other. I decided to parallel the loop that calls fft, due to the complexity of the transformation (more details can be read in reflect section). The parallel method that I decided to use for this loop will be Explicit threads. First, calculate blocksize, which is the end of the second loop( $2 * \text{Math.Floor}((\text{double})N / (\text{double})w\text{Sam}) - 1$ ), decided by the number of threads. Each thread will run a part of the loop, which archives in speeding up the application. I overcame the dependencies problem by putting the complex array into each

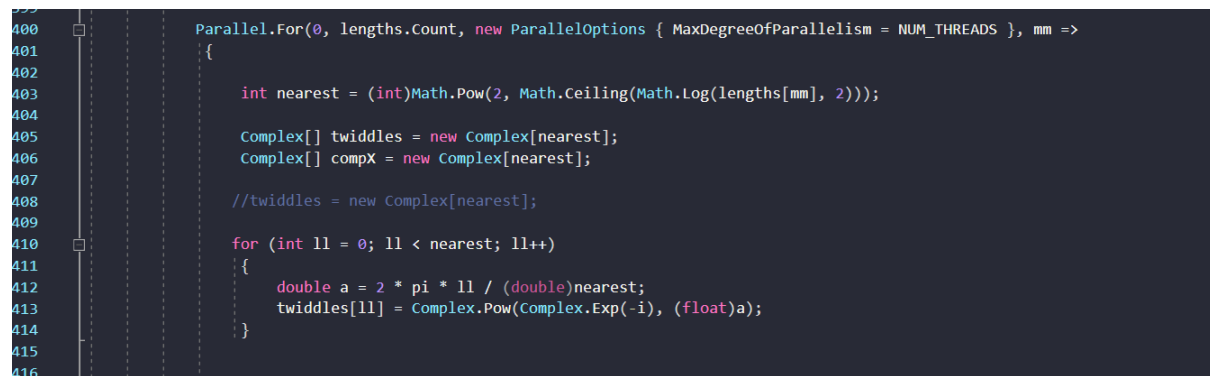
thread instead of leaving it out of the loop

(More detail can be seen in the screenshot from the Explanation section)

### Parallelism for onsetdetection()

By viewing the profiler, it is clear to see the only loop that is worth to parallel is the largest for loop in onsetdetection. There are couple variables that will require to set it to global, those are twiddles, compX, Complex[]Y, absY[], and int ll. The reason remains the same as the pervious why I put complex array in each thread, because of data dependency. Last change that will require to be make is to change the pitches list into array.

The only loop that I paralleled is the main for loop from the class, which does the most amount of work.

A screenshot of a code editor showing C# code. The code is part of a method named 'onsetdetection'. It starts with a 'Parallel.For' loop that iterates from 0 to 'lengths.Count'. Inside the loop, it calculates 'nearest' as the ceiling of the log of 'lengths[mm]' multiplied by 2, then powers of 2. It then initializes 'twiddles' and 'compX' as complex arrays of size 'nearest'. A 'for' loop follows, iterating 'll' from 0 to 'nearest-1'. Inside this loop, it calculates a value 'a' and updates 'twiddles[ll]' using 'Complex.Pow' and 'Complex.Exp'. The code is annotated with line numbers 400 through 416 on the left side.

```
400 Parallel.For(0, lengths.Count, new ParallelOptions { MaxDegreeOfParallelism = NUM_THREADS }, mm =>
401 {
402     int nearest = (int)Math.Pow(2, Math.Ceiling(Math.Log(lengths[mm], 2)));
403
404     Complex[] twiddles = new Complex[nearest];
405     Complex[] compX = new Complex[nearest];
406
407     //twiddles = new Complex[nearest];
408
409     for (int ll = 0; ll < nearest; ll++)
410     {
411         double a = 2 * pi * ll / (double)nearest;
412         twiddles[ll] = Complex.Pow(Complex.Exp(-i), (float)a);
413     }
414 }
415
416
```

Figure 10(onsetdetection loops)

There are two FFT function can be find in application. One is located at the timefreq class, and one is at the mainwindow class. Combining two FFT function into one class will not result in application speed up. For the FFT funciton to speed up will require to change FFT's alogrthim. Unforuntly, I did not manage to change the alogrthim, therefore, I left FFT untouched for both classes. Lastly, there no synchronisation method that is used in both parallzaion, this is because that the parallel library from C# has pre-synchronisation builded into.

(Screen shots of the code after parallelsion will be shown on the Explanation secetion)

## Abstractions or programming language used in synchronization

The abstraction used throughout the application is called TPL, which stands for “Task Parallel Library”. The TPL parallelism abstractions contain several public types and APIs in the System.Threading, and System.Threading.Tasks. Once it is added, the user will be able to use commands that active threadings.

## Result

### Timing

The timing is done by using C# library called Diagnostic. Once the Diagnostic has been added, the function stopwatch will be set on three parts of the application. First timer will time the whole application, last two timer are at freqDomain and onsetdetection timer, which will time their own section. Lastly, two open file commands will not be times because it depend on the user’s interaction.

1. Whole application timer with number of processors

Number of processors	Sequential Time (ms)	Paralleled Time (ms)
1	4357	5281
2	4622	3817
3	3907	2697
4	3911	2483
5	3794	2448
6	3817	2295
7	3837	2285
8	3829	2224
9	3812	2212
10	3751	2201
11	3784	2253
12	3739	2192

2. “freqdomain” timer with number of processors

Number of processors	Sequential Time (ms)	Paralleled Time (ms)
1	2177	3011
2	2474	2000
3	2043	1390
4	2045	1239
5	1917	1180
6	1965	1135
7	1978	1190
8	1999	1008
9	1958	1063
10	1935	1044
11	1972	1073
12	1924	1091

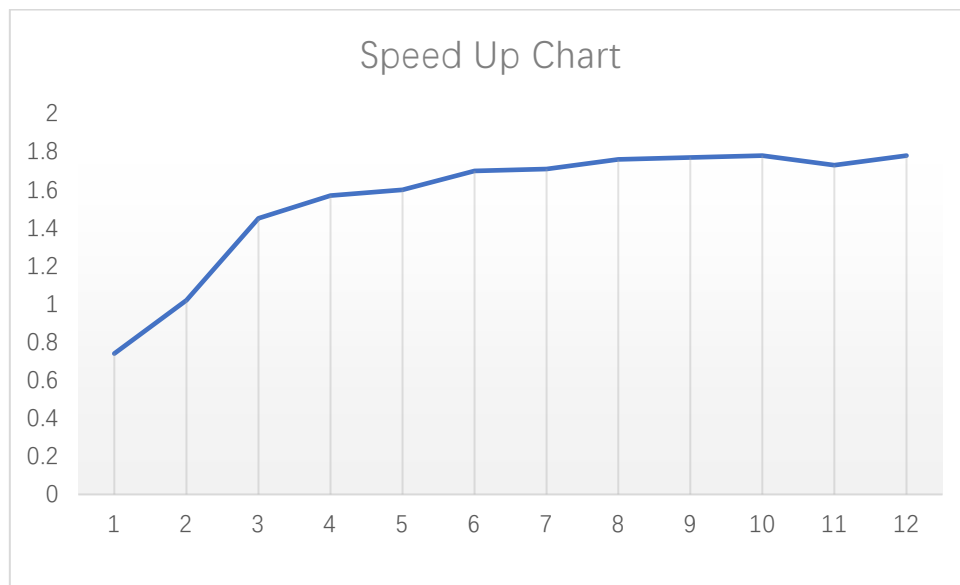
3. “onsetdetection” timer with number of processors

Number of processors	Sequential Time (ms)	Paralleled Time (ms)
1	2102	2147
2	2108	1770



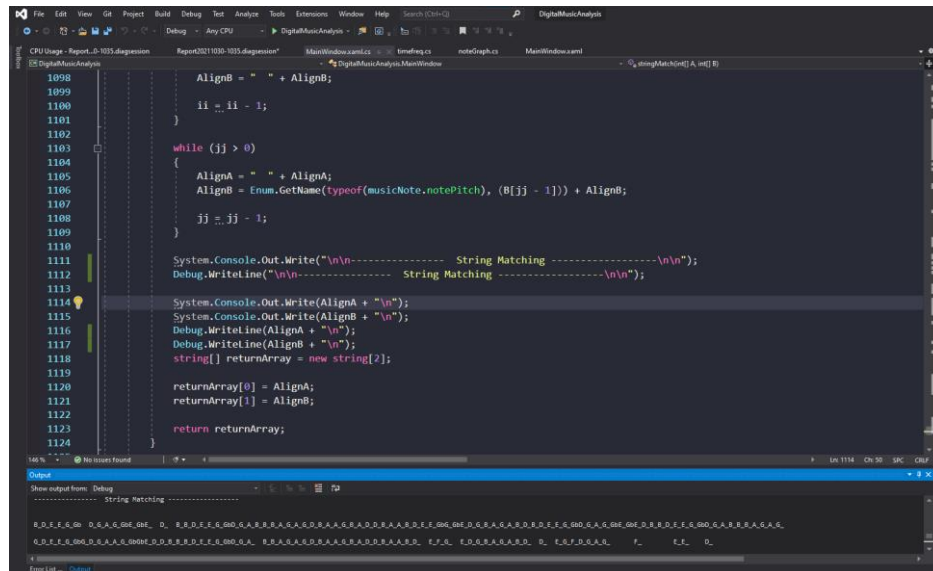
3	1820	1271
4	1826	1207
5	1838	1231
6	1807	1124
7	1818	1057
8	1791	1179
9	1813	1112
10	1776	1118
11	1771	1143
12	1772	1062

Below is the speed up chart by using the formula, execution time of best sequential program, divided by execution time of parallel program.



## Result comparison

Both sequential application's result and the parallel application's result will be compared using the GUI interface. The GUI interface shows the audio frequency, current note playing, and a graph showing which note is playing incorrectly. Comparing the GUI result, it will give us whether the result is correct or not. If it is incorrect, the graph will look different from the sequential application's graph (example in Barrier section). In addition, the application also provides string matching output, comparing output will also give the result on whether the application is correct or not.



```
1090 AlignB = " " + AlignB;
1091 ii = ii - 1;
1092 }
1093 while (jj > 0)
1094 {
1095     AlignA = " " + AlignA;
1096     AlignB = Enum.GetName(typeof(musicNote.notePitch), (B[jj - 1])) + AlignB;
1097     jj = jj - 1;
1098 }
1099
1100 System.Console.Out.WriteLine("\n\n----- String Matching ----- \n\n");
1101 Debug.WriteLine("\n\n----- String Matching ----- \n\n");
1102
1103 System.Console.Out.WriteLine(AlignA + "\n");
1104 System.Console.Out.WriteLine(AlignB + "\n");
1105 Debug.WriteLine(AlignA + "\n");
1106 Debug.WriteLine(AlignB + "\n");
1107 string[] returnArray = new string[2];
1108 returnArray[0] = AlignA;
1109 returnArray[1] = AlignB;
1110 return returnArray;
1111 }
```

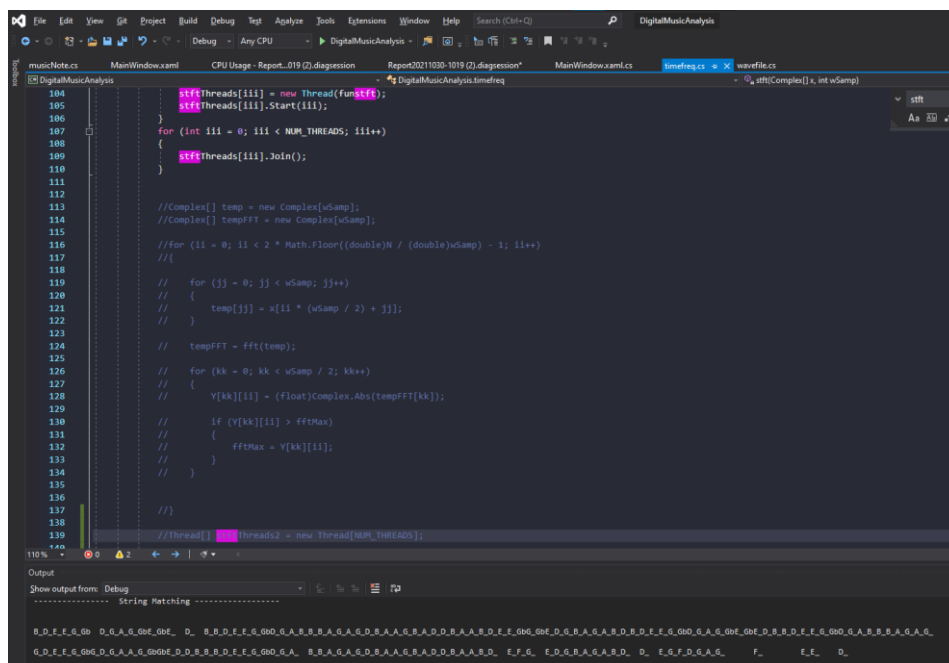
Output

String Matching -----

R\_D\_F\_F\_S\_Gh D\_G\_A\_G\_GhF\_GhF\_ D\_ R\_B\_D\_F\_F\_GhD\_G\_A\_R\_B\_B\_A\_G\_A\_G\_D\_R\_A\_A\_G\_D\_D\_R\_A\_A\_R\_D\_F\_F\_GhD\_GhF\_D\_G\_R\_A\_G\_A\_R\_D\_R\_D\_F\_F\_GhD\_G\_A\_G\_GhF\_GhF\_D\_R\_B\_D\_F\_F\_GhD\_G\_A\_R\_B\_B\_A\_G\_A\_G\_

G\_D\_F\_F\_S\_GhD\_D\_G\_A\_A\_G\_GhDhF\_D\_R\_B\_D\_F\_F\_GhD\_G\_A\_ R\_B\_A\_G\_A\_G\_D\_R\_A\_A\_G\_A\_G\_D\_R\_A\_A\_R\_D\_ F\_F\_G\_ E\_D\_G\_R\_A\_G\_A\_R\_D\_ D\_ F\_G\_F\_D\_G\_A\_G\_ F\_ F\_E\_ D\_

Figure 11(sequential output)



```
104 stffThreads[iii] = new Thread(funstff);
105 stffThreads[iii].Start(iii);
106
107 for (int iiii = 0; iiii < NUM_THREADS; iiii++)
108 {
109     stffThreads[iiii].Join();
110 }
111
112 //Complex[] temp = new Complex[wSamp];
113 //Complex[] tempFFT = new Complex[wSamp];
114
115 //for (ii = 0; ii < 2 * Math.Floor((double)N / (double)wSamp) - 1; ii++)
116 //{
117 //    for (jj = 0; jj < wSamp; jj++)
118 //    {
119 //        temp[jj] = x[ii * (wSamp / 2) + jj];
120 //    }
121 //    tempFFT = fft(temp);
122 //    for (kk = 0; kk < wSamp / 2; kk++)
123 //    {
124 //        Y[kk][iii] = (float)Complex.Abs(tempFFT[kk]);
125 //        if (Y[kk][iii] > fftMax)
126 //        {
127 //            fftMax = Y[kk][iii];
128 //        }
129 //    }
130 //}
131 //Thread[] stffThreads2 = new Thread[NUM_THREADS];
```

Output

String Matching -----

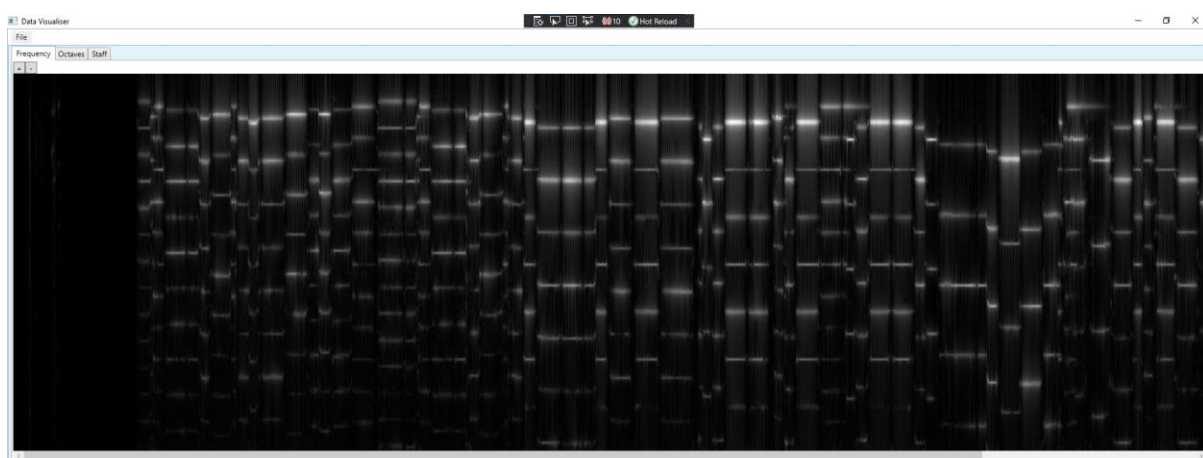
R\_D\_F\_F\_S\_Gh D\_G\_A\_G\_GhF\_GhF\_ D\_ R\_B\_D\_F\_F\_GhD\_G\_A\_R\_B\_B\_A\_G\_A\_G\_D\_R\_A\_A\_G\_D\_D\_R\_A\_A\_R\_D\_F\_F\_GhD\_GhF\_D\_G\_R\_A\_G\_A\_R\_D\_R\_D\_F\_F\_GhD\_G\_A\_G\_GhF\_GhF\_D\_R\_B\_D\_F\_F\_GhD\_G\_A\_R\_B\_B\_A\_G\_A\_G\_

G\_D\_F\_F\_S\_GhD\_D\_G\_A\_A\_G\_GhDhF\_D\_R\_B\_D\_F\_F\_GhD\_G\_A\_ R\_B\_A\_G\_A\_G\_D\_R\_A\_A\_G\_A\_G\_D\_R\_A\_A\_R\_D\_ F\_F\_G\_ E\_D\_G\_R\_A\_G\_A\_R\_D\_ D\_ F\_G\_F\_D\_G\_A\_G\_ F\_ F\_E\_ D\_

Figure 12(parallel output)



*Figure 13(sequential frequency map)*



*Figure 14(parallel frequency map)*



Figure 15(sequential note left, parallel note right)



Figure 16(sequential staff tab)

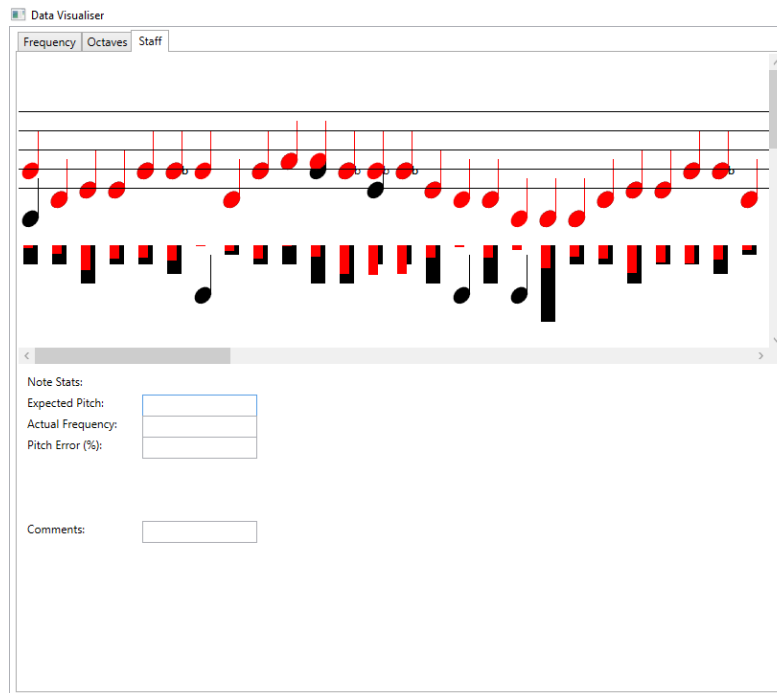


Figure 17(parallel staff tab)

According to the screenshots, both sequential application's output and parallel application's output look identical, making the parallel application's result correct.

## Compiler, software, tools and techniques.

### Compiler

The compiler that is used for the application is the default compiler from C#. The user should change the number of processors he would like to use and then build the application before running in Debug mode; this is because my application does not produce any output when running in Release mode (discuss more in the barrier section).

### Parallelisation framework and Hardware

The application is running on my laptop. Below is the list of my laptop's components

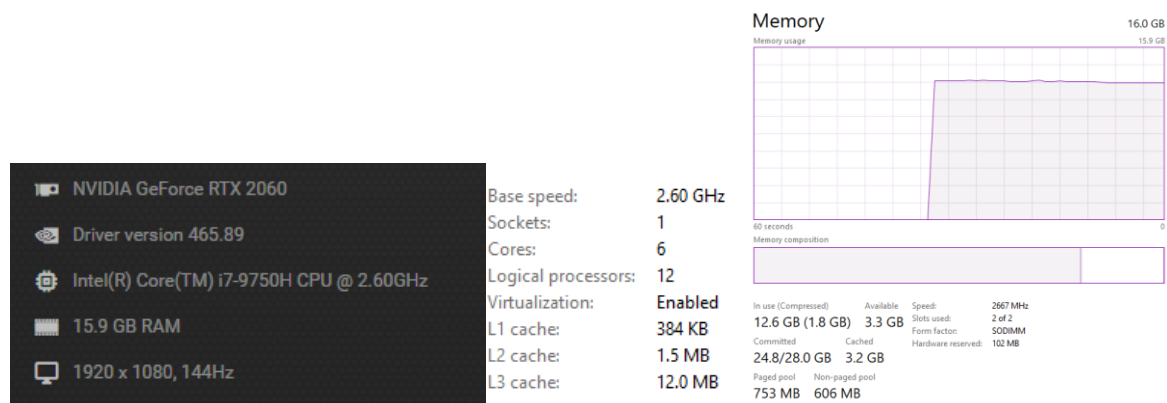


Figure 18(GPU, CPU, Ram, Core, and processor)

The application is built and written on C#, which makes C# is the frame work for parallelisation. The tool that was used for identify application preforming is a feature from Visual studio, called

“Performance Profiler”. The performance profiler gives the user a visual graph on which part of the program is heavily used. The feature also has options to capture GPU usage and many others.

## Techniques

All the techniques and algorithms behind parallelism code are from Wayne’s lecture and the student presentation from Jamie. Both resources give me some great examples of how I should write my threads.

Jamie’s presentation gave some great examples of writing different types of threads. Wayne’s lecture helped me on how to approach application. Once I find the loops that have the potential to be parallel, I will follow the parallelisation logical steps. First is to determine what could be safely parallelised; this means I need to analyse control and data dependencies and decide whether the program needed a new algorithm or transform the algorithm.

## Barriers

The main barrier for this project is to understanding the application. The application contains many very complex algorithms and several loops. The barrier makes the starting of the project very time consuming and difficult. I managed to use high-level class diagram features from the visual studio to help me understand what kind of variables and method is included in each class. Once I understand what the majority of the variables do, the next step is to understand the algorithm. Unfortunately, the algorithm behind fft I did not fully understand; however, I found the data dependencies inside timefreq class and managed to parallel the loop that calls the fft function. This dramatically reduces the time that it takes to run the application.

The next barrier that I encountered is that the function console.writeline does not work on my application. Every time when I run the application, it shows number of error messages from the consoles, but not any console.writeline commands, this makes impossible for me to see any results. After I did some google research, it seems that the cost of this problem could be many, so I decided to avoid this problem and find an away around it, which is using Debug.writeline. This replacement makes the application only be able to print out lines when running in Debug mode, which makes the application slower then running in release mode.

```
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Buffers.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Diagnostics.Tracing.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Diagnostics.Tracing.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Text.RegularExpressions.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.IO.Pipes.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Console.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Threading.Timer.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Runtime.Serialization.Json.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Runtime.Serialization.Xml.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Runtime.Serialization.Primitives.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Reflection.Emit.Lightweight.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Reflection.Primitives.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Runtime.Serialization.Formatters.dll'.
system.windows.data Error: 4 Cannot find source for binding with reference 'ElementName=Parent'. BindingExpressionPath=ActualWidth, DataItem=null; target element is 'ListBoxItem (Name='')'; target property is 'Width' (type 'double').
system.windows.data Error: 4 Cannot find source for binding with reference 'ElementName=Parent'. BindingExpressionPath=Parent.DefinitionMedia.Width, DataItem=null; target element is 'TabControl (Name='TabControl')'; target property is 'Width' (type 'double').
system.windows.data Error: 5 Value produced by BindingExpression is not valid for target property, null BindingExpressionPath=Parent.RenderSize.Height, DataItem=null; target element is 'TabControl (Name='TabControl')'; target property is 'Height' (type 'double').
system.windows.data Error: 4 Cannot find source for binding with reference 'ElementName=Parent'. BindingExpressionPath=Width, DataItem=null; target element is 'WrapPanel (Name='WrapPanel')'; target property is 'Width' (type 'double').
system.windows.data Error: 5 Value produced by BindingExpression is not valid for target property, null BindingExpressionPath=, DataItem=null; target element is 'ScrollViewer (Name='scrolly')'; target property is 'Height' (type 'double').
system.windows.data Error: 5 Value produced by BindingExpression is not valid for target property, null BindingExpressionPath=, DataItem=null; target element is 'ScrollBar (Name='ScrollBar')'; target property is 'ScrollAmount' (type 'double').
system.windows.data Error: 4 Cannot find source for binding with reference 'ElementName=Parent'. BindingExpressionPath=ActualWidth, DataItem=null; target element is 'Grid (Name='')'; target property is 'Width' (type 'double').
system.windows.data Error: 4 Cannot find source for binding with reference 'ElementName=Parent'. BindingExpressionPath=ActualWidth, DataItem=null; target element is 'ScrollViewer (Name='staffscroll')'; target property is 'Width' (type 'double').
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Runtime.Numerics.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.20\System.Threading.Tasks.Parallel.dll'.
The thread 0x0000 has exited with code 0 (0x0).
The thread 0x0000 has exited with code 0 (0x0).
The thread 0x0000 has exited with code 0 (0x0).
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\System.Windows.Forms.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\System.Drawing.Common.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\System.ComponentModel.EventBasedAsync.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\Accessibility.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\Microsoft.Win32.SystemEvents.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\PresentationFramework.SystemXml.dll'.
DigitalMusicAnalysis.exe (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.WindowsDesktop.App\3.1.20\Microsoft.Windows.Common-Theme.dll'.
```

Figure 19(output error)

Furthermore, some paralleled code that was written resulted in slowing down the application. This happened to me while I worked on timefreq class. By revisiting the lectures, I found out why it is slowing down; it is because each "work unit" is not sufficiently large enough, which resulted in did not offset the overhead. The salutation of this problem is to find the loop that does the most amount of work by using a performance profiler and parallel the loops that are doing the most amount of calculation.

The last barrier I had while paralleling is that the output is entirely wrong compared to the sequential version. This is due to the part I tried to parallel having data dependencies. Two methods can potentially overcome this barrier: one is transforming the algorithm, and the second is creating a new algorithm that can reprice the current one. I manage to overcome this problem by transforming the algorithm by bringing the complex array into the thread

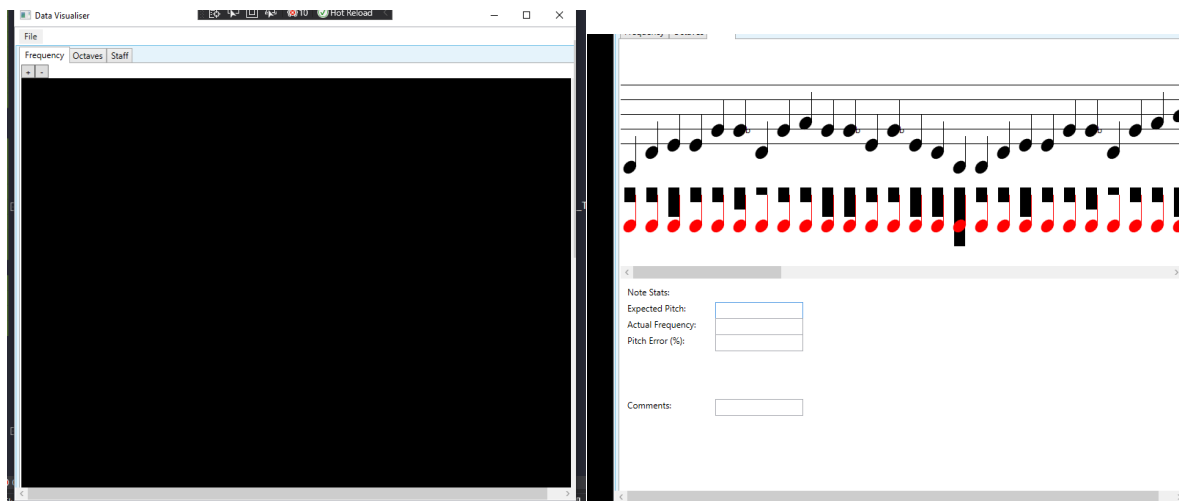


Figure 20(wrong output)

## Explanation Code

timefreq.cs

Add variables and changing some variables to global use

```

6  public class timefreq
7  {
8      public float[][] timeFreqData;
9      public int wSamp;
10     public Complex[] twiddles;
11 }

```

Figure 21(sequential timefreq)

```

8  public class timefreq
9  {
10     const int NUM_THREADS = 4;
11     public float[][] timeFreqData;
12     public int wSamp;
13     public Complex[] twiddles;
14     private int blockSize;
15     private int N;
16     public float[][] Y;
17     public Complex[] newX;
18     public float fftMax;
19 }

```

Figure 22(parallel timefreq)

## First loop parallel

```

26
27 int nearest = (int)Math.Ceiling((double)x.Length / (double)wSamp);
28 nearest = nearest * wSamp;
29
30 Complex[] compX = new Complex[nearest];
31 for (int kk = 0; kk < nearest; kk++)
32 {
33     if (kk < x.Length)
34     {
35         compX[kk] = x[kk];
36     }
37     else
38     {
39         compX[kk] = Complex.Zero;
40     }
41 }
42
43

```

Figure 23(sequential timefreq)

```

40
47 Complex[] compX = new Complex[nearest];
48 //for (int kk = 0; kk < nearest; kk++)
49 //{
50 //    if (kk < x.Length)
51 //    {
52 //        compX[kk] = x[kk];
53 //    }
54 //    else
55 //    {
56 //        compX[kk] = Complex.Zero;
57 //    }
58 //}
59
60 Parallel.For(0, nearest, new ParallelOptions { MaxDegreeOfParallelism = NUM_THREADS }, kk =>
61 {
62     if (kk < x.Length)
63     {
64         compX[kk] = x[kk];
65     }
66     else
67     {
68         compX[kk] = Complex.Zero;
69     }
70 });
71

```

Figure 24(parallel timefreq)

## Second loop parallel

```

43
44 int cols = 2 * nearest / wSamp;
45
46 for (int jj = 0; jj < wSamp / 2; jj++)
47 {
48     timeFreqData[jj] = new float[cols];
49 }
50
51 timeFreqData = stft(compX, wSamp);
52
53 }
54

```

Figure 25(sequential timefreq)

```

79
80 Parallel.For(0, wSamp / 2, new ParallelOptions { MaxDegreeOfParallelism = NUM_THREADS }, jj =>
81 {
82     timeFreqData[jj] = new float[cols];
83 });
84
85 timeFreqData = stft(compX, wSamp);
86
87 }
88

```

Figure 26(parallel timefreq)

## First loop in side stft

```

66 for (ll = 0; ll < wSamp / 2; ll++)
67 {
68     Y[ll] = new float[2 * (int)Math.Floor((double)N / (double)wSamp)];
69 }
70

```

Figure 27(sequential stft)



```

107 Parallel.For(0, wSamp / 2, new ParallelOptions { MaxDegreeOfParallelism = NUM_THREADS }, ll =>
108 {
109     Y[ll] = new float[2 * (int)Math.Floor((double)N / (double)wSamp)];
110 });
111
112 Thread[] stftThreads = new Thread[NUM_THREADS];

```

Figure 28(parallel stft)

Second loop in stft

```

71 Complex[] temp = new Complex[wSamp];
72 Complex[] tempFFT = new Complex[wSamp];
73
74 for (ii = 0; ii < 2 * Math.Floor((double)N / (double)wSamp) - 1; ii++)
75 {
76
77     for (jj = 0; jj < wSamp; jj++)
78     {
79         temp[jj] = x[ii * (wSamp / 2) + jj];
80     }
81
82     tempFFT = fft(temp);
83
84     for (kk = 0; kk < wSamp / 2; kk++)
85     {
86         Y[kk][ii] = (float)Complex.Abs(tempFFT[kk]);
87
88         if (Y[kk][ii] > fftMax)
89         {
90             fftMax = Y[kk][ii];
91         }
92     }
93
94 }
95
96

```

Figure 29(sequential stft)

```

111
112 Thread[] stftThreads = new Thread[NUM_THREADS];
113
114 for (int iii = 0; iii < NUM_THREADS; iii++)
115 {
116     stftThreads[iii] = new Thread(funstft);
117     stftThreads[iii].Start(iii);
118 }
119
120 for (int iii = 0; iii < NUM_THREADS; iii++)
121 {
122     stftThreads[iii].Join();
123 }

```

Figure 30(paralleled stft 1)

```

220 public void funstft(object Id)
221 {
222     int threadId = (int)Id;
223     blockSize = (2 * (int)Math.Floor(N / (double)wSamp) - 1) / NUM_THREADS;
224     int blockSize2 = 2 * (int)Math.Floor(N / (double)wSamp) - 1;
225     int start = threadId * blockSize;
226     int finish = Math.Min(start + blockSize, blockSize2);
227
228     Complex[] temp = new Complex[wSamp];
229     Complex[] tempFFT = new Complex[wSamp];
230
231     for (int ii = start; ii < finish; ii++)
232     {
233
234         for (int jj = 0; jj < wSamp; jj++)
235         {
236             temp[jj] = newX[ii * (wSamp / 2) + jj];
237         }
238
239         tempFFT = fft(temp);
240
241         for (int kk = 0; kk < wSamp / 2; kk++)
242         {
243             Y[kk][ii] = (float)Complex.Abs(tempFFT[kk]);
244
245             if (Y[kk][ii] > fftMax)
246             {
247                 fftMax = Y[kk][ii];
248             }
249         }
250     }
251 }
252

```

Figure 31(paralleled stft 2)

```

175 public Complex[] fft(Complex[] x, int L, Complex[] td)
176 {
177     int ii = 0;
178     int kk = 0;
179     int N = x.Length;
180
181     Complex[] Y = new Complex[N];
182
183     // NEED TO MEMSET TO ZERO?
184
185     if (N == 1)
186     {
187         Y[0] = x[0];
188     }
189     else{
190
191         Complex[] E = new Complex[N/2];
192         Complex[] O = new Complex[N/2];
193         Complex[] even = new Complex[N/2];
194         Complex[] odd = new Complex[N/2];
195
196         for (ii = 0; ii < N; ii++)
197         {
198
199             if (ii % 2 == 0)
200             {
201                 even[ii / 2] = x[ii];
202             }
203             if (ii % 2 == 1)

```

Figure 32(changed FFT's parameters)

mainwindow.cs

```

319 for (int jj = 0; jj < stftRep.timeFreqData[0].Length; jj++)
320 {
321     for (int ii = 0; ii < stftRep.wSamp / 2; ii++)
322     {
323         HFC[jj] = HFC[jj] + (float)Math.Pow((double)stftRep.timeFreqData[ii][jj] * ii, 2);
324     }
325 }
326
327

```

Figure 33(sequential mainwindow)

```

325 Parallel.For(0, stftRep.timeFreqData[0].Length, new ParallelOptions { MaxDegreeOfParallelism = NUM_THREADS }, jj =>
326 {
327     for (int ii = 0; ii < stftRep.wSamp / 2; ii++)
328     {
329         HFC[jj] = HFC[jj] + (float)Math.Pow((double)stftRep.timeFreqData[ii][jj] * ii, 2);
330     }
331 });
332

```

Figure 34(paralleled mainwindow)

```

371 for (int mm = 0; mm < lengths.Count; mm++)
372 {
373     int nearest = (int)Math.Pow(2, Math.Ceiling(Math.Log(lengths[mm], 2)));
374     twiddles = new Complex[nearest];
375     for (ll = 0; ll < nearest; ll++)
376     {
377         double a = 2 * pi * ll / (double)nearest;
378         twiddles[ll] = Complex.Pow(Complex.Exp(-i), (float)a);
379     }
380
381     compX = new Complex[nearest];

```

Figure 35(sequential mainwindow)

```

405 Parallel.For(0, nearest, new ParallelOptions { MaxDegreeOfParallelism = NUM_THREADS }, ll =>
406 {
407     double a = 2 * pi * ll / (double)nearest;
408     twiddles[ll] = Complex.Pow(Complex.Exp(-i), (float)a);
409 });
410 //for (ll = 0; ll < nearest; ll++)
411 //{
412 //    double a = 2 * pi * ll / (double)nearest;
413 //    twiddles[ll] = Complex.Pow(Complex.Exp(-i), (float)a);
414 //}
415

```

Figure 36(paralleled mainwindow)

## Reflection

After completing this project, several goals could be archived, but due to the amount of knowledge and limited time I have, I could not achieve those goals. The first goal for me that I failed to archive is paralleling FFT. The entire application's calculation is mostly done by the method FFT, by transforming the algorithm or creating a new algorithm will significantly reduce the time it takes. I believe to parallel FFT fully, it will require the algorithm to be transformed. In addition, I also believe that there are still ways to parallel the onsetdetection function. The parallel version of my onsetdetection is very basic, and it can still be worked on, including finding more places to parallel, transform or create new algorithms.

Furthermore, After doing some research, the parallelism of this application can be done by using workers from the BackgroundWorker class. However, I did not spend time researching the worker method until the last week. Lastly, the final method that I didn't try out is using my GPU. I'm currently using a NVIDIA GPU, and NVIDIA GPU provides users with a CUDA library, which is a library that helps users to parallel applications.

Overall, after completing this project, my perceptive on programming has changed a lot. I always thought that programming is simply writing code that can run and achieve your goals without worrying how long it will take to run. This project has taught me that writing code is easy, but making it run as efficiently as possible, is very complicated. Most of the time, there is no end to it. There is always a way around it to make it faster, including new algorithms, transform algorithms, threadings or even new hardware. In the future, I can see myself trying out new ways to parallel applications and changing the approach on how to start writing a program with parallelisation in my head.