CPSC 501 Assignment 4 Report
Bernie Mayer
10124540

With this assignment I could not fully get the fast fourier transform to completely work. However I was able to get the main algorithm for the fast fourier to work.

First profile results:

This profile read from a file of 40s and impulse of 4s. Then the FastFourierTransfrom (FFT) convolve was applied and then the data was written to a file. This process is repeated 10 times in order to get better results

Details 〉 Profile 〉 Root   Process

Weight		Self Weight	Symbol Name
5.20 min	100.0%	0 s	▼convolve (1223)
5.20 min	100.0%	0 s	▼Main Thread 0x15514
5.20 min	99.9%	0 s	▼start libdyld.dylib
5.20 min	99.9%	0 s	▼main convolve
5.20 min	99.9%	0 s	▼testProfile convolve
5.15 min	99.1%	179.00 ms	▼convolveFFT convolve
5.15 min	99.0%	5.15 min	▼four1 convolve
251.00 ms	0.0%	251.00 ms	sin libsystem_m.dylib
15.00 ms	0.0%	0 s	▶0xffffffffffffffe
1.00 ms	0.0%	0 s	▶printf libsystem_c.dylib
1.50 s	0.4%	0 s	▶writeToWaveFile convolve
1.04 s	0.3%	38.00 ms	▶readFileDataIntoArray convolve
126.00 ms	0.0%	126.00 ms	convertShortArrayToDouble convolve
79.00 ms	0.0%	79.00 ms	normalizeArray convolve
11.00 ms	0.0%	11.00 ms	DYLD-STUB$$fread convolve
2.00 ms	0.0%	0 s	▶0xffffffffffffffe
1.00 ms	0.0%	0 s	▶malloc libsystem_malloc.dylib
1.00 ms	0.0%	0 s	▶fopen libsystem_c.dylib
1.00 ms	0.0%	0 s	▶0x10043900f convolve
1.00 ms	0.0%	0 s	▶exit libsystem_c.dylib
2.00 ms	0.0%	0 s	▼_dyld_start dyld
2.00 ms	0.0%	0 s	▼dyldbootstrap::start(macho_header const*, int, char const**, long, macho_header const*, unsigned long*) dyld
2.00 ms	0.0%	0 s	▶dyld::_main(macho_header const*, unsigned long, int, char const**, char const**, char const**, unsigned long*) dyld

These results show that the method four1 is where the code spent most of its time.

To optimize for this code to reduce the amount that four1 is called, I moved the calculation of padded_h which is the padded version of the IR array or h array to outside the for loop that calculates the fast fourier for the segment. The segment in this case is the data that is being transformed and then multiplied and later on overlap added to get the final audio data.

Second profile:

4.48 min	100.0%	0 s	▼convolve (1474)	
4.48 min	100.0%	0 s	▼Main Thread 0x19a7d	
4.48 min	99.9%	0 s	▼start	libdyld.dylib
4.48 min	99.9%	0 s	▼main	convolve
4.48 min	99.9%	0 s	▼testProfile	convolve
4.44 min	99.1%	152.00 ms	▼convolveFFT	convolve
4.44 min	99.0%	4.44 min	▶four1	convolve
4.00 ms	0.0%	4.00 ms	DYLD-STUB$$sin	convolve
1.00 ms	0.0%	0 s	▶0xfffffffffffffffe	
1.23 s	0.4%	0 s	▶writeToWaveFile	convolve
931.00 ms	0.3%	26.00 ms	▶readFileDataIntoArray	convolve
126.00 ms	0.0%	126.00 ms	convertShortArrayToDouble	convolve
80.00 ms	0.0%	80.00 ms	normalizeArray	convolve
8.00 ms	0.0%	8.00 ms	DYLD-STUB$$fread	convolve
5.00 ms	0.0%	0 s	▶fopen	libsystem_c.dylib
1.00 ms	0.0%	0 s	▶_dyld_start	dyld

The second profile was faster by 0.72 min. This optimization improves performance since the calculation is only performed once per call in convolveFFT instead of each time a new segment is transformed. Since the padded_h transform stays constant this change will not cause any changes to the results. This refactor is a reduce work inside a loop refactor since it reduces the amount of work that is done inside the calculation of the segment loop.

The next optimization I performed was to make the four_1 method faster. This optimization took two data accesses to an array and changed them to access the array once and then store the access.

Old version:

```
                    tempr = wr * data[j] - wi * data[j + 1];
                    tempi = wr * data[j + 1] + wi * data[j];
```

New version:

```
   double data_1 = data[j];
   double data_2 = data[j + 1];
                    tempr = wr * data_1 - wi * data_2;
                    tempi = wr * data_2 + wi * data_1;
```

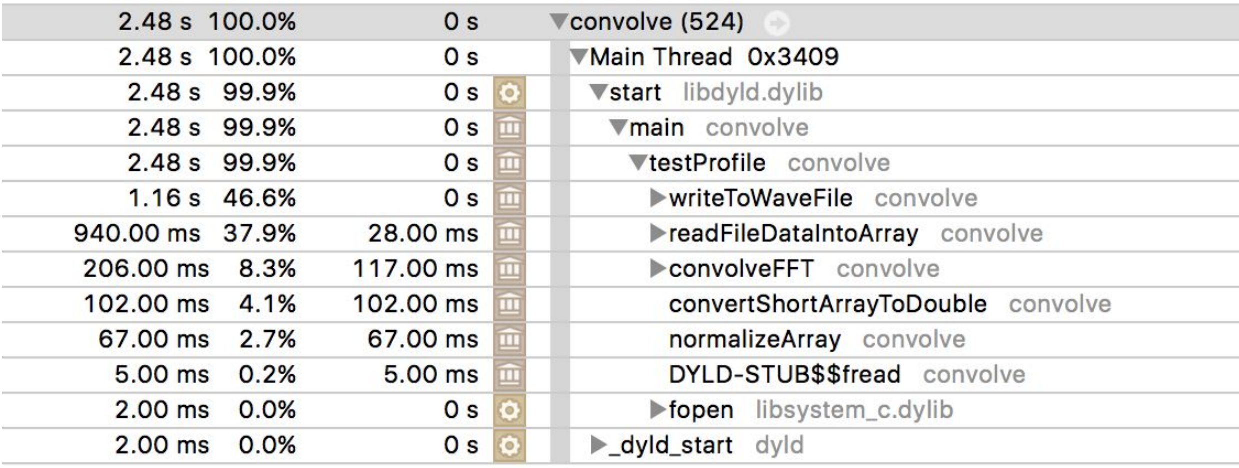
Third profile:

4.49 min	100.0%	0 s	▼convolve (1728)
4.49 min	100.0%	0 s	▼Main Thread 0x22b2e
2.00 ms	0.0%	0 s	▶_dyld_start dyld
4.49 min	99.9%	0 s	▼start libdyld.dylib
4.49 min	99.9%	0 s	▼main convolve
4.49 min	99.9%	0 s	▼testProfile convolve
1.00 ms	0.0%	0 s	▶readHeaderOfAudioFile convolve
1.00 ms	0.0%	1.00 ms	free libsystem_malloc.dylib
2.00 ms	0.0%	0 s	▶fopen libsystem_c.dylib
6.00 ms	0.0%	6.00 ms	DYLD-STUB$$fread convolve
73.00 ms	0.0%	73.00 ms	normalizeArray convolve
121.00 ms	0.0%	121.00 ms	convertShortArrayToDouble convolve
943.00 ms	0.3%	32.00 ms	▶readFileDataIntoArray convolve
1.22 s	0.4%	0 s	▶writeToWaveFile convolve
4.45 min	99.1%	150.00 ms	▼convolveFFT convolve
2.00 ms	0.0%	0 s	▶0xfffffffffffffffe
4.45 min	99.0%	4.45 min	▼four1 convolve
102.00 ms	0.0%	102.00 ms	sin libsystem_m.dylib

This optimization made the profile take 4.49 min. Which is not really an improvement. However I believe that this optimization is a worthwhile optimization, it might even be an optimization that the compilier is already taking to speed up the code. Therefore I believe that it is still a worthwhile change.

In order to speed up this code I now looked at reducing the number of calls to the four1 method. I saw that I was calling it each time in a for loop, this behavoir was also seemingly a bug.

Fourth Profile:



2.48 s	100.0%	0 s	▼convolve (524)
2.48 s	100.0%	0 s	▼Main Thread 0x3409
2.48 s	99.9%	0 s	▼start libdyld.dylib
2.48 s	99.9%	0 s	▼main convolve
2.48 s	99.9%	0 s	▼testProfile convolve
1.16 s	46.6%	0 s	▶writeToWaveFile convolve
940.00 ms	37.9%	28.00 ms	▶readFileDataIntoArray convolve
206.00 ms	8.3%	117.00 ms	▶convolveFFT convolve
102.00 ms	4.1%	102.00 ms	convertShortArrayToDouble convolve
67.00 ms	2.7%	67.00 ms	normalizeArray convolve
5.00 ms	0.2%	5.00 ms	DYLD-STUB$$fread convolve
2.00 ms	0.0%	0 s	▶fopen libsystem_c.dylib
2.00 ms	0.0%	0 s	▶_dyld_start dyld

This optimization speeds up the program by a lot and also makes the Fast Fourier Transform code get closer to working properly. The code used to run in 4.49 minutes and it now runs in seconds. This is a massive improvement over the old version and allows for much greater speed.

The code used to be calling the method four1 each time a multiplaction between the real and imaginary components of the signals was multiplied. The loop used to have the following structure.

```
for (int k = 0; k < SEGMENT_SIZE; k+=2)
   {

         //calculate the imaginary and real components for the multiplication between the arrays

         four1(results...)
}
```

The call to four1(results..) was moved to outside loop, and massively sped up the code since doing that work inside that loop was unneeded and was likely causing bugs.

This optimization also changes the bottlenecks of this code. The code is spending much less time convolving the data. The bottlenecks of the code are now in the reading and the writing to the file. In order to show better results for this I decided to make the code be repeated more. This is the result of that profiling. The profiling is now run on the same data 1000 times instead of 10 times. This will allow for more accurate results

Weight		Self Weight	Symbol Name
4.48 min	100.0%	0 s	▼convolve (649)
4.48 min	100.0%	0 s	▼Main Thread 0x78bc
4.48 min	99.9%	0 s	▼start libdyld.dylib
4.48 min	99.9%	0 s	▼main convolve
4.48 min	99.9%	1.00 ms	▼testProfile convolve
2.08 min	46.2%	0 s	▶writeToWaveFileDouble convolve
1.62 min	36.0%	2.71 s	▶readFileDataIntoArray convolve
25.67 s	9.5%	16.01 s	▶convolveFFT convolve
13.47 s	5.0%	13.47 s	convertShortArrayToDouble convolve
7.39 s	2.7%	7.39 s	normalizeArray convolve
475.00 ms	0.1%	475.00 ms	DYLD-STUB$$fread convolve
290.00 ms	0.1%	0 s	▶fopen libsystem_c.dylib
48.00 ms	0.0%	0 s	▶0xfffffffffffffffe
45.00 ms	0.0%	1.00 ms	▶malloc libsystem_malloc.dylib
15.00 ms	0.0%	0 s	▶printf libsystem_c.dylib
13.00 ms	0.0%	0 s	▶fseek libsystem_c.dylib
7.00 ms	0.0%	0 s	▶readHeaderOfAudioFile convolve
6.00 ms	0.0%	1.00 ms	▶free_large libsystem_malloc.dylib
4.00 ms	0.0%	1.00 ms	▶free libsystem_malloc.dylib
1.00 ms	0.0%	1.00 ms	madvise libsystem_kernel.dylib
1.00 ms	0.0%	1.00 ms	0x7fff90b6ed20 libsystem_m.dylib
1.00 ms	0.0%	0 s	▶exit libsystem_c.dylib
3.00 ms	0.0%	0 s	▶_dyld_start dyld

The next bottleneck I was able to identify was how I was writing to the file. I was not writing to the file in the most efficient way. In order to make the code run faster I would use one fwrite() call instead of iterating over the array in order to write to the file.

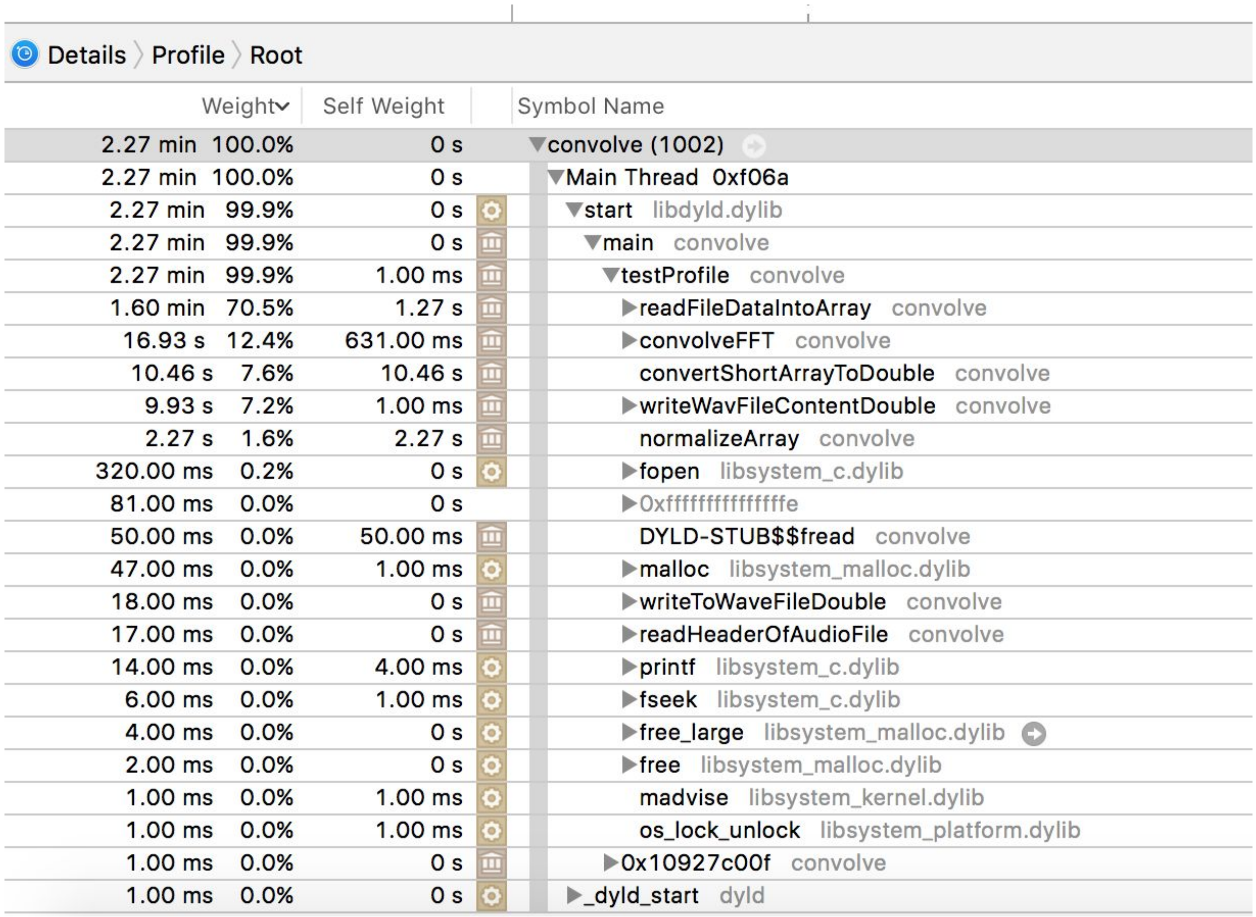
Fifth Profile:

Details 〉 Profile 〉 Root

Weight		Self Weight	Symbol Name	
2.69 min	100.0%	0 s	▼convolve (845)	
2.69 min	100.0%	0 s	▼Main Thread 0xc027	
2.69 min	99.9%	0 s	▼start	libdyld.dylib
2.69 min	99.9%	0 s	▼main	convolve
2.69 min	99.9%	2.00 ms	▼testProfile	convolve
1.64 min	61.1%	2.68 s	▶readFileDataIntoArray	convolve
26.32 s	16.3%	16.57 s	▶convolveFFT	convolve
14.23 s	8.8%	14.23 s	convertShortArrayToDouble	convolve
13.41 s	8.3%	0 s	▶writeToWaveFileDouble	convolve
7.63 s	4.7%	7.63 s	normalizeArray	convolve
543.00 ms	0.3%	543.00 ms	DYLD-STUB$$fread	convolve
345.00 ms	0.2%	0 s	▶fopen	libsystem_c.dylib
48.00 ms	0.0%	1.00 ms	▶malloc	libsystem_malloc.dylib
15.00 ms	0.0%	0 s	▶0xfffffffffffffffe	
10.00 ms	0.0%	0 s	▶printf	libsystem_c.dylib
7.00 ms	0.0%	0 s	▶readHeaderOfAudioFile	convolve
3.00 ms	0.0%	0 s	▶free_large	libsystem_malloc.dylib
3.00 ms	0.0%	1.00 ms	▶free	libsystem_malloc.dylib
2.00 ms	0.0%	0 s	▶fseek	libsystem_c.dylib
1.00 ms	0.0%	0 s	▶0x10dbd600f	convolve
2.00 ms	0.0%	0 s	▶_dyld_start	dyld

The code now runs in 2.69 minutes instead of 4.48 minutes of the old version. This optimization now writes the file as byte array, which is must faster than writing for each array item. This optimization is an example of an IO Technique.

I decided to do a compiler optimization for the next optimization. This is the result of the that profile.



Details 〉 Profile 〉 Root

Weight		Self Weight	Symbol Name
2.27 min	100.0%	0 s	▼convolve (1002)
2.27 min	100.0%	0 s	▼Main Thread 0xf06a
2.27 min	99.9%	0 s	▼start libdyld.dylib
2.27 min	99.9%	0 s	▼main convolve
2.27 min	99.9%	1.00 ms	▼testProfile convolve
1.60 min	70.5%	1.27 s	▶readFileDataIntoArray convolve
16.93 s	12.4%	631.00 ms	▶convolveFFT convolve
10.46 s	7.6%	10.46 s	convertShortArrayToDouble convolve
9.93 s	7.2%	1.00 ms	▶writeWavFileContentDouble convolve
2.27 s	1.6%	2.27 s	normalizeArray convolve
320.00 ms	0.2%	0 s	▶fopen libsystem_c.dylib
81.00 ms	0.0%	0 s	▶0xfffffffffffffffe
50.00 ms	0.0%	50.00 ms	DYLD-STUB$$fread convolve
47.00 ms	0.0%	1.00 ms	▶malloc libsystem_malloc.dylib
18.00 ms	0.0%	0 s	▶writeToWaveFileDouble convolve
17.00 ms	0.0%	0 s	▶readHeaderOfAudioFile convolve
14.00 ms	0.0%	4.00 ms	▶printf libsystem_c.dylib
6.00 ms	0.0%	1.00 ms	▶fseek libsystem_c.dylib
4.00 ms	0.0%	0 s	▶free_large libsystem_malloc.dylib
2.00 ms	0.0%	0 s	▶free libsystem_malloc.dylib
1.00 ms	0.0%	1.00 ms	madvise libsystem_kernel.dylib
1.00 ms	0.0%	1.00 ms	os_lock_unlock libsystem_platform.dylib
1.00 ms	0.0%	0 s	▶0x10927c00f convolve
1.00 ms	0.0%	0 s	▶_dyld_start dyld

The code become slightly faster when I used gcc 01, I could not figure out how to get 02 or even 03 to work with the profiler.

As for regression tests, I would have done those at each step by comparing the files. However I had problems with some of the Fast Fourier Code. One technique I would have used was diff, or even manually listening to the music files.

My last optimization would be de fuse this chunk of code:

```
for (int r = 0; r < SEGMENT_SIZE * 2; r+=2)
{
```

```
    if ((baseIndex + r) < P && (baseIndex + 1 + r) < P) {

      y[baseIndex + r] += (float) results[r];
      y[baseIndex + r + 1] += (float) results[r + 1];
    }
}
```

I would have made two loops so that the if statement was not exucuted as much.

```
  for (int r = 0; r < (baseIndex + r) < P; r+=2)
  {
      y[baseIndex + r] += (float) results[r];
      y[baseIndex + r + 1] += (float) results[r + 1];
    }
```

This code would likely function in the same way as the old code. Just would not have had the if statement.

I used github for this project. Here is the link to that github:

https://github.com/BernieMayer/CPSC501_A4