Jason K Lamphere
CSCI-551
Professor: Sam Siewert

FFT's

My primary interest outside of computers is music and I've invested/wasted a significant amount of time over the course of my life using DAW's (digital audio workstations) and other audio processing tools. This sent me searching for an algorithm fundamental to the processing and quickly settling on the Fast Fourier Transform or FFT for short. Like all transforms the purpose of a Fourier transform is to map a function from its original function space to another function space. In the case of a Fourier Transform the goal is to take a function and decompose it into its constituent sinusoidal frequencies. Since any waveform can be reproduced by adding sinusoidal functions together this can be useful for processing and making sense of complex functions and waveforms. A simple example that I will be using throughout this project is interpreting the waveform(signal or amplitude over time) of a musical signal or chord and expressing it instead in terms of the volumes and frequencies of its constituent notes so that a filter can be applied to attenuate or remove one of those constituent inputs.

A project folder for a single song may have Gigabytes of raw audio information that has to get processed before it gets rendered down into a familiar and portable format like mp3. Making optimal use of the hardware available to consumers so that work can be performed as quickly as possible is an important job of any standard daw. On today's computer systems that means making use of the multiple cores available to try and speed up an algorithm that may get asked to perform on very large data sets. In our case, this means FFT's.

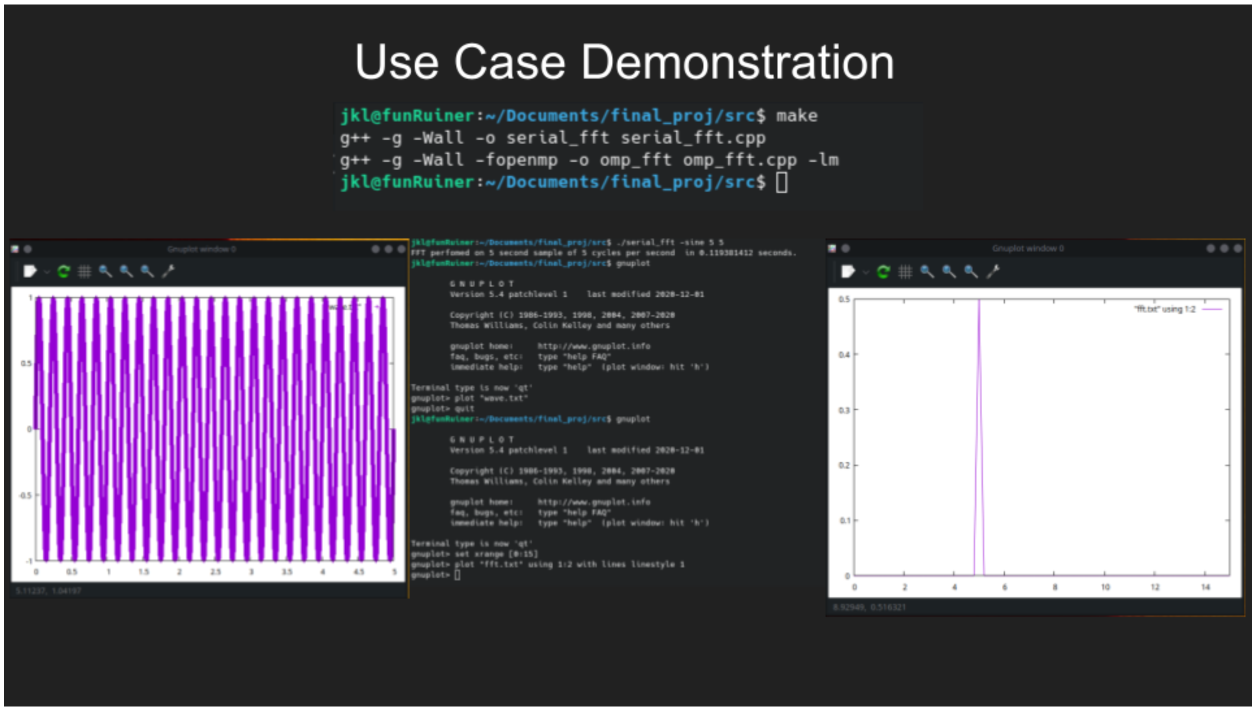
An FFT is an algorithm that computes the Discrete Fourier Transform of a sequence in O(N Log N) instead of O(N^2) as would occur when one simply applies the definition of DFT. Our goal is to further improve this through parallelization to achieve the smallest time complexity possible, which should be O(n/p log p) where p is our number for threads.

The most common FFT algorithm in use today is known as the Cooley-Tukey algorithm of which there are many varieties. Some of the simpler easier to grasp, less space-efficient varieties provide less opportunity for speed up via parallelization. The simplest out of place recursive version is known as the radix-2 DIT case doesn't present much if any opportunity for parallelization at all. However, there are a number of iterative approaches that attempt to limit the space complexity to in-place that make use of multiple nested loops and has several of the steps decomposed into separate loops instead of all occurring in one. These present ample opportunity to see if parallelization attempts can be made. I will be attempting to parallelize an Iterative radix-2 FFT that is implemented using bit reversal permutation based on this pseudocode

```
algorithm iterative-fft is
    input: Array a of n complex values where n is a power of 2.
    output: Array A the DFT of a.

    bit-reverse-copy(a, A)
    n ← a.length
    for s = 1 to log(n) do
        m ← $2^s$
        $\omega_m$ ← exp(−2πi/m)
        for k = 0 to n-1 by m do
            ω ← 1
            for j = 0 to m/2 – 1 do
                t ← ω A[k + j + m/2]
                u ← A[k + j]
                A[k + j] ← u + t
                A[k + j + m/2] ← u – t
                ω ← ω $\omega_m$
    return A

algorithm bit-reverse-copy(a,A) is
    input: Array a of n complex values where n is a power of 2.
    output: Array A of size n.

    n ← a.length
    for k = 0 to n - 1 do
        A[rev(k)] := a[k]
```

While Fourier Transforms are calculated through integration the process behind rapidly executing the discrete Fourier transform involves factorization using matrix multiplication. This pattern of matrix multiplication when drawn out as a diagram creates a pattern that looks like a butterfly which is why it's often referred to as a series of butterfly operations in the literature. During these butterfly equations, our inputs are multiplied by a series of multiplicative trigonometric constants that are referred to in the literature as “Twiddle Factors”. One of the ways we will be attempting to induce speed up is by separating the calculation of these twiddle factors into a table before run so the process can be effectively parallelized. Many simplified versions of the code calculate these factors during each iteration of the main loop and thus become a hurdle to parallelization this way.

Sequential Solution and Computation Time

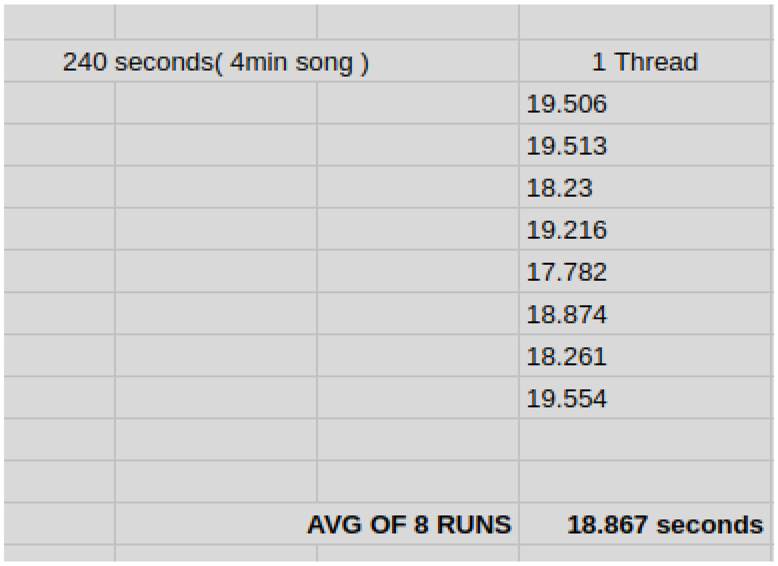


Use Case Demonstration

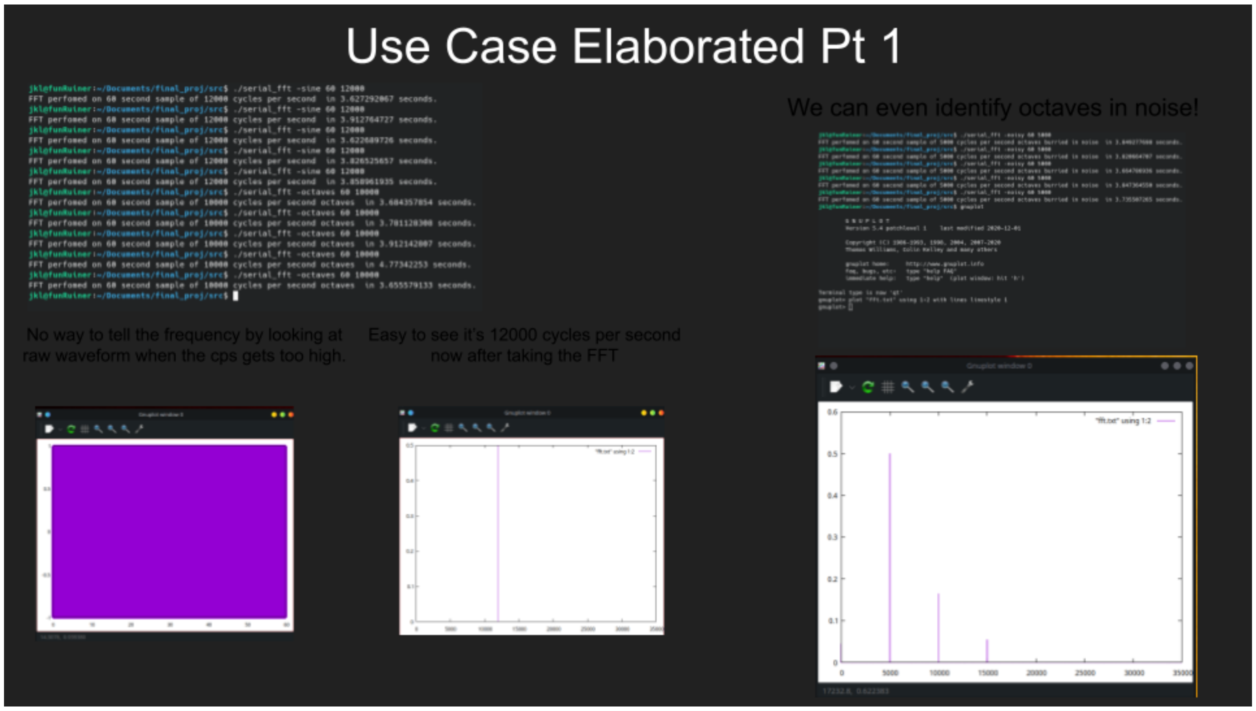
```
jkl@funRuiner:~/Documents/final_proj/src$ make
g++ -g -Wall -o serial_fft serial_fft.cpp
g++ -g -Wall -fopenmp -o omp_fft omp_fft.cpp -lm
jkl@funRuiner:~/Documents/final_proj/src$ 
```

```
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -sine 5 5
FFT perfomed on 5 second sample of 5 cycles per second  in 0.119381412 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ gnuplot

        G N U P L O T
        Version 5.4 patchlevel 1    last modified 2020-12-01

        Copyright (C) 1986-1993, 1998, 2004, 2007-2020
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot "wave.txt"
gnuplot> quit
jkl@funRuiner:~/Documents/final_proj/src$ gnuplot

        G N U P L O T
        Version 5.4 patchlevel 1    last modified 2020-12-01

        Copyright (C) 1986-1993, 1998, 2004, 2007-2020
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> set xrange [0:15]
gnuplot> plot "fft.txt" using 1:2 with lines linestyle 1
gnuplot> 
```

Gnuplot window 0
Gnuplot window 0
"fft.txt" using 1:2



240 seconds(4min song)			1 Thread
			19.506
			19.513
			18.23
			19.216
			17.782
			18.874
			18.261
			19.554
		AVG OF 8 RUNS	**18.867 seconds**



Use Case Elaborated Pt 1

```
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -sine 60 12000
FFT perfomed on 60 second sample of 12000 cycles per second  in 3.627292067 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -sine 60 12000
FFT perfomed on 60 second sample of 12000 cycles per second  in 3.912764727 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -sine 60 12000
FFT perfomed on 60 second sample of 12000 cycles per second  in 3.622689726 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -sine 60 12000
FFT perfomed on 60 second sample of 12000 cycles per second  in 3.826525657 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -sine 60 12000
FFT perfomed on 60 second sample of 12000 cycles per second  in 3.858961935 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -octaves 60 10000
FFT perfomed on 60 second sample of 10000 cycles per second octaves  in 3.684357854 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -octaves 60 10000
FFT perfomed on 60 second sample of 10000 cycles per second octaves  in 3.781128308 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -octaves 60 10000
FFT perfomed on 60 second sample of 10000 cycles per second octaves  in 3.912142087 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -octaves 60 10000
FFT perfomed on 60 second sample of 10000 cycles per second octaves  in 4.77342253 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -octaves 60 10000
FFT perfomed on 60 second sample of 10000 cycles per second octaves  in 3.655579133 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ 
```

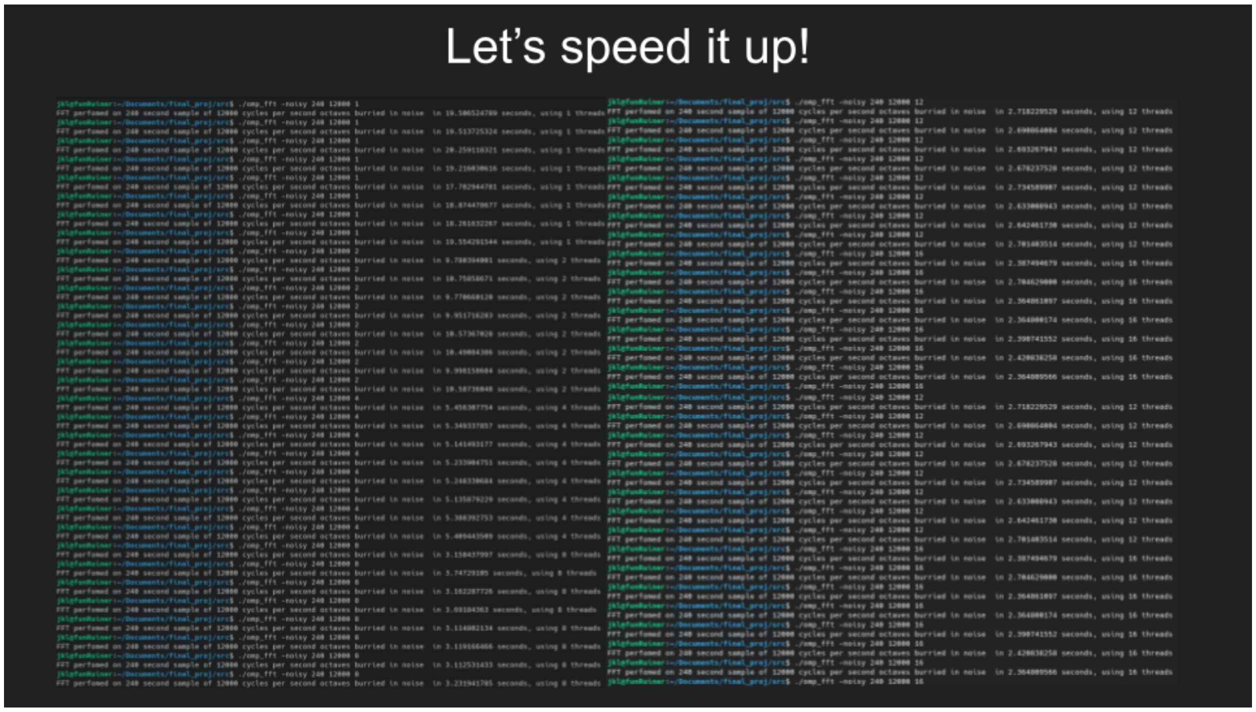
No way to tell the frequency by looking at raw waveform when the cps gets too high.

Easy to see it's 12000 cycles per second now after taking the FFT

We can even identify octaves in noise!

```
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.846277680 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.820064707 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.664700336 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.847364550 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.735507265 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ gnuplot

        G N U P L O T
        Version 5.4 patchlevel 1    last modified 2020-12-01

        Copyright (C) 1986-1993, 1998, 2004, 2007-2020
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot "fft.txt" using 1:2 with lines linestyle 1
gnuplot> 
```

Parallel design and solution with computation time

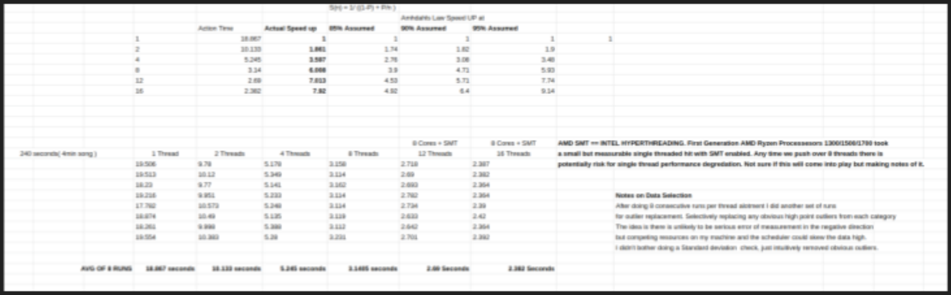


Let's speed it up!

```
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 19.506524789 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 19.513725324 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 20.259118321 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 19.216030616 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 17.782944781 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 18.874478077 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 18.261632287 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 1
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 19.554291544 seconds, using 1 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 9.780394001 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 10.75858671 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 9.770660120 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 9.951716283 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 10.57367028 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 10.49004306 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 9.998150604 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 2
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 10.58730048 seconds, using 2 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.456307754 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.349337857 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.141493177 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.233904751 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.248330684 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.135879229 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.308392753 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 4
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 5.409443509 seconds, using 4 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.158437997 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.74729185 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.182287726 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.69184363 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.114802134 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.119166466 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.112531433 seconds, using 8 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 8
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 3.231941785 seconds, using 8 threads
```

```
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.718229529 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.690064004 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.693267943 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.678237528 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.734589987 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.633000943 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.642461738 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.701403514 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.387494679 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.704629000 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.364061097 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.364000174 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.390741552 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.420838258 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.364009566 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.718229529 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.690064004 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.693267943 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.678237528 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.734589987 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.633000943 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.642461738 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 12
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.701403514 seconds, using 12 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.387494679 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.704629000 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.364061097 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.364000174 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.390741552 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.420838258 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
FFT perfomed on 240 second sample of 12000 cycles per second octaves burried in noise  in 2.364009566 seconds, using 16 threads
jkl@funRainer:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
```

Speed Up: A Closer Look



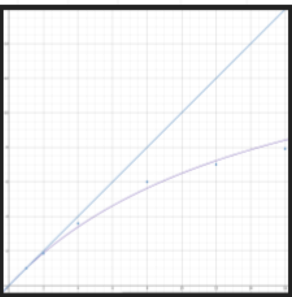
S(n) = 1/ ((1-P) + P/n)

	Action Time	Actual Speed up	85% Assumed	90% Assumed	95% Assumed	
1	18.867	**1**	1	1	1	1
2	10.133	**1.861**	1.74	1.82	1.9	
4	5.245	**3.597**	2.76	3.08	3.48	
8	3.14	**6.008**	3.9	4.71	5.93	
12	2.69	**7.013**	4.53	5.71	7.74	
16	2.382	**7.92**	4.92	6.4	9.14	

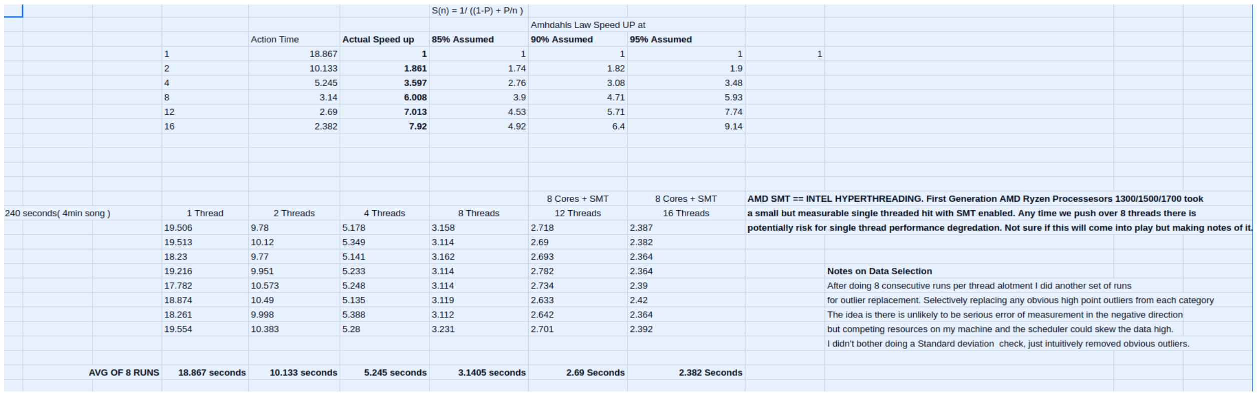
240 seconds(4min song)	1 Thread	2 Threads	4 Threads	8 Threads	8 Cores + SMT 12 Threads	8 Cores + SMT 16 Threads
	19.506	9.78	5.178	3.158	2.718	2.387
	19.513	10.12	5.349	3.114	2.69	2.382
	18.23	9.77	5.141	3.162	2.693	2.364
	19.216	9.951	5.233	3.114	2.782	2.364
	17.782	10.573	5.248	3.114	2.734	2.39
	18.874	10.49	5.135	3.119	2.633	2.42
	18.261	9.998	5.388	3.112	2.642	2.364
	19.554	10.383	5.28	3.231	2.701	2.392
AVG OF 8 RUNS	**18.867 seconds**	**10.133 seconds**	**5.245 seconds**	**3.1405 seconds**	**2.69 Seconds**	**2.382 Seconds**

AMD SMT == INTEL HYPERTHREADING. First Generation AMD Ryzen Processesors 1300/1500/1700 took a small but measurable single threaded hit with SMT enabled. Any time we push over 8 threads there is potentially risk for single thread performance degredation. Not sure if this will come into play but making notes of it.

Notes on Data Selection
After doing 8 consecutive runs per thread alotment I did another set of runs
for outlier replacement. Selectively replacing any obvious high point outliers from each category
The idea is there is unlikely to be serious error of measurement in the negative direction
but competing resources on my machine and the scheduler could skew the data high.
I didn't bother doing a Standard deviation check, just intuitively removed obvious outliers.



I used a table to estimate parallelization against my real world data and kept adjusting P until I reached the best fit I could. There was no perfect P that fit all data points perfectly but 94% seems to be the sweet spot. It's notable performance curve worsens as thread count gets closer to the total cores available. There is still improvement but lets and less as we go further.



S(n) = 1/ ((1-P) + P/n)

Amhdahls Law Speed UP at

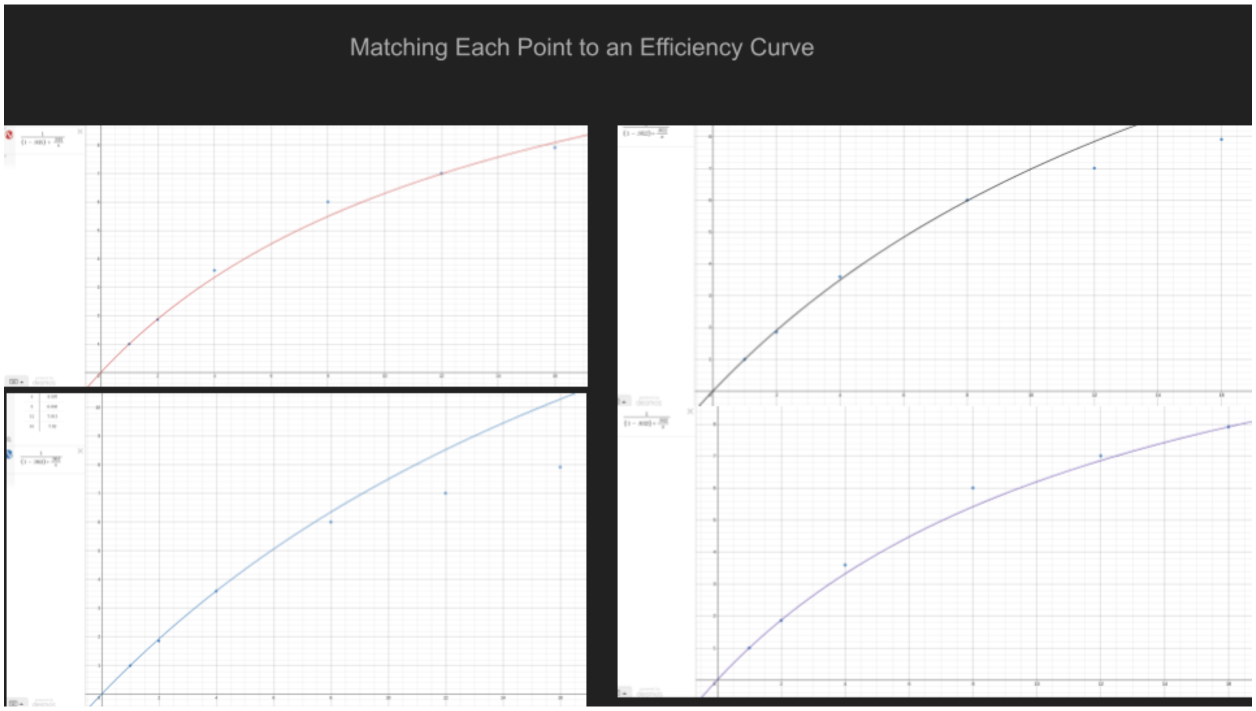
	Action Time	Actual Speed up	85% Assumed	90% Assumed	95% Assumed	
1	18.867	**1**	1	1	1	1
2	10.133	**1.861**	1.74	1.82	1.9	
4	5.245	**3.597**	2.76	3.08	3.48	
8	3.14	**6.008**	3.9	4.71	5.93	
12	2.69	**7.013**	4.53	5.71	7.74	
16	2.382	**7.92**	4.92	6.4	9.14	

240 seconds(4min song)	1 Thread	2 Threads	4 Threads	8 Threads	8 Cores + SMT 12 Threads	8 Cores + SMT 16 Threads
	19.506	9.78	5.178	3.158	2.718	2.387
	19.513	10.12	5.349	3.114	2.69	2.382
	18.23	9.77	5.141	3.162	2.693	2.364
	19.216	9.951	5.233	3.114	2.782	2.364
	17.782	10.573	5.248	3.114	2.734	2.39
	18.874	10.49	5.135	3.119	2.633	2.42
	18.261	9.998	5.388	3.112	2.642	2.364
	19.554	10.383	5.28	3.231	2.701	2.392
AVG OF 8 RUNS	**18.867 seconds**	**10.133 seconds**	**5.245 seconds**	**3.1405 seconds**	**2.69 Seconds**	**2.382 Seconds**

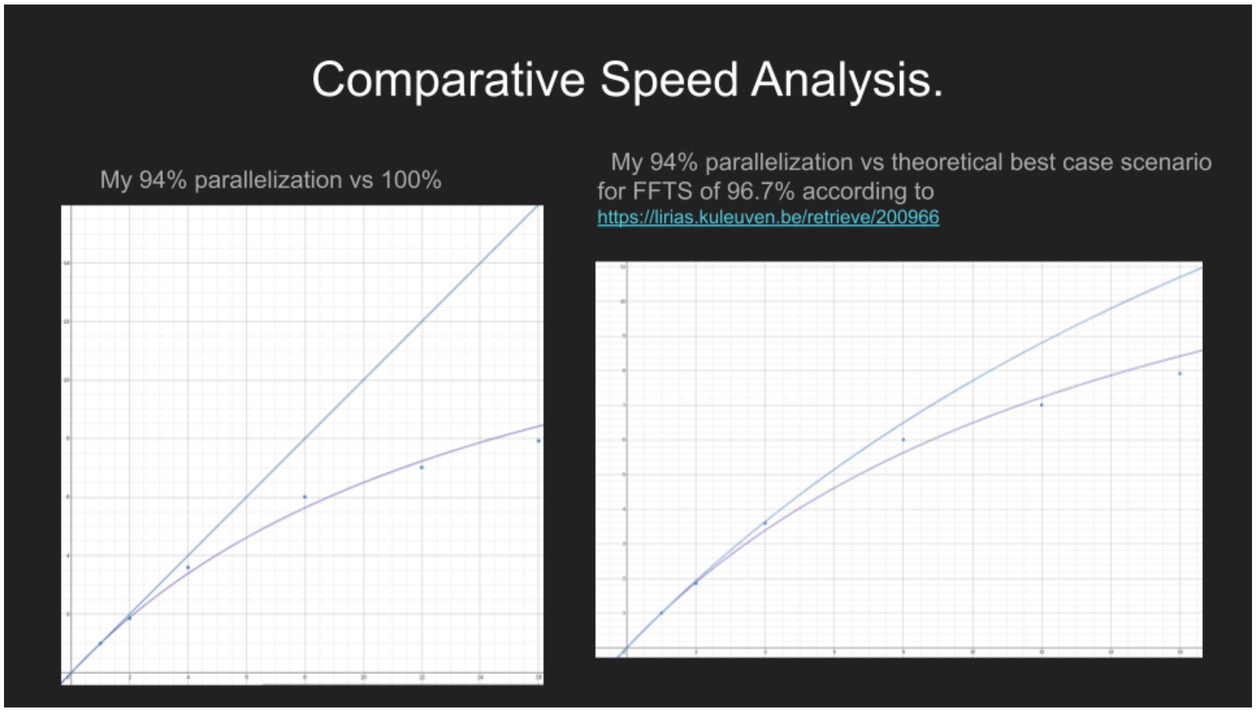
AMD SMT == INTEL HYPERTHREADING. First Generation AMD Ryzen Processesors 1300/1500/1700 took a small but measurable single threaded hit with SMT enabled. Any time we push over 8 threads there is potentially risk for single thread performance degredation. Not sure if this will come into play but making notes of it.

Notes on Data Selection
After doing 8 consecutive runs per thread alotment I did another set of runs
for outlier replacement. Selectively replacing any obvious high point outliers from each category
The idea is there is unlikely to be serious error of measurement in the negative direction
but competing resources on my machine and the scheduler could skew the data high.
I didn't bother doing a Standard deviation check, just intuitively removed obvious outliers.

https://docs.google.com/spreadsheets/d/1_v3FaX-F1XovysnmPOWQ9fInSAZIukWGBXyd2aWtQG0/edit?usp=sharing



Matching Each Point to an Efficiency Curve

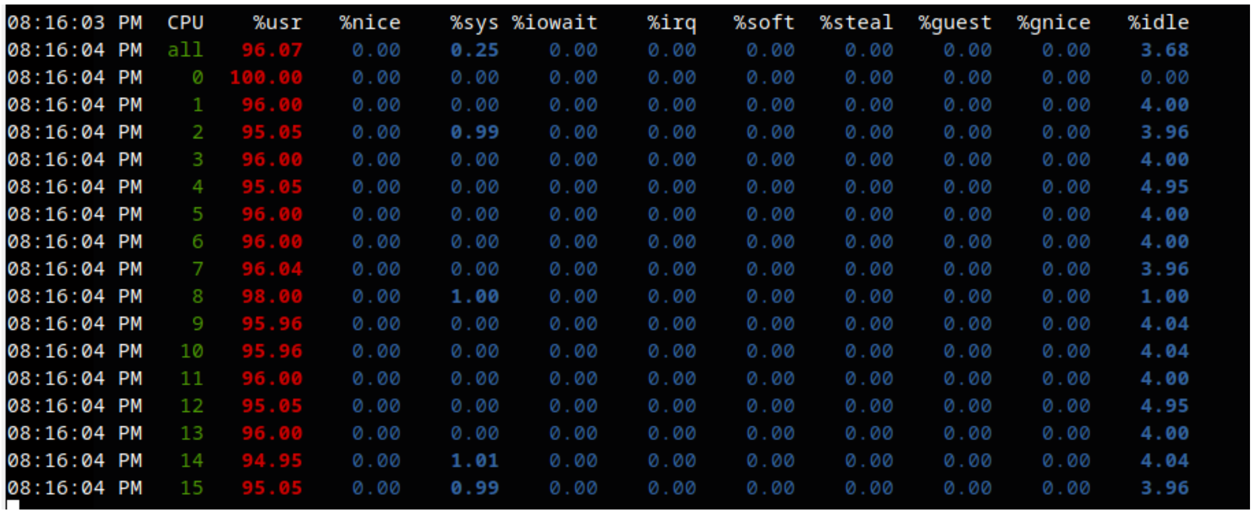


Comparative Speed Analysis.

My 94% parallelization vs 100%

My 94% parallelization vs theoretical best case scenario for FFTS of 96.7% according to https://lirias.kuleuven.be/retrieve/200966

In the end, we have 6% of the function that is serial and around 94% of it can be made parallel. The program made maximum use of all my cores. The beginning and end of the program run it is single-threaded, simply generating signals and writing them down. The actual FFT portion you can tell is running when all of your available cores spike in utilization together(or as many as you selected as an input parameter). Example derived by running **mpstat -P ALL 1**



```
08:16:03 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:16:04 PM  all   96.07    0.00    0.25    0.00    0.00    0.00    0.00    0.00    0.00    3.68
08:16:04 PM    0  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
08:16:04 PM    1   96.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.00
08:16:04 PM    2   95.05    0.00    0.99    0.00    0.00    0.00    0.00    0.00    0.00    3.96
08:16:04 PM    3   96.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.00
08:16:04 PM    4   95.05    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.95
08:16:04 PM    5   96.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.00
08:16:04 PM    6   96.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.00
08:16:04 PM    7   96.04    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    3.96
08:16:04 PM    8   98.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00    1.00
08:16:04 PM    9   95.96    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.04
08:16:04 PM   10   95.96    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.04
08:16:04 PM   11   96.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.00
08:16:04 PM   12   95.05    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.95
08:16:04 PM   13   96.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    4.00
08:16:04 PM   14   94.95    0.00    1.01    0.00    0.00    0.00    0.00    0.00    0.00    4.04
08:16:04 PM   15   95.05    0.00    0.99    0.00    0.00    0.00    0.00    0.00    0.00    3.96
```

The first few frames of mpstat will show only one core being utilized while that core generates the signal. Then when the FFT portion kicks in utilization spikes on all cores.

By shrinking the time to process a 4 min piece of “audio” from 20 seconds to 2 seconds plus some change we have not only produced a measurable difference in speed but one that could be felt any time an FFT needs to be performed.

Spreadsheets used to track data and calculate speed up based on Amdahl’s Law

https://docs.google.com/spreadsheets/d/1_v3FaX-F1XovysnmPOWQ9fInSAZIukWGBXyd2aWtQG0/edit?usp=sharing

Slides that were used in my video and to construct this document.

https://docs.google.com/presentation/d/1uNcFvTCgEbdALvfrYZXjPSW26iA1Jd4aeP6kOWofAo8/edit#slide=id.g107f9645bd5_0_15

Youtube Link(please do not share)

https://youtu.be/Asoy12T2egg