FFT’s

Fast Fourier Transform
Algorithms

What is a Fourier Transform

Well What is a Transform?

In this specific case it maps a function from its original function space into another function space via integration, where some of the properties of the original function might be more easily characterized and manipulated than in the original function space.

So what is a Fourier Transform?

A transform that decomposes a function into its constituent sinusoidal frequencies.Given the fact any function can be reproduced by adding sinusoidal functions together this can be useful for making sense of complex functions.

A great example of this is interpreting the waveform(signal or amplitude over time) of a musical chord and expressing it instead in terms of the volumes and frequencies of its constituent notes.

$$F(w) := F\{f(t)\} := \int_{-\infty} e^{-iwt} f(t)dt$$

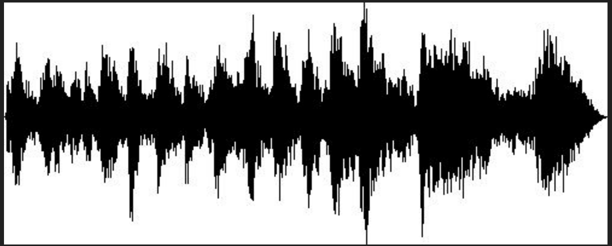
The function

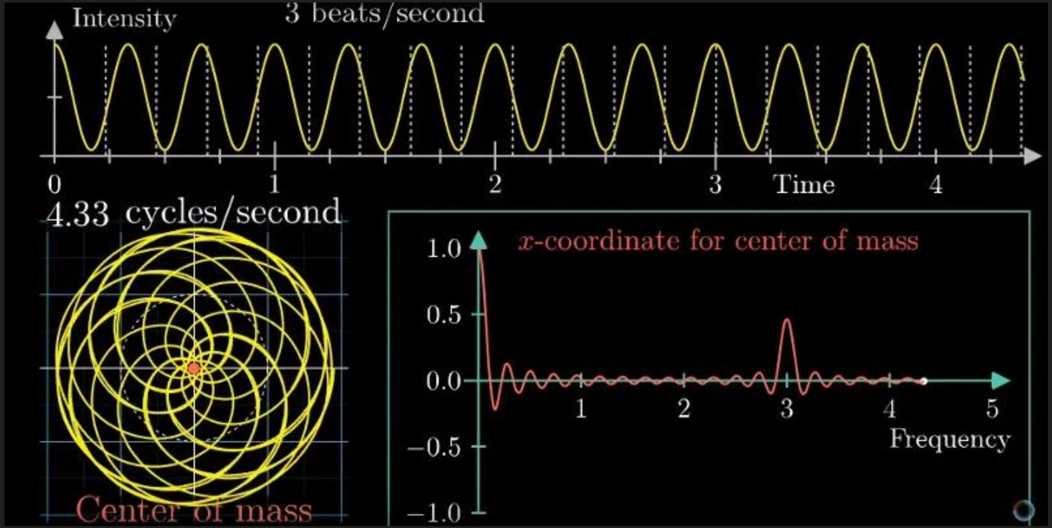
$$F^{-1}\{F(w)\} := \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{iwx} F(w)dw$$

Is called an inverse Fourier transform of $F(w)$.

Some clarification...

Up top and to the right here we have a simple sine wave. The sound we hear is generally made up of many of these layered together such that it results in something a little like this below.





This can be hard to make sense of and interpret but if we go back to the simple sine wave and apply a fourier transform to it you can see off to our right a visible spike highlighting the frequency of the sine wave above. You apply this a more complex signal and what you get is a series of spikes that indicate the volume of all the contributing sinusoidal frequencies that make up the complete sound.

Applications

FFT’s get used in digital recording, sampling, additive synthesis and pitch correction software

The FFT's importance derives from the fact that it has made working in the frequency domain equally computationally feasible as working in the temporal or spatial domain. Some of the important applications of the FFT include

Fast large-integer and polynomial multiplication

Efficient matrix-vector multiplication for Toeplitz, circulant and other structured matrices

Filtering algorithms (see overlap-add and overlap-save methods)

Fast algorithms for discrete cosine or sine transforms (e.g. fast DCT used for JPEG and MPEG/MP3 encoding and decoding)

Fast Chebyshev approximation

Solving difference equations

Computation of isotopic distributions.

So how do you pull this off as an algorithm

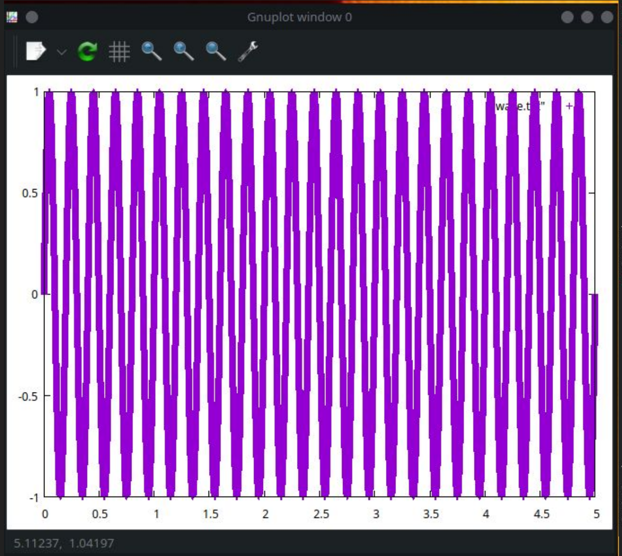
```
algorithm iterative-fft is

    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

        ω_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 ω_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]
```

At their simplest a Fast Fourier Transform creates a discrete Fourier transform of the input by using a recursive divide and conquer approach

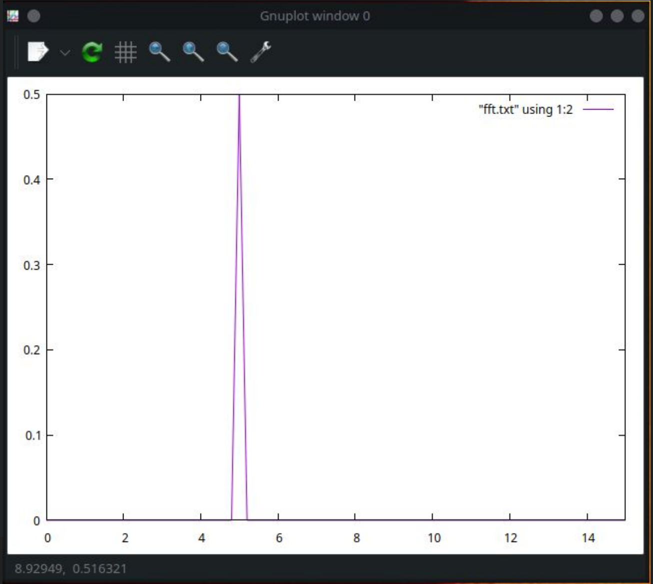
This approach generally results in O(NlogN). There exists FFT algorithms that avoid recursion such as the iterative radix-2 variant listed to the left. This is the variation I decided to try and make parallel as it’s design allows for better optimization than some of the simpler FFT’s.

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>
```



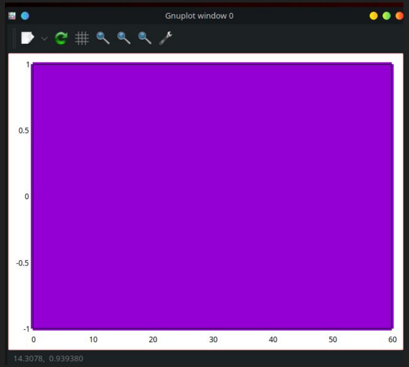
Use Case Elaborated Pt 1

```
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 3.912142807 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$
```

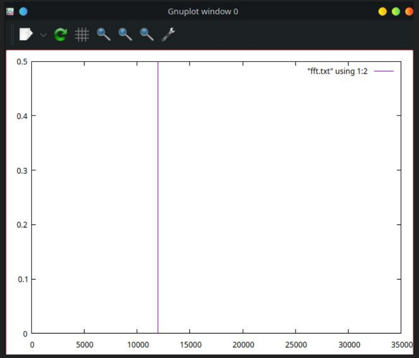
We can even identify octaves in noise!

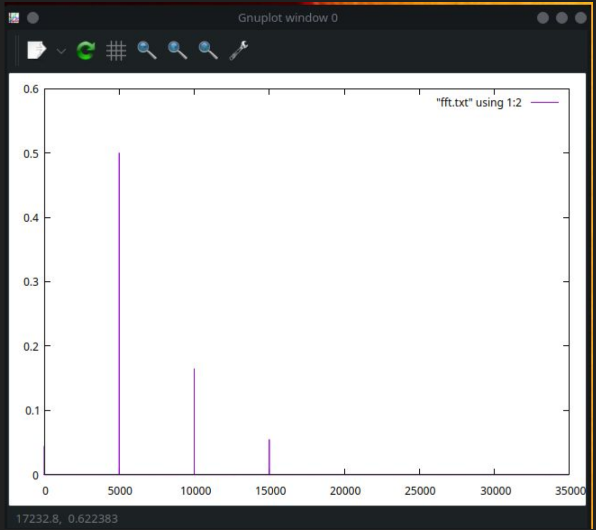
```
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.849277698 seconds.
jkl@funRuiner:~/Documents/final_proj/src$ ./serial_fft -noisy 60 5000
FFT perfomed on 60 second sample of 5000 cycles per second octaves burried in noise  in 3.820664707 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 3.664786936 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>
```

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





Let’s speed it up!

```
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 18.874478677 seconds, using 1 threads
jkl@funRuiner:~/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 18.261632267 seconds, using 1 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 10.49084386 seconds, using 2 threads
jkl@funRuiner:~/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 9.998158684 seconds, using 2 threads
jkl@funRuiner:~/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 10.58736048 seconds, using 2 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 5.388392753 seconds, using 4 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 3.74729105 seconds, using 8 threads
jkl@funRuiner:~/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 3.162287726 seconds, using 8 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.690864004 seconds, using 12 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.734589907 seconds, using 12 threads
jkl@funRuiner:~/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 2.633008943 seconds, using 12 threads
jkl@funRuiner:~/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 2.642461730 seconds, using 12 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.364861097 seconds, using 16 threads
jkl@funRuiner:~/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 2.364800174 seconds, using 16 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.364809566 seconds, using 16 threads
jkl@funRuiner:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
jkl@funRuiner:~/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@funRuiner:~/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 2.690864004 seconds, using 12 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.734589907 seconds, using 12 threads
jkl@funRuiner:~/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 2.633008943 seconds, using 12 threads
jkl@funRuiner:~/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 2.642461730 seconds, using 12 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.364861097 seconds, using 16 threads
jkl@funRuiner:~/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 2.364800174 seconds, using 16 threads
jkl@funRuiner:~/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@funRuiner:~/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@funRuiner:~/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 2.364809566 seconds, using 16 threads
jkl@funRuiner:~/Documents/final_proj/src$ ./omp_fft -noisy 240 12000 16
```

Speed Up: A Closer Look

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

Amhdahls Law Speed UP at

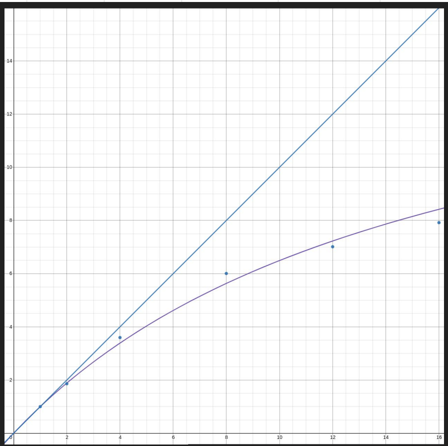
	Action Time	Actual Speed up	85% Assumed	90% Assumed	95% Assumed	
1	18.867	**1**	1	1	1	1
2	10.133	**1.861**	1.74	1.82	1.9	
4	5.245	**3.597**	2.76	3.08	3.48	
8	3.14	**6.008**	3.9	4.71	5.93	
12	2.69	**7.013**	4.53	5.71	7.74	
16	2.382	**7.92**	4.92	6.4	9.14	

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.

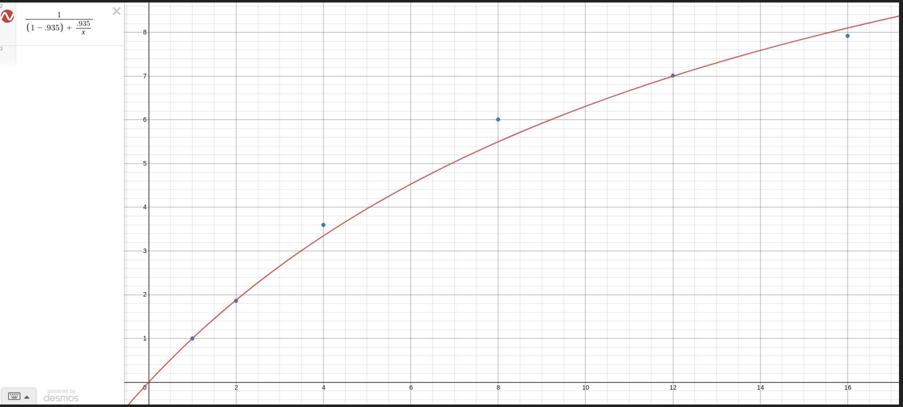
Notes on Data Selection

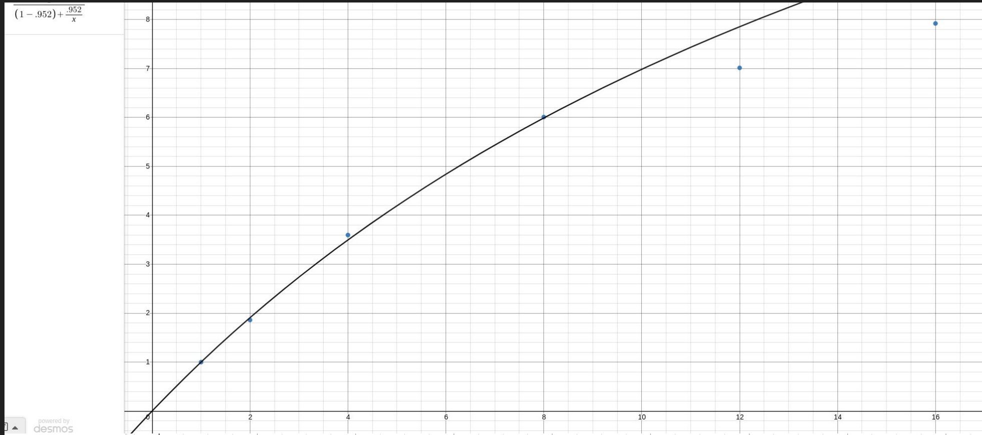
After doing 8 consecutive runs per thread alotment I did another set of runs for outlier replacement. Selectively replacing any obvious high point outliers from each category The idea is there is unlikely to be serious error of measurement in the negative direction but competing resources on my machine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

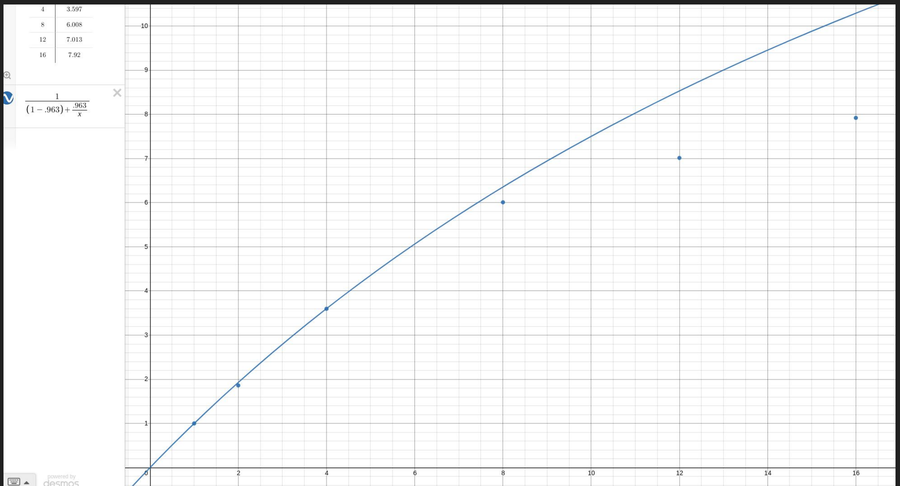


I used a table to estimate parallelization against my real world data and kept adjusting P until I reached the best fit I could. There was no perfect P that fit all data points perfectly but 94% seems to be the sweet spot. It’s notable performance curve worsens as thread count gets closer to the total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

Matching Each Point to an Efficiency Curve

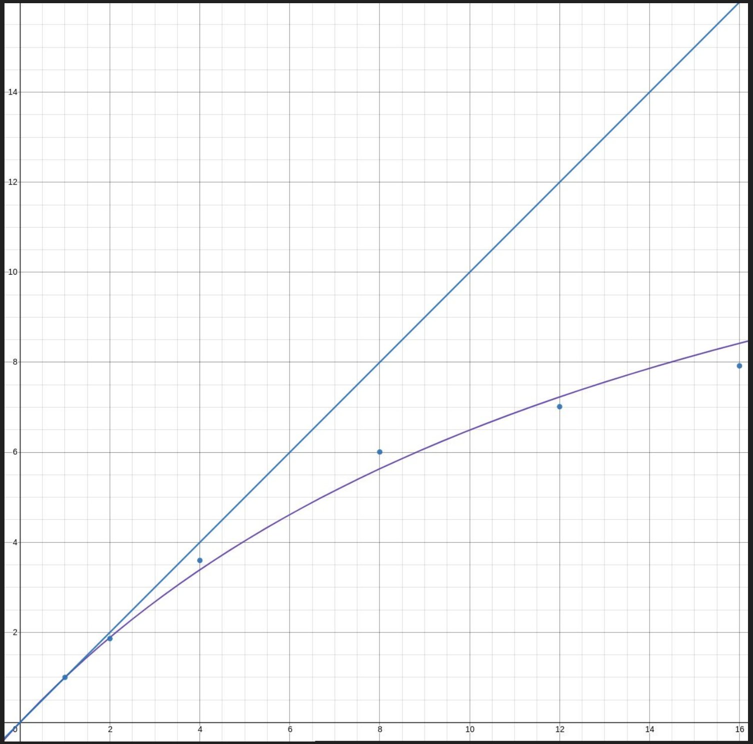






Comparative Speed Analysis.

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