Variations and Applications of the Fast Fourier Transform Algorithms

K. Marcolini, University of Miami

***Abstract*—This paper will cover basic material relating to the Fast Fourier Transform (FFT). It will begin going over its history and what the FFT actually accomplishes, and then proceed into the many different versions developed, specifically focusing on differences in algorithms and computational complexity, as well as primary applications where each is used. Discussed at the end will be potential future advancements and applications of the FFT.**

***Index Terms*—Fast Fourier Transform (FFT), computational complexity, applications of FFT**

I. Introduction

The Fourier Transform is a mathematical operation to express any function with regards to time as a function with regards to frequency. This is essentially useful is signal processing, as all signals being analyzed are functions of time, whether they be sound waves that are audible to humans, electromagnetic waves for communication devices in the gigahertz range (10^9), or up to X-Rays in the petahertz range (10^{15}). Analyzing a signal with respect to frequency rather than time can tell a lot about a source, whether it's a broad sound, a specific color, or a radio signal at a specific frequency.

The Discrete Fourier Transform (DFT) is essentially a Fourier Transform that takes a discrete-time input and transforms it to frequency. Typically this is used for any continuous signals, which are sampled over a finite duration of time to be converted to frequency. Although effective in frequency analysis, performing the mathematical operations was very computationally costly.

A Fast Fourier Transform (FFT) algorithm is any algorithm that improves the complexity of the original DFT while still effectively transforming a signal to its frequency domain. The premise behind the FFT was thought up in the early 1800s by Carl Gauss, but was further developed in the 1960s by James Cooley and John Tukey. Since being investigated, applications for the FFT have included everything from transient and frequency analysis of discrete audio signals, to spectral analysis in chemistry, which analyzes chemical properties of compounds by looking at their optical spectrum.

II. Different Versions of FFT

A. DFT - Original Development of the FFT

Any algorithm that can perform a DFT who's upper complexity is on the lines of $O(N\log_2 N)$ is considered to be an FFT. The original DFT algorithm is given by:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi k \frac{n}{N}} \qquad k = 0, \ldots, N-1. \tag{1}$$

This process takes a total of N operations to evaluate the sum, which will leave N total output terms. In summing all the terms, it is clear that the basic DFT algorithm has a computational complexity of $O(N^2)$.

B. Cooley-Tukey FFT

The most common implementation of the FFT is the original algorithm proposed by Cooley and Tukey back in 1965. They were able to reinvent Gauss's findings, adding in newer research on its complexity and computation requirements. The algorithm works recursively, in that if the DFT is size N, you can split it up into two smaller DFTs, N_1 and N_2, such that $N = N_1 N_2$.

This recursive method, the radix-2 case, typically assigns N_1 and N_2 to $N/2$. This will give two summations; both are similar-looking to the DFT algorithm, but one will be summed over the even indices ($n=2b$) and one over the odd ($n=2b+1$):

$$X_k = \sum_{b=0}^{\frac{N}{2}-1} x_{2b} e^{-\frac{j2\pi}{N}(2b)k} + \sum_{b=0}^{\frac{N}{2}-1} x_{2b+1} e^{-\frac{j2\pi}{N}(2b+1)k}. \tag{2}$$

This algorithm is considered divide-and-conquer because by splitting up the DFT into two smaller functions, computations become uncomplicated, leading the complexity of this algorithm to converge to $O(N\log_2 N)$.

C. Prime-Factor FFT

Similar to Cooley-Tukey's algorithm, the Prime-Factor FFT splits a size-N DFT into two separate DFTs, but this time, $N1$ and $N2$ are both relatively prime, rather than both equaling $N/2$. Two numbers are relatively prime if their only common factor is 1. This algorithm served as motivation to Cooley and Tukey's widely used FFT algorithm.

In order to express the Prime-Factor algorithm, the inputs and outputs need to be redefined altogether. The input, n, and output, k, would respectively become:

$$n = n_1 N_2 + n_2 N_1 \mod N,$$
$$k = k_1 N_2^{-1} N_2 + k_2 N_1^{-1} N_1 \mod N. \quad (3)$$

By reassigning these values, the new FFT becomes:

$$X_{k_1 N_2^{-1} N_2 + k_2 N_1^{-1} N_1} = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} x_{n_1 N_2 + n_2 N_1} e^{-\frac{j2\pi}{N_2} n_2 k_2} \right) e^{-\frac{j2\pi}{N_1} n_1 k_1}. \quad (4)$$

Although this is still a O(Nlog$_2N$) algorithm, it only works with relatively prime N's, and its accuracy is questioned. It has been combined with Cooley and Tukey's FFT for the case when N_1 and N_2 are relatively prime. Because of the task of having to re-index and re-express the DFT each time, this algorithm is mostly never used by itself.

D. Bruun's FFT

Bruun's FFT algorithm is fairly new compared to the others listed in this study. It works recursively, factorizing real polynomials in the transformation, and does not bother with imaginary numbers until the last stage of the summation. Using the expression for the DFT, Bruun's algorithm starts by renaming the exponential function with imaginary parts:

$$\omega_N^n = e^{-\frac{j2\pi}{N} n} \qquad n = 0, \ldots, N-1. \quad (5)$$

Bruun then defined a polynomial, $x(z)$, given by:

$$x(z) = \sum_{n=0}^{N-1} x_n z^n. \quad (6)$$

By reducing this polynomial farther, you get the solution to this FFT,

$$X_k = x\left(\omega_N^k\right) = x(z) \mod\left(z - \omega_N^k\right), \quad (7)$$

where "mod" is simply the remainder of this reduction.

While this algorithm is fast, it is not the most accurate, therefore making it more efficient to use the Cooley-Tukey FFT. Because of this, it is hardly used in practical applications.

E. Rader's FFT

The Rader FFT is an algorithm that works by changing the DFT into a circular convolution. It only works for prime bases of the DFT, and typically it is more effective and simpler to use the Cooley-Tukey algorithm. The only real use for Rader's FFT is for large prime bases. It works by re-indexing n and k of the DFT. The new n becomes $n = g^q \bmod(N)$, and the new k becomes $k = g^{-p} \bmod(N)$. The DFT is transformed to:

$$X_0 = \sum_{n=0}^{N-1} x_n,$$
$$X_{g^{-p}} = x_0 + \sum_{q=0}^{N-2} x_{g^q} e^{-\frac{j2\pi}{N} g^{-(p-q)}} \qquad p = 0, \ldots, N-2. \quad (8)$$

The result of this FFT sum is a circular convolution of two sequences, both N–1 in size. These sequences, labeled a_q and b_q, are given by:

$$a_q = x_{g^q}, \quad b_q = e^{-\frac{j2\pi}{N} g^{-q}}. \quad (9)$$

In doing this convolution, N–1 may have large prime factors, making it necessary to perform this algorithm recursively. This adds to the complexity of it, making it O(Nlog$_2N$) plus O(N) extra operations. Thus, in practice, this algorithm is not typically used a lot, because of the extra operations and computations needed.

F. Bluestein's FFT

Like Rader's FFT, Bluestein's algorithm evaluates using circular convolution. The difference it works for arbitrary-sized DFTs rather than solely prime-sized ones. Bluestein's algorithm works by re-indexing and re-expressing the DFT sum, like before with Rader's. The product in the DFT, nk, is now replaced with:

$$nk = -\frac{(n-k)^2 + n^2 + k^2}{2}. \quad (10)$$

Now the new transform becomes:

$$X_k = e^{-\frac{j\pi}{N} k^2} \sum_{n=0}^{N-1} \left(x_n e^{-\frac{j\pi}{N} n^2} \right) e^{\frac{j\pi}{N}(k-n)^2} \qquad k = 0, \ldots, N-1. \quad (11)$$

The resultant convolution sequences, both length-N, that emerge from this summation are:

$$a_n = x_n e^{-\frac{j\pi}{N} n^2} \; m, \quad b_n = e^{-\frac{j\pi}{N} n^2}. \quad (12)$$

Now the algorithm can be rewritten using these sequences,

$$X_k = b_k^* \sum_{n=0}^{N-1} a_n b_{k-n} \qquad k = 0, \ldots, N-1, \quad (13)$$

where b_k^* is the phase factor of the FFT.

The Bluestein FFT is still a O(Nlog$_2N$) algorithm, except in practice, it performs slower than the Cooley-Tukey FFT, making it more uncommon in practical applications.

One usage of the Bluestein algorithm is for Z-Transforms. It can be rewritten as:

$$X_k = \sum_{n=0}^{N-1} x_n z^{kn} \qquad k = 0, \ldots, M-1, \quad (14)$$

for any number of N or M inputs and outputs. One name coined from this is the Bluestein Chirp Z-Transform. For $|z|=1$, b_m becomes a complex sine function. This causes the output to sweep frequencies, known as "chirping," which is why it is known as a Chirp Z-Transform.

III. Applications of the FFT

Briefly described at the beginning of this paper are some of the different applications where the FFT is used. These include audio and signal processing, communications, physics, optics, chemistry, among many others.

The FFT is an essential tool in audio when trying to analyze frequency content in digital signals. If an audio signal is viewed in time, it may resemble a sine wave, which could say a lot about its sound characteristics. Most other signals, though, could be a random conglomeration of changing peaks, making it more difficult to tell what exactly is going on. Therefore, by viewing an FFT plot in conjunction with its transient signal plot, it is easy to see all of the frequency content as well as its full time-domain signal.

For example, take a digital frequency equalization filter. It would be very abstract to try and view solely the time-domain signal of this, if you wanted to add some sort of band-stop or shelving filter. So instead, real-time FFTs are taken with the

signal playing, which allows you to view as well as hear the filtering. A screenshot of a multiband shelving and low/hi-pass filter is shown below.

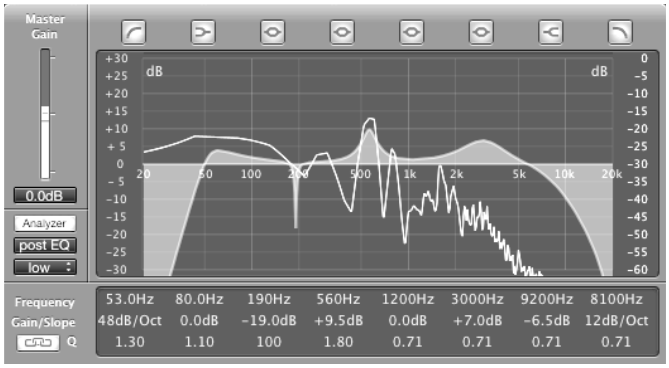

Fig. 1. Stereo equalizer typical in audio processing, showing spectrum.

Similar to audio, optics is a continuously growing field, which uses the FFT to analyze its frequency and wavelengths. If a sine wave is oscillating at 440Hz, you will hear a mid-pitched sine, and you could view an FFT graph which shows a spike at that frequency. The color red, however, is not audible. Taking an FFT of will give a spike right at that color's oscillating frequency, around 450 THz (10^{12}).

There are endless applications for the FFT in orthogonal frequency-division multiplexing (OFDM) communication technology. Digital signals being transferred wirelessly over multiple bands to other devices need some sort of transformation to be able send and receive this data. The FFT is the most efficient, real-time transformation method available.

A more specific, yet simple example of this is within wireless routers. It works on either one, or multiple frequency bands in the gigahertz range. It works by taking the inverse FFT of a data signal, which the sender can then modulate and transmit. The receiver grabs that signal, demodulates it, and then takes its FFT to encode it.

IV. Conclusion

While the idea of the FFT has been around for a long time, it is still being developed and optimized today. In January 2012, MIT came out with a new algorithm, the "Nearly Optimal Sparse Fourier Transform," which functions by splitting up the initial signal into smaller frequency bands to lower the amount of computation that goes into solving the FFT. This is more effective for audio and imaging, where frequencies are generally sparse and can be analyzed more easily. In splicing signals up continuously, the MIT researchers were able to determine the period of the waveform, which assisted in determining its most dominant frequency.

The paper proves that this method can achieve $O(k\log_2 N)$ and $O(k \log N \log_2(N/k))$ for the general case, where k is the number of non-zero Fourier coefficients. Both of these complexities improve upon the original $O(N\log_2 N)$ case.

Although not an ideal algorithm for everything, it does offer something completely fresh in the engineering world and opens the doors for more research in optimizing the FFT.

References

[1] Cooley, James W.; Lewis, Peter A. W.; Welch, Peter D.;, "The Fast Fourier Transform and Its Applications," Education, IEEE Transactions on , vol.12, no.1, pp.27-34, March 1969.

[2] Stone, H.S.; , "R66-50 An Algorithm for the Machine Calculation of Complex Fourier Series," Electronic Computers, IEEE Transactions on, vol.EC-15, no.4, pp.680-681, Aug. 1966.

[3] Chan, S.C.; Ho, K.L.; , "On indexing the prime factor fast Fourier transform algorithm," Circuits and Systems, IEEE Transactions on , vol.38, no.8, pp.951-953, Aug 1991.

[4] Bruun, G.; , "z-transform DFT filters and FFT's," Acoustics, Speech and Signal Processing, IEEE Transactions on , vol.26, no.1, pp. 56- 63, Feb 1978.

[5] Rader, C.; Brenner, N.; , "A new principle for fast Fourier transformation," Acoustics, Speech and Signal Processing, IEEE Transactions on , vol.24, no.3, pp. 264- 266, Jun 1976.

[6] Bluestein, L.; , "A linear filtering approach to the computation of discrete Fourier transform," Audio and Electroacoustics, IEEE Transactions on , vol.18, no.4, pp. 451- 455, Dec 1970.

[7] Rabiner, L.; Schafer, R.; Rader, C.; , "The chirp z-transform algorithm," Audio and Electroacoustics, IEEE Transactions on , vol.17, no.2, pp. 86-92, Jun 1969.

[8] H. Hassanieh, P. Indyk, D. Katabi, E. Price: "Nearly Optimal Sparse Fourier Transform," arXiv.org,/ abs/1201.2501v1, Cornell University Library, Jan. 2012.