Radix-2 Decimation-In-Time Fast Fourier Transform

Michael Balas

January 13, 2019

Background

The notation used in this paper is derived from the textbook: *Fast Fourier Transform: Algorithms and Applications.*

Introduction

The Fast Fourier Transform (FFT) refers to a family of algorithms that efficiently perform the discrete Fourier Transform (DFT). The FFT is one of the most important and ubiquitous tools used for data analysis and digital signal processing [8]. It was even included as one of the top ten algorithms of the 20^{th} century by the IEEE journal, published in *Computing in Science and Engineering.* [4]. Some of its extensive applications include: optical signal processing, pattern recognition, noise filtering, video/image compression, and medical imaging [8]. Many fast algorithms have been developed, such as the split radix, vector radix, prime factor, and Winograd Fourier transform algorithm, to name a few. This paper will examine the radix-2 decimation-in-time FFT algorithm, the most widely used FFT algorithm, developed by Cooley and Tukey in 1965 [6].

The Discrete Fourier Transform

Computers and digital processing systems have to process continuous signals using discrete methods. This requires that a finite number of samples are taken, and that the input signal be both time-limited and band-limited [7]. Although this is just an approximation, it works to the extent where the differences are unnoticeable. Using these assumptions, the DFT works to map a sequence in the time or spatial domain into the frequency domain [8]. The DFT and inverse discrete Fourier Transform (IDFT) can be described as follows:

$$\text{DFT:} \qquad X^F(k) = \sum_{n=0}^{N-1} x(n) e^{(\frac{-j2\pi}{N})kn},\ k = 0, 1, ..., N-1$$

$$\text{IDFT:} \qquad X(n) = \frac{1}{N} \sum_{k=0}^{N-1} X^F(k) e^{(\frac{j2\pi}{N})kn},\ n = 0, 1, ..., N-1$$

where $x(n),\ n = 0, 1, ..., N-1$ is a uniformly sampled sequence, $e^{(\frac{-j2\pi}{N})kn}$ is the N^{th} root of unity, $X^F(k), k = 0, 1, ..., N-1$ is the k-th DFT coefficient, and $j = \sqrt{-1}$ [8]. Essentially, the DFT turns a function of time into a function of frequency, while the IDFT does the reverse, each by decomposing a given function into a series of sinusoidal functions. For the sake of brevity, this paper will only focus on the forward DFT, though the FFT algorithms work on the IDFT using the same principles. The DFT is usually expressed with the *twiddle factor* [8]:

$$W_N^{kn} = exp\Big[(\frac{-j2\pi}{N})kn\Big]$$

$$X^F(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn},\ k = 0, 1, ..., N-1 \tag{1}$$

Computations that directly use the DFT require approximately N^2 operations for N data points [3]. This results in significantly long times for computation when N is large, highlighting the need for a faster implementation.

The Fast Fourier Transform

This algorithm follows a divide-and-conquer approach, whereby it decomposes an N-point sequence into two $N = 2$ sequences; one of even samples and another of odd samples [8, 9]. This FFT algorithm is thus called a *decimation-in-time* (DIT) algorithm [6]. It is assumed that the N-point sequence is an integer power of 2 ($N = 2^l,\ l = integer$), hence the name *radix-2* [6,8,9]. The N-point DFT can be expressed in terms of the DFTs of the decimated sequences as follows [8]:
Seperating $x(n)$ into its even and odd components:

$$X^F(k) = \sum_{n\,even} x(n) W_N^{kn} + \sum_{n\,odd} x(n) W_N^{kn}$$

Substituting variables $n = 2r$ for n even, and $n = 2r + 1$ for n odd:

$$\begin{aligned} &= \sum_{r=0}^{(N/2)-1} x(2r) W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1) W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x(2r) (W_N^2)^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1) (W_N^2)^{rk} \end{aligned} \tag{2}$$

Considering that:

$$W_N^2 = \left[\frac{-j2(2\pi)}{N}\right] = \left[\frac{-j2\pi)}{N/2}\right] = W_{N/2}$$

The equation can be written as:

$$X^F(k) = \sum_{r=0}^{(N/2)-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1) W_{N/2}^{rk} \tag{3}$$

$$= G^F(k) + W_N^k H^F(k),\ k = 0, 1, ..., \frac{N}{2} - 1 \tag{4}$$

Where $G^F(k)$ and $H^F(k)$ are the $N/2$-point DFTs of the even and odd samples of $x(n)$, respectively [8]. Since $X^F(k)$ is periodic with period N, $X^F(k) = X^F(k+N)$ [8]. Additionally, $G^F(k)$

and $H^F(k)$ are periodic with period $\frac{N}{2}$, and can thus be expressed as: $G^F(k) = G^F(k + \frac{N}{2})$ and $H^F(k) = H^F(k + \frac{N}{2})$ [5]. These properties of symmetry are used as follows [8]:

$$X^F(k + N/2) = G^F(k) + W_N^{k+N/2} H^F(k) \tag{5}$$

Considering that:

$$W_N^{N/2} = exp\Big(\frac{-j2\pi}{N}\frac{N}{2}\Big) = exp(-j\pi) = -1 \tag{6}$$

Using the identity from (6), $W_N^{k+N/2}$ can be written as:

$$W_N^{k+N/2} = exp\Big[\frac{-j2\pi}{N}\Big(k + \frac{N}{2}\Big)\Big] = W_N^k W_N^{N/2} = -W_N^k \tag{7}$$

Using equation (4) and equation (5) substituted with (7), the entire procedure is written as:

$$\begin{cases} X^F(k) = G^F(k) + W_N^k H^F(k), & k = 0, 1, ..., \frac{N}{2} - 1 \\ X^F(k + \frac{N}{2}) = G^F(k) - W_N^k H^F(k), & k = 0, 1, ..., \frac{N}{2} - 1 \end{cases} \tag{8}$$

The direct computation of $G^F(k)$ and $H^F(k)$ each requires $(\frac{N}{2})^2$ complex multiplications and $(\frac{N}{2})^2$ complex additions [8]. Moreover, the $N/2$-point DFTs must then be combined, entailing N complex multiplications (due to the multiplications of the odd sum with the twiddle factor), and N complex additions (due to the additions of the previous product with the even sum) [5]. Thus, the computation of $X^F(k)$ requires $(\frac{N}{2})^2 + (\frac{N}{2})^2 + N = \frac{N^2}{2} + N$ complex multiplications and additions. When N is large, this is already about half of the operations required to directly compute the DFT.

Having performed the DIT once, further savings can be achieved by repeating the process and decomposing the sequence into successively smaller subsequences of even and odd samples [5,8]. The decimation is repeated until two-point sequences are obtained [8]. For $N = 2^v$, this decimation can be performed $v = log_2 N$ times [5]. Thus the total computational complexity is $O(Nlog_2 N)$ because there are $log_2 N$ splitting steps, in which each step is $O(N)$ [1]. This is much faster than directly computing the DFT, $O(N^2)$. The difference between both methods is outlined in Table 1.

Table 1: Comparison of Complexity

Algorithmic Complexity	# of Computations		
# of Samples: N	10^3	10^6	10^9
Direct DFT: N^2	10^6	10^{12}	10^{18}
Radix-2 DIT FFT: $Nlog_2N$	10^4	20×10^6	30×10^9

Assuming one nanosecond per computation, it would require approximately 30 years using the DFT directly for a sample of size 10^{18}. Conversely, it would require only about 30 seconds using the FFT. Below, an iterative implementation of the FFT is given in pseudo-code. For simplicity, the code assumes that the twiddle factors are stored in the array, w, in bit-reversed order [2]. The term *butterfly* refers to the basic computational step of the algorithm. The pseudo-code is adapted from Chu and George's book: *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms.*

Algorithm 1 The iterative radix-2 DIT FFT algorithm in pseudo-code

```
1: begin
2: PairsInGroup ← N/2                          ▷ Begin with N/2 butterflies in one group
3: NumOfGroups ← 1                             ▷ Same twiddle factor is employed in a group
4: Distance = N/2
5: while NumOfGroups < N do
6:     for K ← 0 to NumOfGroups − 1 do         ▷ Combine pairs in each group
7:         JFirst ← 2 * K * PairsInGroup
8:         JLast ← JFirst + PairsInGroup − 1
9:         Jtwiddle ← K                        ▷ Access consecutive w[m]
10:        W ← w[Jtwiddle]                     ▷ Assume w[m] = ω_N^ℓ , m bit-reverses ℓ
11:        for J ← JFirst to JLast do
12:            Temp ← W * a[J + Distance]
13:            a[J + Distance] ← a[J] − Temp
14:            a[J] ← a[J] + Temp
15:        end for
16:    end for
17:    PairsInGroup ← PairsInGroup/2
18:    NumOfGroups ← NumOfGroups * 2
19:    Distance ← Distance/2
20: end while
21: end
```

Conclusion

By efficiently implementing the DFT, the Cooley-Tukey FFT algorithm will remain as one of the most important discoveries in computer science. In addition to reducing the computational complexity of an N-point DFT to $Nlog_2N$ complex operations, the FFT reduces storage requirements and minimizes the risk of computational errors due to finite bit length arithmetic [8]. With the exponential rate of technological advancement, active research is still being pursued to find even more efficient means of processing digital signals.

References

[1] Jörg Arndt. *Matters Computational: ideas, algorithms, source code*. Springer Science & Business Media, 2010.

[2] Eleanor Chu and Alan George. *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC Press, 1999.

[3] James W Cooley, Peter AW Lewis, and Peter D Welch. The fast fourier transform and its applications. *IEEE Transactions on Education*, 12(1):27–34, 1969.

[4] Jack Dongarra and Francis Sullivan. Guest editors' introduction: The top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000.

[5] Madhavi S Kapale and Nilesh P Bodne. Design & simulation parallel pipelined radix-2ˆ 2 fft architecture for real valued signals.

[6] Douglas A Lyon. The discrete fourier transform, part 2: Radix 2 fft. *journal of object technology*, 8(5), 2009.

[7] Brad Osgood. The fourier transform and its applications. *Lecture Notes for EE*, 261:20, 2009.

[8] Kamisetty Ramamohan Rao, Do Nyeon Kim, and Jae Jeong Hwang. *Fast Fourier transform-algorithms and applications*. Springer Science & Business Media, 2011.

[9] Ali Saidi. Decimation-in-time-frequency fft algorithm. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume 3, pages III–453. IEEE, 1994.