CS260 PROJECT PROPOSAL: COMPARISON OF COOLEY–TUKEY ALGORITHM AND RADIX-4 FFT ALGORITHM FOR FFT

Hao Liu(185896) Mohamed Bouaziz(184732) Tongzhou Gu(186085) Juyi Lin(187176)
Jinjie Mai(179794)
{hao.liu, mohamed.bouaziz, tongzhou.gu, juyi.lin, jinjie.mai}@kaust.edu.sa

1 Introduction

Fourier Transform (FT) is one of the most well-known mathematical transforms that convert a continuous input function from the time domain into the frequency domain. From the perspective of an engineer, with Fourier Transform, a function of analog signal strength against time could be converted into a function against frequency, or vice versa, which means any signal could be seen as a combination of an infinite number of sine waves at different frequencies. For a discrete input function, such as a function describing a digital signal waveform, there is a "discrete version" of Fourier Transform named Discrete Fourier Transform (DFT), and a discrete function could be regarded as the sum of a finite number of sine waves, depending on the number of the sampling points. In practical use of Discrete Fourier Transform, we also have its efficient implementation known as Fast Fourier Transform (FFT).

The industry has widely adopted fast Fourier Transform as one of the most commonly-used algorithms. It has demonstrated its massive value in signal processing, especially for wireless communication, as well as digital image and audio processing. For example, Orthogonal frequency division multiplexing (OFDM)[1] and its derived method Orthogonal Frequency Division Multiple Access (OFDMA)[2] is used by modern wireless communication technology such as LTE Mobile Network[3] and Wi-Fi 6 Wireless LAN[4] to improve bandwidth utilization and achieve a higher transmission rate. This method heavily utilizes FFT to modulate and demodulate the signal. Another famous application is image and audio compression. Derived from FFT, Discrete Cosine Transform (DCT) is another transform that can deliver a higher compression ratio than DFT when used by compression algorithms. One Fast DCT implementation is based on FFT with additional pre- and post-processing[5]. So far, many mainstream image and audio compression formats, including MP3 for audio, JPEG for images, and MPEG for video, are DCT-based[6].

Among the FFT implementations, Cooley-Tukey FFT[7] is considered the standard implementation, and there are also several variants, including Radix-4 FFT[8], Rader's FFT[9], Winograd FFT[10], Quantum Fourier transform (QFT)[11], and Split-radix FFT[12]. This project aims to analyze and re-implement Cooley-Tukey FFT and Radix-4 FFT algorithms. Then we are devoted to making a detailed comparison between them through theoretical and experimental analysis.

2 Discrete Fourier Transform (DFT)

2.1 Mathematical Problem Formulation

In this section, we give a mathematical formulation of discrete Fourier transform (DFT) [13] and explain the naive DFT algorithm in detail.

For brevity, we consider the one-dimensional case of DFT in the following.

Suppose that we have a discrete sequence

$x_n = x[0], x[1], x[2], ...x[n-1]$ **in length** N**.**

DFT is a function $\mathcal{F}$, which will transform the given sequence $x\{n\}$ to a new sequence,

$X_n = X[0], X[1], X[2], ...X[n-1]$**, also in length** N**.**

Formally, DFT function $\mathcal{F}$ is defined by:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n] \cdot e^{-i2\pi\frac{kn}{N}}, \quad n \in \mathbb{Z}, \tag{1}$$

where i is the complex number.

The inverse DFT $\mathcal{F}^{-1}$, alias Inverse Discrete Fourier Transform (IDFT), is the function that can reconstruct $x\{n\}$ from $X\{n\}$ with no loss of information. Similarly, we can write this process as:

$$x_k = \sum_{n=0}^{N-1} X_n \cdot e^{i2\pi\frac{kn}{N}}, \quad n \in \mathbb{Z}, \tag{2}$$

Moreover, we have Euler's formula:

$$e^{ix} = \cos x + i \sin x \tag{3}$$

Combing Eq. 3 and Eq. 1, we can also write DFT function $\mathcal{F}$ from the complex plane to the trigonometric domain as:

$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n \cdot \left[\cos(\frac{2\pi}{N}kn) - i \cdot \sin(\frac{2\pi}{N}kn)\right] \tag{4}$$

2.2 Complexity Analysis of Naive DFT

```python

def dft(x):
    '''Compute the Discrete Fourier Transform of an input signal x of N samples.'''
    N = len(x)
    X = np.zeros(N, dtype=np.complex)
    # Compute each X[m]
    for k in range(N):
        # Compute similarity between x and the m'th basis
        for n in range(N):
            X[k] = X[k] + x[n] * np.exp(-2j * np.pi * k * n / N)
    return X
```

Listing 1: DFT Implementation

Algorithm 1: A naive implementation for DFT

Data: Input discrete sequence $x_n = x[0], x[1], x[2], ...x[n-1]$ in length N
Result: Output sequence $X_n = X[0], X[1], X[2], ...X[n-1]$, also in length N
Init: Initialize $X_n = X[0], X[1], X[2], ...X[n-1]$ as zeros

1 **for** $k \leftarrow 0$ **to** $N-1$ **do**
2   **for** $n \leftarrow 0$ **to** $N-1$ **do**
3         $X[k] = X[k] + x[n] \cdot e^{-i2\pi\frac{kn}{N}}$
4   **end**
5 **end**
6 **return** X_n

$$N \times N = N^2 = O(N^2)$$

$$MN \times MN = (MN)^2 = O((MN)^2)$$

2.3 DFT Matrix

Considering Eq. 1, if we denote the last term as:

$$W_N = e^{-i2\pi/N} = \cos(2\pi N) - i\sin(2\pi/N) \tag{5}$$

We can rewrite Eq. 1 using Eq. 5 as:

$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n \cdot W_N^{kn}, \quad n \in \mathbb{Z}, \tag{6}$$

Using this notation Eq. 5, we can explicitly write DFT as a linear transformation. For instance, if we have $x\{n\}$ with $n = 5$, then $W = W_5$, we can expand DFT as following:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W^6 & W^8 \\ 1 & W^3 & W^6 & W^9 & W^{12} \\ 1 & W^4 & W^8 & W^{12} & W^{16} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{7}$$

Another interesting fact is that $g(k) = W_N^{nk}$ is a periodic function with period N. We will find writing DFT in matrix form is more intuitive to lead us to go from DFT to FFT.

2.4 Two-Dimensional DFT

Signal and image processing problems are usually dealing with multi-dimensional data.

Assuming that we are processing an image $f(x, y)$ of size $M \times N$, the two-dimensional DFT is defined as:

$$\mathcal{F}(u, v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x, y)e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{8}$$

where $x, u = 0, 1, 2, ..., M-1$ and $y, v = 0, 1, 2, ..., N-1$.

Similar to the conversion from Eq. 1 to Eq. 2, the inverse of 2D DFT can be written as:

$$f(x, y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \mathcal{F}(u, v)e^{i2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{9}$$

where $x, u = 0, 1, 2, ..., M-1$ and $y, v = 0, 1, 2, ..., N-1$.

Comparing Eq. 1 with Eq. 8, we can observe the computing efficiency of transforming high dimensional data from time domain to frequency domain becomes tedious with naive DFT. That's why developing fast Fourier transform (FFT) algorithm is so essential for practical application.

3 Description of Algorithms

3.1 Algorithms 1: Cooley–Tukey algorithm

This famous algorithm was invented by Cooley, and Tukey [7], engineers at the IBM research center in the early 1960s. It has had a considerable impact on the development of digital signal processing applications due to its efficiency [14]. A discrete Fourier transform calculation is a product of a matrix and a vector. It, therefore, requires T^2 multiplications/additions of complex numbers (T is the order of the matrix).

Assuming that a computer performs 10^9 operations per second, a transform calculation on a signal of $T = 10^3$ samples will require 10^{-3}s. A calculation on an image of size $T \times T = 10^6$ will require T^4 or 10^{12} operations which costs fifteen minutes. If we consider processing data in a three-dimensional domain (on vectors of size $T \times T \times T$), we would need T^6 or 10^{18} operations, which require a few decades.

The fast Fourier transform considerably reduces the number of operations: instead of performing T^2 operations, it will suffice to perform $T \log_2 T$. In the three previous examples, we will have to perform 10^4, 2×10^7 and 3×10^{10} operations which will require respectively $10^{-5}s$, $2 \times 10^{-2}s$ and $30s$. To explain this algorithm, we will use recursion by showing that the computation of a Fourier transform of size T comes down to the computation of two Fourier transforms of size $T/2$ followed by $T/2$ multiplications.

We want to calculate for $k = 0, \cdots, T-1$

$$X(k) = \sum_{t=0}^{T-1} x(t) \exp(-2\pi j \frac{k.t}{T}) \tag{10}$$

We pose $t = 2n$ if t is even and $t = 2n+1$ if t is odd. $X(k)$ is then written by posing $N = T/2$

$$X(k) = \sum_{n=0}^{N-1} x(2n) \exp(-2\pi j \frac{k.2n}{T}) + \sum_{n=0}^{N-1} x(2n+1) \exp(-2\pi j \frac{k.(2n+1)}{T}) \tag{11}$$

Let us name the sequences.

$$\begin{gathered} t = 0, \cdots, 2N-1 : x_{2N}(t) = x(t) \\ n = 0, \cdots, N-1 : x_N^o(n) = x(2n) \\ n = 0, \cdots, N-1 : x_N^i(n) = x(2n+1) \\ k = 0, \cdots, 2N-1 : X_{2N}(k) = X(k) \end{gathered}$$

With these notations, we get

$$X_{2N}(k) = \sum_{n=0}^{N-1} x_N^o(n) \exp\left(-2\pi j \frac{k \cdot n}{N}\right) + \sum_{n=0}^{N-1} x_N^i(n) \exp\left(-2\pi j \frac{k \cdot n}{N}\right) \exp\left(-2\pi j \frac{k}{2N}\right) \tag{12}$$

In the second summation of the right-hand side of the equation, the factor $exp(-2\pi j \frac{k}{2N})$ does not depend on n. We therefore have the following equation for $k = 0, \cdots, 2N-1$.

$$X_{2N}(k) = \left[\sum_{n=0}^{N-1} x_N^o(n) \exp\left(-2\pi j \frac{k \cdot n}{N}\right)\right] + \exp\left(-2\pi j \frac{k}{2N}\right) \left[\sum_{n=0}^{N-1} x_N^i(n) \exp\left(-2\pi j \frac{k \cdot n}{N}\right)\right] \tag{13}$$

If $0 \leq k \leq N-1$, we recognize in the two expressions between square brackets the discrete Fourier transforms of the sequences of the even-numbered samples $x_N^o(n)$ and the odd-numbered samples $x_N^i(n)$ which we call $X_N^o(k)$ and $X_N^i(k)$

Therefore, for $k = 0, \cdots, N-1$

$$X_{2N}(k) = X_N^o(k) + \exp(-\pi j \frac{k}{N}) X_N^i(k) \tag{14}$$

When $N \leq k \leq 2N-1$, we can write

$$k = \ell + N \tag{15}$$

and notice that

$$\exp(-\pi j \frac{k}{N}) = -\exp(-\pi j \frac{\ell}{N}) \tag{16}$$

Therefore, for $\ell = 0, \cdots, N-1$, the following equation is used

$$X_{2N}(\ell + N) = \left[\sum_{n=0}^{N-1} x_N^o(n) \exp(-2\pi j \frac{(\ell + N).n}{N})\right] + \exp\left(-2\pi j \frac{\ell + N}{2N}\right) \left[\sum_{n=0}^{N-1} x_N^i(n) \exp(-2\pi j \frac{(\ell + N).n}{N})\right] \tag{17}$$

and noting that

$$\exp(-2\pi j \frac{(\ell + N).n}{N}) = \exp(-2\pi j \frac{\ell.n}{N}) \tag{18}$$

we have an analogous writing for $\ell = 0, \cdots, N-1$

$$X_{2N}(\ell + N) = X_N^o(\ell) - exp(-\pi j \frac{\ell}{N}) X_N^i(\ell) \tag{19}$$

We can change the name of the variable ℓ to k and combine the two equations $X_{2N}(k)$ and $X_{2N}(\ell + N)$ for $k = 0, \cdots, N-1$

$$X_{2N}(k) = X_N^o(k) + \exp\left(-\pi j \frac{k}{N}\right) X_N^i(k)$$

$$X_{2N}(k+N) = X_N^o(k) - \exp\left(-\pi j \frac{k}{N}\right) X_N^i(k)$$

3.2 Algorithms 2: Radix-4 FFT Algorithm

3.2.1 Description

The first idea is decimation in time. It means to divide the sequence x(n) in the time domain. We know that Radix-2 algorithm divides x(n) into two groups of parity according to the value of n. The purpose is to transform the DFT of computing N points into the DFT of computing 2 points at N/2 point. And then divide into the DFT of computing 4 points at N/4 points. Such operation will be repeated until the sequence is decomposed into the operation of two-point DFT, which is simple addition and subtraction. [8]

Radix-4 FFT algorithm is divide the sequence which length $N = 4^l$ into 4 sequence. As we have explained above, we can transform the DFT of computing N points into the DFT of computing 4 points at N/4 point. Then we transform the DFT of computing N/4 points into the DFT of computing 4 points at N/16 point. In this way, finally, we get the operation of two-point DFT.

3.2.2 Analysis

$$\begin{aligned}
X(k) \quad &= \quad \sum_{n=0}^{N-1} x(n) e^{-\left(i\frac{2\pi nk}{N}\right)} \\
&= \sum_{n=0}^{\frac{N}{4}-1} x(4n) e^{-\left(i\frac{2\pi \times (4n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) e^{-\left(i\frac{2\pi(4n+1)k}{N}\right)} \\
&+ \sum_{n=0}^{\frac{N}{4}-1} x(4n+2) e^{-\left(i\frac{2\pi(4n+2)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3) e^{-\left(i\frac{2\pi(4n+3)k}{N}\right)} \\
&= \mathrm{DFT}_{\frac{N}{4}}[x(4n)] + W_N^k \mathrm{DFT}_{\frac{N}{4}}[x(4n+1)] + W_N^{2k}\,\mathrm{DFT}_{\frac{N}{4}}[x(4n+2)] + W_N^{3k}\,\mathrm{DFT}_{\frac{N}{4}}[x(4n+3)]
\end{aligned} \tag{20}$$

The radix-4 butterfly equations are as follows:

$$\begin{aligned}
X(r) \quad &= \quad X_0(r) + W_N^r X_1(r) + W_N^{2r} X_2(r) + W_N^{3r} X_3(r) \\
X(r + N/4) \quad &= \quad X_0(r) + W_N^{r+N/4} X_1(r) + W_N^{2(r+N/4)} X_2(r) + W_N^{3(r+N/4)} X_3(r) \\
X(r + N/2) \quad &= \quad X_0(r) + W_N^{r+N/2} X_1(r) + W_N^{2(r+N/2)} X_2(r) + W_N^{3(r+N/2)} X_3(r) \\
X(r + 3N/4) \quad &= \quad X_0(r) + W_N^{r+3N/4} X_1(r) + W_N^{2(r+3N/4)} X_2(r) + W_N^{3(r+3N/4)} X_3(r)
\end{aligned} \tag{21}$$

$$where\ 0 <= r <= N/4 - 1,\ r \in \mathbb{Z}$$

We could simplify Eq. 21 to:

$$\begin{aligned} X(r) &= X_0(r) + W_N^r X_1(r) + W_N^{2r} X_2(r) + W_N^{3r} X_3(r) \\ X(r+N/4) &= X_0(r) - jW_N^r X_1(r) - W_N^{2(r)} X_2(r) + jW_N^{3(r)} X_3(r) \\ X(r+N/2) &= X_0(r) - W_N^r X_1(r) + W_N^{2(r)} X_2(r) - W_N^{3(r)} X_3(r) \\ X(r+3N/4) &= X_0(r) + jW_N^r X_1(r) - W_N^{2r} X_2(r) - jW_N^{3r} X_3(r) \end{aligned} \tag{22}$$

From above, we can find that there is 12 complex addition in radix-4 operations.

If the FFT length N = 4^M, DFT can continue recursively decomposed. To determine the total computational cost, $M = log_4(N) = \frac{log_2(N)}{2} stages$ each stage need $\frac{N}{4}$ butterflies.

$$complex\ multiplies = 3\frac{N}{4}\frac{log_2(N)}{2} = \frac{3}{8}Nlog_2(N) \tag{23}$$

$$complex\ add = 8\frac{N}{4}\frac{log_2(N)}{2} = Nlog_2(N) \tag{24}$$

4 Research Plan and Method

- **Understand the algorithms of FFT**: Many algorithms have been proposed for FFT. We will do a literature review to gain a global knowledge of FFT algorithms. Specifically, we will study the Cooley–Tukey algorithm and Radix-4 algorithm for FFT.
- **Study the time complexity of the algorithms**: We will use theoretical analysis to study the time complexity of both algorithms.
- **Create software for the two algorithms and compare the theoretical and experimental results of the two algorithms**: We will use Matlab or Python to implement the two algorithms for FFT. Besides, we will compare the performance of the two algorithms. Finally, we will also compare the theoretical and experimental results of the two algorithms.

References

[1] Jean Armstrong. Ofdm for optical communications. *Journal of Lightwave Technology*, 27(3):189–204, 2009.

[2] Harri Holma and Antti Toskala. *LTE for UMTS: OFDMA and SC-FDMA Based Radio Access*. John Wiley & Sons, April 2009.

[3] Innovations. LTE in a nutshell. *White paper*.

[4] Boris Bellalta. IEEE 802.11ax: High-efficiency WLANS. *IEEE Wirel. Commun.*, 23(1):38–46, February 2016.

[5] Steven W Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, USA, 1997.

[6] Wikipedia contributors. Discrete cosine transform. `https://en.wikipedia.org/w/index.php?title=Discrete_cosine_transform&oldid=1100242965`, July 2022.

[7] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[8] Douglas L Jones. Radix-4 FFT algorithms - CiteSeerX. September 2006.

[9] C M Rader. Discrete fourier transforms when the number of data samples is prime. *Proc. IEEE*, 56(6):1107–1108, June 1968.

[10] Winograd. On computing the discrete fourier transform. *Math. Comput.*

[11] Thomas J. Watson IBM Research Center and Don Coppersmith. *An Approximate Fourier Transform Useful in Quantum Factoring*. IBM Thomas J. Watson Research Division, 1994.

[12] Duhamel and Hollmann. 'split radix'FFT algorithm. *Electron. Lett.*

[13] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008.

[14] Wikipedia contributors. Cooley–Tukey FFT algorithm. `https://en.wikipedia.org/w/index.php?title=Cooley%E2%80%93Tukey_FFT_algorithm&oldid=1111684678`, September 2022.