Group 17: Project Final Report

Mei Qi Tang
McGill University
mei.q.tang@mail.mcgill.ca

***Abstract*—Fourier Transforms have large importance in a wide area of scientific applications, but the naive algorithm is highly inefficient. This project first explores the implementation of the Discrete Fourier Transform (DFT) as the naive approach, and then explores two different implementations of the Cooley-Tuckey's Fast Fourier Transform (FFT), out-of-place and in-place FFT, as optimization solutions. After comparing the performance of the three algorithms, it is shown that the in-place FFT is the most efficient. Tests were setup to benchmark the time and space complexities of these algorithms. A general overview of Fourier Transforms, the three algorithms, and their performance analysis will be presented.**

***Index Terms*—Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), Discrete Signal Processing, ECSE 444 - Microprocessors**

I. Description

Fourier Transforms are used in nearly every field of engineering and physical sciences that involves signal processing. In the digital world, signals are usually represented as discrete signals, were each data point is a sample in time of a continuous signal, possibly produced in the physical world. Fourier Transforms for discrete signals are called Discrete Fourier Transforms (DFT).

The DFT is defined by equation (1), where it returns a set of N frequency-domain outputs. Each output is denoted by X_k and is the result of a summation of the time-domain input x_n multiplied by an eigenfunction term $e^{-i2\pi kn/N}$.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N} \qquad (1)$$
$$k = 0, ..., N-1$$

Following this mathematical description of the DFT, we observe that each summation computes in the order of $\mathcal{O}(N)$, and that the transform needs N number of summation iterations to complete. Hence, the runtime of a naively implemented DFT will be in the order of $\mathcal{O}(N^2)$.

A $\mathcal{O}(N^2)$ time complexity for such a widely used computation is problematic, which is leads us to exploring a more optimal algorithm that is designed to compute the DFT in only $\mathcal{O}(N \log N)$: the Fast Fourier Transform (FFT). More specifically, both the out-of-place and in-place Cooley-Tukey FFT algorithms will be implemented.

The Cooley-Tukey FFT essentially re-expresses the DFT in terms of smaller DFTs, recursively. Specifically, the radix-2 decimation-in-time FFT divides a DFT of size N into two smaller DFTs of size $N/2$, at each recursion. The first divided DFT consists of even-indexed inputs ($x_{2m} = x_0, x_2, ..., x_{N-2}$) and the second divided DFT consists of odd-indexed inputs ($x_{2m+1} = x_1, x_3, ..., x_{N-1}$). Once these divided DFTs are each computed recursively according to the same division, they are then combined to produce the overall DFT of the whole input signal. The mathematical expression of the DFT in (1) can essentially be re-arranged into equation (2), where E_k is the DFT of even-indexed part of x_n and O_k is the DFT of odd-indexed part of x_n.

$$\begin{aligned} X_k &= \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{-j2\pi}{N/2}mk} + e^{\frac{-j2\pi}{N}k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{-j2\pi}{N/2}mk} \\ &= E_k + e^{\frac{-j2\pi}{N}k} O_k \end{aligned} \qquad (2)$$

Moreover, the DFT has some symmetry characteristics that can be described by equation (3):

$$X_{k+\frac{N}{2}} = E_k - e^{\frac{-j2\pi}{N}k} O_k \qquad (3)$$

The rearrangement from equation (2) reduces the total number of summations from N to $\frac{N}{2}$, and the symmetry from (3) further reduces the number of computations for E_k, O_k, and X_k to also be $\frac{N}{2}$. Hence, the total runtime became $\mathcal{O}(\frac{N}{2} \times \frac{N}{2}) = \mathcal{O}((\frac{N}{2})^2)$. Lastly, applying this reduction recursively by the divide-and-conquer method, the FFT can achieve an overall time complexity of $\mathcal{O}(N \log N)$.

II. Implementation

All three algorithms used the *cexp* function from the *complex.h* library to compute the exponential of a complex number.

A. Discrete Fourier Transform

This first algorithm is the naive DFT, serving as a baseline for performance comparison. The DFT was implemented according to equation (1), using a nested double for-loop, each iterating N times. This algorithm is an out-of-place algorithm that returns the computed signal while keeping the input unmodified.

B. Out-of-Place Fast Fourier Transform

The first implementation of the Cooley-Tukey FFT is a recursive, out-of-place algorithm that returns the computed signal while keeping the input unmodified. This algorithm follows the equation (2) and uses dynamic memory allocation on the heap with *calloc* to return new array of the computed data for each recursive step.

C. In-Place Fast Fourier Transform

The second implementation of the FFT is also recursive, but is an in-place algorithm that computes and directly modifies the input signal to serve as the output signal. This algorithm was also implemented according to equation (2) but uses static memory allocation on the stack for the temporary buffer array used to move data around, while still computing in-place.

III. Performance

A. Testing

Each algorithm needed to be accurately tested before measuring their performances. The testing plan was as follows:

1) Choose test parameters (number of samples, waveform frequency, etc.) and generate discrete input signal.
2) Save input signal to *csv* file.
3) Run FFT algorithm on input signal.
4) Save output signal to *csv* file.
5) With the same parameters, generate discrete signal in Python and run Python's FFT function on the signal.
6) Read *csv* file in Python and plot both time and frequency domain signals. Verify if the plots are identical.

Figure 1 show an example of a discrete input signal and its transformed frequency signal being plotted in Python.

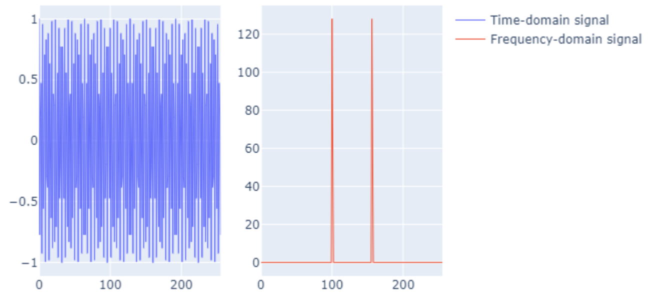

Fig. 1. Discrete Signals Before and After the FFT

B. Time Complexity

After running 100 iterations of each algorithm with (N) samples as powers of 2 from 64 to 16384, the average computation time for each algorithm as a function of N are recorded in Table I.

TABLE I
Runtime Performance Comparison

# samples (N)	DFT (ms)	out-of-place FFT (ms)	in-place FFT (ms)
64	0.7	0.16	0.04
128	2.7	0.32	0.1
256	10.32	0.56	0.2
512	41.62	1.16	0.46
1024	170.68	2.6	0.94
2048	686.36	5.44	2.26
4096	2712.08	11.9	4.4
8192	10871.12	26.34	9.7
16384	48543.22	55.32	23.02

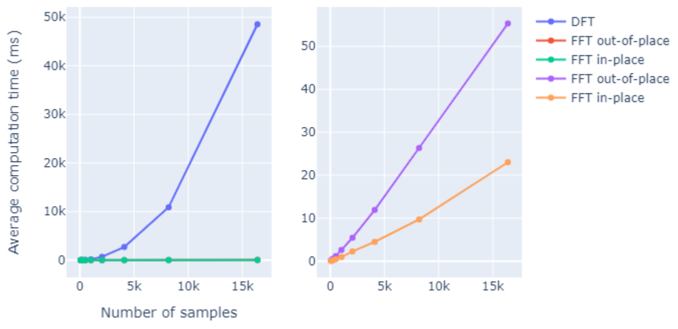

Fig. 2. Runtime Performance Comparison Between DFT and FFT (left), and Between out-of-place FFT and in-place FFT (right).

It is very apparent that the DFT is $\mathcal{O}(N^2)$ as the number of samples increases, with the highest computation time as 48 seconds per iteration, while the out-of-place FFT took at most 55 ms and the in-place FFT at most 23.02 ms.

These computation times are also plotted in Figure 2 to better visualize the comparison. While the left subplot of figure 2 shows the $\mathcal{O}(N^2)$ of the DFT in contrast to the $\mathcal{O}(N \log N)$ of FFT, the right subplot shows that the in-place FFT has the best runtime performance overall.

C. Space Complexity

Since the DFT is not recursive, it only occupies $\mathcal{O}(N)$ of space for its input and output arrays. In comparison, the recursive nature of FFTs will either require memory to store context for each recursion call. Hence, memory usage is $\mathcal{O}(N \log N)$.

Also, since the in-place FFT is computing without dynamically allocating heap memory, unlike the out-of-place FFT, it has a better memory complexity and also saves on computation time by only accessing the stack, which is allocated at compile time.

Furthermore, all three algorithms are using *double complex* data types for the signals to maintain high precision and avoid numerical computational inaccuracies as much as possible. If it is acceptable to reduce numerical precision by half, we can further decrease the size of datatype to *float complex* and reduce memory usage.

IV. Conclusion

In conclusion, three different algorithms computing the Discrete Fourier Transform were compared in terms of their runtime and memory usage performance. Both of the Fast Fourier Transforms were shown to have drastically reduced computation time with $\mathcal{O}(N \log N)$ compared to the native DFT algorithm of $\mathcal{O}(N^2)$. The in-place FFT is ultimately not only the most effective algorithm in terms of computation time, with an average of only 23 ms for computing 16k samples, but it is also efficient in memory usage, computing the DFT in-place without dynamically allocating memory at each recursive step.