ECE 310 Fall 2023

Lecture 26

Fast Fourier transform

Corey Snyder

Learning Objectives

After this lecture, you should be able to:

- Explain the computational complexity of directly computing the DFT according to the summation or matrix formulation vs. the complexity of the fast Fourier transform.
- Apply the merging equations of the decimation-in-time fast Fourier transform to compute the DFT a short sequence.
- Draw decimation-in-time butterfly diagrams for short sequences.

Recap from previous lecture

We now turn our attention to the fast Fourier transform algorithm, which is a computationally efficient implementation for computing the DFT of a discrete-time signal. The computational efficiency of the FFT will enable convolution via the DFT to be quite efficient. We will derive one version of the fast Fourier transform and compare its computational complexity against simpler implementations of the DFT.

1 Directly computing the DFT

We begin by considering the computational complexity of calculating the DFT using the DFT sum. For signal $x[n]$, we compute the DFT $X[k]$ as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi k}{N}n}, \; 0 \leq k \leq N-1. \tag{1}$$

For a single value of k, we perform the above length-N summation, thus we incur N *multiply-add* operations. Here, we take the operation of multiplying two numbers and adding to a result – one "multiply-add" – to be our fundamental unit of cost in calculating the DFT. The multiplication is $x[n]e^{-j\frac{2\pi k}{N}n}$ while the addition is adding to the sum for the current value of k. Furthermore, we must perform this length-N summation for N values of k, i.e. $0 \leq k \leq N-1$. Therefore, we like to say that the DFT summation takes $\mathcal{O}(N \cdot N) = \mathcal{O}(N^2)$ operations. The notation for $\mathcal{O}$ is referred to as *Big O notation* and is another way of saying "on the order of".

A more detailed discussion of Big O notation is best left to CS 173 or CS/ECE 374, but for this class we will use the notation to refer to the limiting behavior of some function. In this case, the limiting behavior of the computational cost of calculating the DFT. This limit is with respect to N and as $N \to \infty$. When using Big O notation, we keep the terms that are strictly greater than any other term and omit any scalars. Consider the following couple examples:

$$8N^3 + 20N^2 + N = \mathcal{O}(N^3), \tag{2}$$

$$N \log N + N = \mathcal{O}(N \log N). \tag{3}$$

Line 2 omits the scalar on each term and only includes the N^3 since N^3 dominates (is consistently greater than) N^2 and N as $N \to \infty$. Similarly, in line 3, we only keep the $N \log N$ term since $N \log N$ dominates N.

The $\mathcal{O}(N^2)$ cost of the DFT sum also holds for the matrix-vector multiplication method we introduced in lecture 21. This method will still likely be faster for the same software and hardware reasons we cited previously. This means each multiply-add operation in the $\mathcal{O}(N^2)$ cost is faster for matrix-vector multiplication than the direct summation implementation. Either way, we would like to improve on the $\mathcal{O}(N^2)$ cost and this is where the fast Fourier transform comes into play.

2 Fast Fourier transform

The *fast Fourier transform* (FFT) refers to a class of algorithms that compute the DFT with better computational complexity than the direct methods we discussed in the previous section. These FFT algorithms typically obtain $\mathcal{O}(N \log N)$ cost complexity, which presents a drastic improvement over the previous $\mathcal{O}(N^2)$ complexity. In this course, we will cover one such algorithm known as the *decimation-in-time* algorithm, though many others exist. The log we use to denote the computational complexity of the FFT is the base-2 log, i.e. $\log_2$, though we omit the subscript-2 for simplicity.

2.1 Decimation-in-time FFT

The key idea behind the decimation-in-time (DIT) algorithm, and many other FFTs like it, is to recursively split the discrete-time signal $x[n]$ into small pieces, quickly compute the DFT of shorter sequences, then efficiently merge these shorter results. Before beginning our derivation of the DIT FFT algorithm, we define the *twiddle factor* W:

$$W = e^{-j2\pi} \tag{4}$$

$$W_N^{kn} = e^{-j\frac{2\pi kn}{N}}. \tag{5}$$

We start by splitting the DFT sum for $X[k]$ into summations over the even and odd indices of $x[n]$. Without loss of generality, we assume N is divisble by 2.

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi k}{N}n}, \ 0 \le k \le N-1 \\ &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \qquad (6) \\ &= \sum_{m=0}^{\frac{N}{2}-1} x[2m] W_N^{k(2m)} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] W_N^{k(2m+1)} \qquad (7) \end{aligned}$$

Now, let $a[n] = x[2n]$ and $b[n] = x[2n+1]$ denote the even and odd-index samples of $x[n]$, respectively. We can then further simplify line 7 as follows:

$$\begin{aligned} X[k] &= \sum_{n=0}^{\frac{N}{2}-1} a[n] W_N^{k(2n)} + \sum_{n=0}^{\frac{N}{2}-1} b[n] W_N^{k(2n+1)}, \ 0 \le k \le N-1 \qquad (8) \\ &= \sum_{n=0}^{\frac{N}{2}-1} a[n] W_{\frac{N}{2}}^{kn} + \sum_{n=0}^{\frac{N}{2}-1} b[n] W_{\frac{N}{2}}^{kn} W_N^k \qquad (9) \\ &= A[k] + W_N^k B[k], \ 0 \le k \le N-1. \qquad (10) \end{aligned}$$

We make a substitution in line 9 for the twiddle factor via

$$W_N^{2kn} = e^{-j(2)\frac{2\pi kn}{N}} = e^{-j\frac{2\pi kn}{\frac{N}{2}}} = W_{\frac{N}{2}}^{kn}. \tag{11}$$

This substitution allows us to arrive at the nicely simplified form in line 10. Here, $A[k]$ and $B[k]$ are the length-$\frac{N}{2}$ DFTs of $a[n]$ and $b[n]$, respectively:

$$A[k] = \sum_{n=0}^{\frac{N}{2}-1} a[n]W_{\frac{N}{2}}^{kn},\ 0 \leq k \leq \frac{N}{2} - 1 \tag{12}$$

$$B[k] = \sum_{n=0}^{\frac{N}{2}-1} b[n]W_{\frac{N}{2}}^{kn},\ 0 \leq k \leq \frac{N}{2} - 1. \tag{13}$$

This suggests that Eqn. 10 can only actually be evaluated for $0 \leq k \leq \frac{N}{2} - 1$. However, since $A[k]$ and $B[k]$ are each of length $\frac{N}{2}$, we know they must be $\frac{N}{2}$-periodic.

$$A\left[k + \frac{N}{2}\right] = A[k] \tag{14}$$

$$B\left[k + \frac{N}{2}\right] = B[k] \tag{15}$$

This is the same as when we say an ordinary length-N DFT is N-periodic. Thus, we can write the DFT $X[k]$ in terms of $A[k]$ and $B[k]$ as follows:

$$X[k] = A[k] + W_N^k B[k],\ 0 \leq k \leq \frac{N}{2} - 1 \tag{16}$$

$$X\left[k + \frac{N}{2}\right] = A[k] - W_N^k B[k],\ 0 \leq k \leq \frac{N}{2} - 1. \tag{17}$$

Note that we have $-W_N^k$ attached to $B[k]$ in Eqn. 17 since we are evaluating $W_N^{k+N/2} \equiv e^{-j\frac{2\pi k}{N}} e^{-j\pi} = -W_N^k$. Equations 16 and 17 are the important *merging formulas* for the decimation-in-time FFT. These merging formulas apply to *any* even length signal $x[n]$ and allow us to compute a length-N DFT by merging two length-$\frac{N}{2}$ DFTs. Thus, we can recurse this procedure until the divided signal is just length-2. In programming terms, this is our base case. At length-2, the merging formulas are easy to see. Let $x[n] = \{x[0], x[1]\}$:

$$a[n] = x[0],\ 0 \leq n \leq 0 \tag{18}$$

$$A[k] = x[0],\ 0 \leq k \leq 0 \tag{19}$$

$$b[n] = x[1],\ 0 \leq n \leq 0 \tag{20}$$

$$B[k] = x[1],\ 0 \leq k \leq 0. \tag{21}$$

The limits on n and k may look odd, but we want to emphasize these are length-1 sequences in the time and DFT-domains. The merging equations for a length-2 DFT will then combine these trivial 1-point DFTs

$$X[0] = A[0] + W_2^0 B[0] \tag{22}$$

$$= x[0] + x[1] \tag{23}$$

$$X[1] = A[0] - W_2^0 B[0] \tag{24}$$

$$= x[0] - x[1]. \tag{25}$$

Altogether, we can apply the merging formulas in multiple stages to compute the DFT of longer sequences and this will provide the computational savings of the FFT. Let's look at a length-8 FFT to see this fully.

Exercise 1: Let $x[n]$ be a length-8 signal. We would like to use the decimation-in-time merging formulas to compute the DFT.

We start by dividing (or decimating) $x[n]$ until we reach length-2 signals.

$$a[n] = x[2n] = \{x[0], x[2], x[4], x[6]\} \tag{26}$$
$$b[n] = x[2n+1] = \{x[1], x[3], x[5], x[7]\} \tag{27}$$
$$c[n] = a[2n] = \{x[0], x[4]\} \tag{28}$$
$$d[n] = a[2n+1] = \{x[2], x[6]\} \tag{29}$$
$$e[n] = b[2n] = \{x[1], x[5]\} \tag{30}$$
$$f[n] = b[2n+1] = \{x[3], x[7]\} \tag{31}$$

Following our previous length-2 DFT example, we then have

$$C[k] = x[0] + x[4],\ C[k+1] = x[0] - x[4],\ 0 \leq k \leq 0 \tag{32}$$
$$D[k] = x[2] + x[6],\ D[k+1] = x[2] - x[6],\ 0 \leq k \leq 0 \tag{33}$$
$$E[k] = x[1] + x[5],\ E[k+1] = x[1] - x[5],\ 0 \leq k \leq 0 \tag{34}$$
$$F[k] = x[3] + x[7],\ F[k+1] = x[3] - x[7],\ 0 \leq k \leq 0 \tag{35}$$

Next, we combine these length-2 DFTs to form the length-4 DFTs $A[k]$ and $B[k]$:

$$A[k] = C[k] + W_4^k D[k],\ A[k+2] = C[k] - W_4^k D[k],\ 0 \leq k \leq 1 \tag{36}$$
$$B[k] = E[k] + W_4^k F[k],\ A[k+2] = E[k] - W_4^k F[k],\ 0 \leq k \leq 1 \tag{37}$$

Finally, we merge the length-4 $A[k]$ and $B[k]$ to obtain the final DFT $X[k]$:

$$X[k] = A[k] + W_8^k B[k],\ 0 \leq k \leq 3 \tag{38}$$
$$X[k+4] = A[k] - W_8^k B[k],\ 0 \leq k \leq 3. \tag{39}$$

2.2 Butterfly diagrams

We conclude this lecture with a popular pictorial representation of decimiation-in-time FFTs known as a *butterfly diagram*. These diagrams visualize each step in the merging equations as we combine shorter DFTs into larger DFTs until we obtain the final result. Another benefit of butterfly diagrams is that they can help make the computational complexity of the FFT clearer to see. The name "butterfly diagram" comes from the symmetrical triangles we see in the diagram that (kind of) resemble a butterfly.

Figure 1 shows a single butterfly diagram to represent the merging formulas in Eqns. 16 and 17. We see that a butterfly provides the DFT result for a given index k and its symmetrical counterpart at $k + \frac{N}{2}$. The triangles in the diagram indicate locations where multiplication may occur. If there is no label next to a triangle, we assume this is simply multiplication by one. Furthermore, the dots where two arrows meet represent the addition of those two elements.

We can then use butterfly diagrams to represent the FFT we computed in Exercise 1. Figure 2 depicts this length-8 FFT. We observe the three merging stages moving left-to-right that merge length-1, length-2, and length-4 DFTs, respectively. Note that at each stage, we have $\frac{N}{2} = 4$ butterflies. Each single butterfly diagram requires two multiply-add operations. Thus, each stage in total requires N multiply-add operations. Finally, each time we merge, we double the length of the signals we are combining. Therefore, we must merge $\log_2 N$ times to go from length-1 DFTs to computing our final length-N DFT. Altogether, this shows us that the FFT has $\mathcal{O}(N \log N)$ computational complexity. There are certainly other constant-factor savings we can find. For example every W_N^0 scaling coefficient is simply multiplication by 1 and thus does not require multiplication. We instead want to focus on the main takeaway that FFT algorithms like decimation-in-time have $\mathcal{O}(N \log N)$ complexity that greatly improves on a direct $\mathcal{O}(N^2)$ implementation.

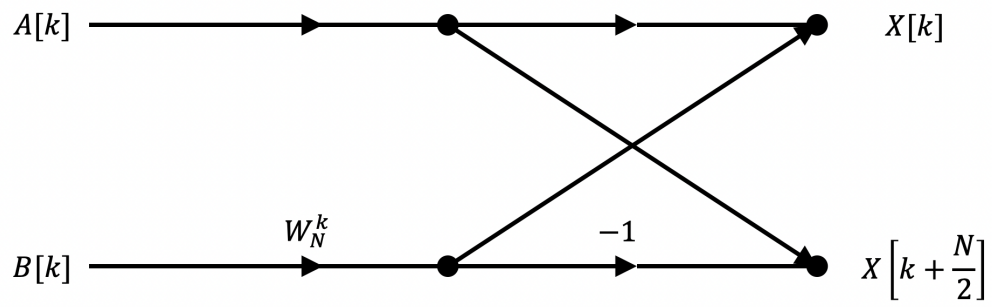

Figure 1: Single butterfly diagram.

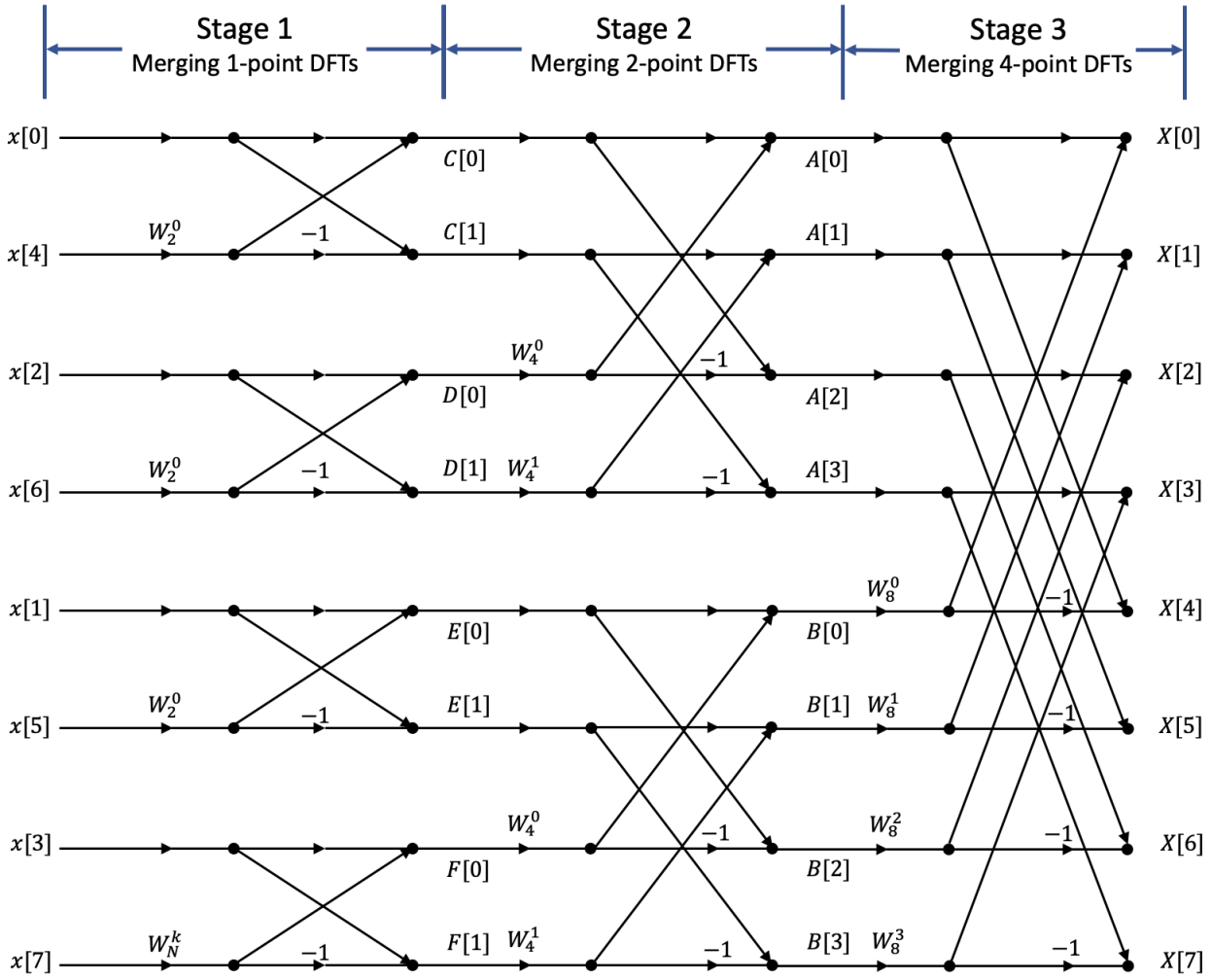

Figure 2: Butterfly diagram for length-8 FFT of Exercise 1.