Introduction to the Fast-Fourier Transform (FFT) Algorithm

C.S. Ramalingam

Department of Electrical Engineering
IIT Madras

The Discrete Fourier Transform (DFT)

- DFT of an N-point sequence x_n, $n = 0, 1, 2, \ldots, N-1$ is defined as

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi k}{N}n} \qquad k = 0, 1, 2, \cdots, N-1$$

- An N-point sequence yields an N-point transform
- X_k can be expressed as an *inner product*:

$$X_k = \begin{bmatrix} 1 & e^{-j\frac{2\pi k}{N}} & e^{-j\frac{2\pi k}{N}2} & \ldots & e^{-j\frac{2\pi k}{N}(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

The Discrete Fourier Transform (DFT)

- Notation: $W_N = e^{-j\frac{2\pi}{N}}$. Hence,

$$X_k = \begin{bmatrix} 1 & W_N^k & W_N^{2k} & \ldots & W_N^{(N-1)k} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

- By varying k from 0 to $N-1$ and combining the N inner products, we get the following:

$$\mathbf{X} = \mathbf{W}\mathbf{x}$$

- $\mathbf{W}$ is an $N \times N$ matrix, called as the "DFT Matrix"

The DFT Matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ & & & \vdots & \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}_{N \times N}$$

- The notation $\mathbf{W}_N$ is used if we want to make the size of the DFT matrix explicit

How Many Complex Multiplications Are Required?

- Each inner product requires N complex multiplications
  - There are N inner products
- Hence we require N^2 multiplications
- However, the first row and first column are all 1s, and *should not be counted as multiplications*
  - There are $2N - 1$ such instances
- Hence, the number of complex multiplications is $N^2 - 2N + 1$, i.e., $(N - 1)^2$

How Many Complex Additions Are Required?

- Each inner product requires $N - 1$ complex additions
  - There are N inner products
- Hence we require $N(N - 1)$ complex additions

Total Operation Count

- No. of complex multiplications: $(N-1)^2$
- No. of complex additions: $N(N-1)$
- The operation count for multiplications and additions assumes that W_N^k has been computed offline and is available in memory
  - If pre-computed values of W_N^k are not available, then the operation count will increase
- We will assume that all the required W_N^k have been pre-computed and are available

Operation Count Makes DFT Impractical

- For large N,

  $(N-1)^2 \approx N^2$

  $N(N-1) \approx N^2$

- Hence both multiplications and additions are $O(N^2)$
- If $N = 10^3$, then $O(N^2) = 10^6$, i.e., a million!
- This makes the straightforward method slow and impractical even for a moderately long sequence

The Divide and Conquer Approach

- Suppose N is even and we split the sequence into two halves.
  - Each sequence has N/2 points
- Suppose we compute the $\dfrac{N}{2}$ point DFT of each sequence
  - Multiplications : $2 \times \left(\frac{N}{2}\right)^2 = \frac{N^2}{2}$
- Suppose we are able to combine the individual DFT results to get the originally required DFT
  - Some computational overhead will be consumed to combine the two results
- If $\frac{N^2}{2}$ + overhead $< N^2$, then this approach will reduce the operation count

The Divide and Conquer Approach

Let $N = 8$

- Straightforward implementation requires, *approximately*, 64 multiplications
- The "divide and conquer" approach requires, *approximately*, $2 \times \left(\frac{8}{2}\right)^2 +$ overhead, i.e., $32 +$ overhead multiplications
- Questions:
  - Can the two DFTs be combined to get the original DFT ?
  - If so, how ? What is the overhead involved ?
  - Will $32 +$ overhead be less than 64 ?

The Decimation in Time (DIT) Algorithm

- From $\{x_n\}$ form two sequences as follows:

$$\{g_n\} = \{x_{2n}\} \qquad \{h_n\} = \{x_{2n+1}\}$$

- $\{g_n\}$ contains the even-indexed samples, while $\{h_n\}$ contains the odd-indexed samples
- The DFT of $\{x_n\}$ is

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \, W_N^{nk} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x_{2r} \, W_N^{(2r)k} + \sum_{r=0}^{\frac{N}{2}-1} x_{2r+1} \, W_N^{(2r+1)k} \\ &= \sum_{r=0}^{\frac{N}{2}-1} g_r \, W_N^{(2r)k} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} h_r \, W_N^{(2r)k} \end{aligned}$$

The Decimation in Time (DIT) Algorithm

- But,

$$W_N^{2rk} = e^{-j\frac{2\pi}{N}(2rk)} = e^{-j\frac{2\pi}{N/2}(rk)} = W_{N/2}^{rk}$$

and hence

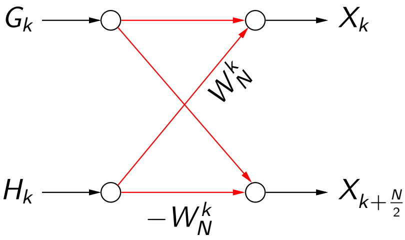
$$\begin{aligned} X_k &= \sum_{r=0}^{\frac{N}{2}-1} g_r \, W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} h_r \, W_{N/2}^{rk} \\ &= G_k + W_N^k H_k \qquad k = 0, 1, \ldots, N-1 \end{aligned}$$

- $\{G_k\}$ and $\{H_k\}$ are $\frac{N}{2}$ point DFTs
- The overhead for combining the two $\frac{N}{2}$ point DFTs is the multiplicative factor W_N^k for $k = 0, 1, \ldots, N-1$
  - W_N^k is called "twiddle factor"

The Decimation in Time (DIT) Algorithm

- The $N/2$ point DFTs $\{G_k\}$ and $\{H_k\}$ are periodic with period $N/2$
  - $$G_{k+\frac{N}{2}} = G_k$$
    $$H_{k+\frac{N}{2}} = H_k$$
- $W_N^{k+\frac{N}{2}} = -W_N^k$
- Hence, if $X_k = G_k + W_N^k H_k$, then $X_{k+\frac{N}{2}} = G_k - W_N^k H_k$
  - $W_N^k H_k$ needs to be computed only once for $k = 0$ to $\frac{N}{2} - 1$
- Thus, the multiplication overhead due to the twiddle factors is only $\frac{N}{2}$

Butterfly Diagram



- $X_k = G_k + W_N^k H_k$
- $X_{k+\frac{N}{2}} = G_{k+\frac{N}{2}} + W_N^{k+\frac{N}{2}} H_{k+\frac{N}{2}}$
  $= G_k - W_N^k H_k$

The Decimation in Time (DIT) Algorithm

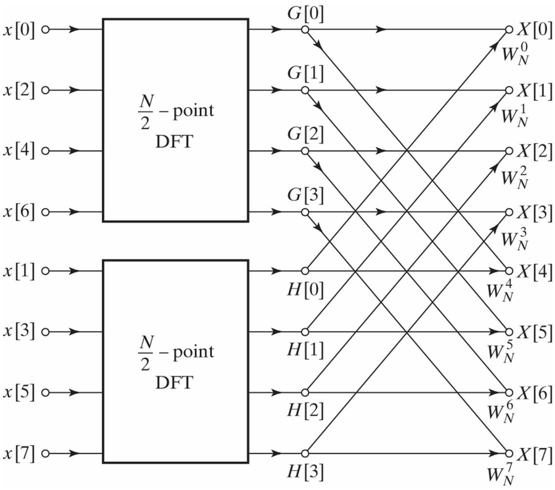


Figure 9.4 Flowgraph of Decimation in Time algorithm for $N = 8$ (Oppenheim and Schafer, *Discrete-Time Signal Processing*, 3rd edition, Pearson Education, 2010, p. 726)

“Divide and Conquer” Results in Savings!

- For $N = 8$, the straightforward approach requires, *approximately*, 64 multiplications
- The “Divide and Conquer” approach, after the first stage, requires $32 + 4 =$ 36 multiplications
- Thus, this approach clearly reduces the number of additions and multiplications required

Reusing the "Divide and Conquer" Strategy

- The same idea can be applied for calculating the $\frac{N}{2}$ point DFT of the sequences $\{g_r\}$ and $\{h_r\}$
  - Computational savings can be obtained by dividing $\{g_r\}$ and $\{h_r\}$ into *their* odd- and even-indexed halves
- This idea can be applied recursively $\log_2 N$ times if N is a power of 2
  - Such algorithms are called radix 2 algorithms
- If $N = 2^\gamma$, then the final stage sequences are all of length 2
- For a 2-point sequence $\{p_0, p_1\}$, the DFT coefficients are

$$P_0 = p_0 + p_1 \qquad P_1 = p_0 - p_1$$

DIT Flowgraph for $N = 8$

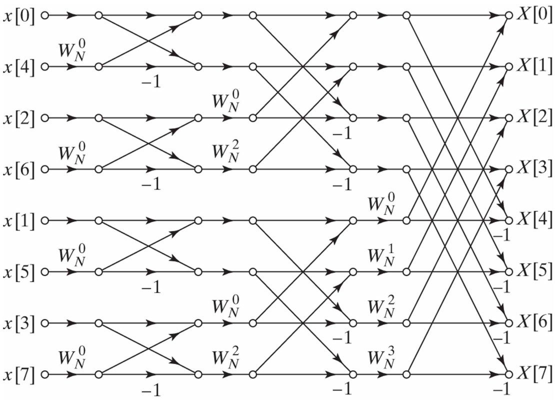


Figure 9.11 Flowgraph of Decimation in Time algorithm for $N = 8$ (Oppenheim and Schafer, *Discrete-Time Signal Processing*, 3rd edition, Pearson Education, 2010, p. 730)

Overall Operation Count

- The direct method requires N^2 multiplications
- After the first split, $N^2 \longrightarrow 2\left(\frac{N}{2}\right)^2 + \frac{N}{2}$
  - $\frac{N}{2}$ is due to the *twiddle factors*
- After the second split, $\left(\frac{N}{2}\right)^2 \longrightarrow 2\left(\frac{N}{4}\right)^2 + \frac{N}{4}$

  Hence,

  $$N^2 \longrightarrow 2\left(\frac{N}{2}\right)^2 + \underbrace{\frac{N}{2}}_{\text{first stage}} \longrightarrow 4\left(\frac{N}{4}\right)^2 + \underbrace{\frac{N}{2} + \frac{N}{2}}_{\text{second stage}}$$

- Generalizing, if there are $\log_2 N$ stages, the number of multiplications needed will be, *approximately*, $\frac{N}{2}\log_2 N$

Overall Operation Count

- If $W_N^{k+\frac{N}{2}} = -W_N^k$ is not considered, the overhead count will be N and not $\frac{N}{2}$
- In this case,

$$N^2 \longrightarrow 2\left(\frac{N}{2}\right)^2 + \underbrace{N}_{\text{first stage}} \longrightarrow 4\left(\frac{N}{4}\right)^2 + \underbrace{N + N}_{\text{second stage}}$$

- Hence the overall multiplication count will be $N \log_2 N$
- For $N = 1024$

$$N^2 = 1,048,576 \qquad N \log_2 N = 10,240$$

Savings of two orders of magnitude!

- Recall that, for $N = 8$, the first split requires the data to be arranged as follows:

$$x_0, \quad x_2, \quad x_4, \quad x_6, \qquad x_1, \quad x_3, \quad x_5, \quad x_7$$

- In the second and final split, the data appear in the following order:

$$x_0, \quad x_4, \qquad x_2, \quad x_6, \qquad x_1, \quad x_5, \qquad x_3, \quad x_7$$

- The final order is said to be in "bit reversed" form:

Original	Binary Form	Reversed Form	Final
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

An Algorithm For Sequence Reversal

- Consider the card sequence *7, 8, 9, 10, J, Q, K, A*
- First, reverse pairwise:
  - *8, 7, 10, 9, Q, J, A, K*
- Then swap the adjacent pairs:
  - *10, 9, 8, 7, A, K, Q, J*
- Finally, swap the two groups of 4 (each group is half the original size):
  - *A, K, Q, J, 10, 9, 8, 7* Done!

How To Use It For Bit Reversal

- The first step of swapping of bits pairwise can be done with bitwise AND/OR and bit shift operators
- Pick out the even and odd bits by using masks
  - ABCDEFGH & 01010101 = 0B0D0F0H
    ABCDEFGH & 10101010 = A0C0E0G0
- Left shift the first result and right shift the second result
  - B0D0F0H0
  - 0A0C0E0G
- Bitwise OR the above results
  - B0D0F0H0 $\oplus$ 0A0C0E0G = BADCFEHG
- Pairwise bit swapping accomplished!

C Code For Bit Reversal

```
unsigned reverse_bits(unsigned input)
{
//works on 32-bit machine
input = (input & 0x55555555) <<  1 | (input & 0xAAAAAAAA) >> 1;
input = (input & 0x33333333) <<  2 | (input & 0xCCCCCCCC) >> 2;
input = (input & 0x0F0F0F0F) <<  4 | (input & 0xF0F0F0F0) >> 4;
input = (input & 0x00FF00FF) <<  8 | (input & 0xFF00FF00) >> 8;
input = (input & 0x0000FFFF) << 16 | (input & 0xFFFF0000) >> 16;

return input;
}
```

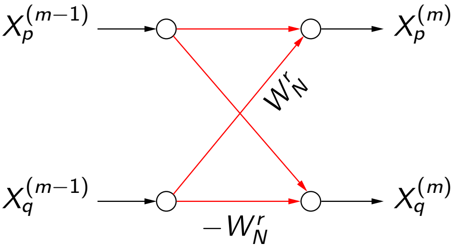
- Bit reversal for the entire array can take a large overhead if performed inefficiently
- There are several efficient algorithms for sorting an array in bit-reversed order
  - *Bit reversal on uniprocessors* by Alan H. Karp, SIAM Review, Vol. 38, March 1996, pp. 1–26
  - `http://www-graphics.stanford.edu/~seander/bithacks.html#BitReverseTable`

In-Place Computation

- Notation:
  - First stage: $X_k^{(0)} = x_k$
  - Last stage: $X_k^{(\log_2 N)} = X_k$
- For the m-th stage butterfly
  - Input: $X_p^{(m-1)}$, $X_q^{(m-1)}$
  - Output: $X_p^{(m)}$, $X_q^{(m)}$
- The corresponding equations are

$$
\begin{aligned}
X_p^{(m)} &= X_p^{(m-1)} + W_N^r \, X_q^{(m-1)} \\
X_q^{(m)} &= X_p^{(m-1)} - W_N^r \, X_q^{(m-1)}
\end{aligned}
$$

In-Place Computation

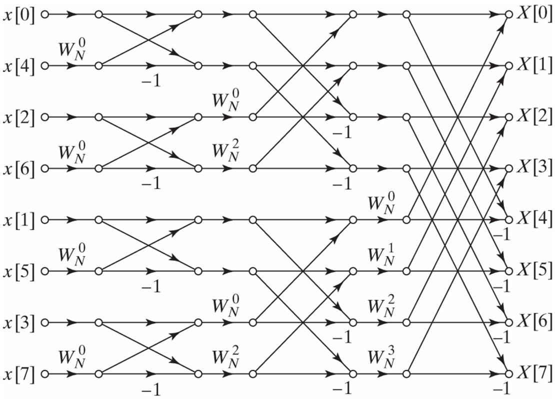


- $X_p^{(m-1)}$ and $X_q^{(m-1)}$ are needed for computing $X_p^{(m)}$ and $X_q^{(m)}$
- *They are not needed for any other pair*
- Hence

$$X_p^{(m)} \longleftarrow X_p^{(m-1)}$$
$$X_q^{(m)} \longleftarrow X_q^{(m-1)}$$

- This is called "in-place computation"

In-Place Computation



- x_0 and x_4 are not needed once that butterfly is computed
- Hence they can be overwritten with the results of the butterfly computation
  - Same is true for other pairs also

The Decimation in Frequency (DIF) Algorithm

- Another method of splitting the input sequence into half is as follows:

  $x_0, \; x_1, \; x_2, \; x_3, \qquad x_4, \; x_5, \; x_6, \; x_7$

- Similar savings are obtained in this case also
- The output X_k will now appear in bit reversed order
- This method is called as the Decimation in Frequency algorithm

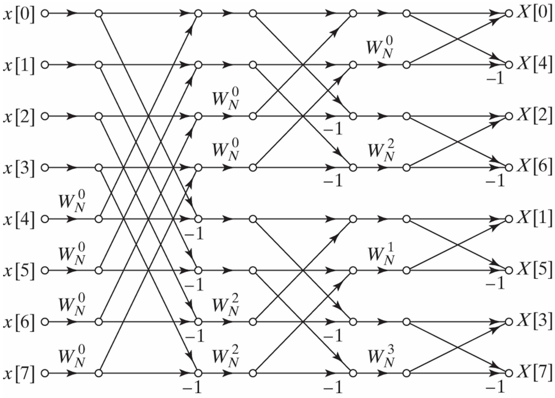


Figure 9.22 Flowgraph of Decimation in Frequency algorithm for $N = 8$ (Oppenheim and Schafer, *Discrete-Time Signal Processing*, 3rd edition, Pearson Education, 2010, p. 740)

Prime Factor Algorithms

- When N is not a power of 2 but is a composite number, it can be expressed in terms of its prime factors
  - Example: $N = 6 = 3 \times 2$
- We can now split the given sequence into 3 segments of 2 samples each
  - $x_0, \ x_3, \qquad x_1, \ x_4, \qquad x_2, \ x_5$
- Three 2-point DFTs are computed and combined to get the final DFT
- Significant computational savings is obtained, as before
- Efficient algorithms exist even when N is prime!
  - http://en.wikipedia.org/wiki/Rader's_FFT_algorithm