FAST FOURIER TRANSFORM

QUANTUM CRYPTOLOGY

Pushpendra Pal
Master of Cryptology 2022-2024
Indian Statistical Institute Kolkata
isi.pushp@gmail.com

November 03, 2023

ABSTRACT

The Discrete Fourier Transform (DFT) plays an important role in many applications of digital signal processing, including linear filtering, correlation analysis and spectrum analysis. This calls for a better and efficient algorithm. Fast Fourier Transform is one, which takes a divide and conquer approach where an N point DFT can be broken into no of smaller DFTs. Subsequently, the no of multiplications and additions required would be scaled down as well. DFT may as well look like a formulation which could only work for signal transformation but it can be used for any array multiplication. Specifically, any number in a specific radix can be written as a polynomial, and thus a DFT can be applied over that. Thereafter polynomials can be multiplied using fewer multiplications-additions than would be required in the naive case.

Keywords Discrete Fourier Transform · Fast Fourier Transform · Split radix-2 FFT · phase factor.

1 Introduction

Let's start with the computation of a 6 point DFT using the matrix multiplication.

$$DFT(a,b,c,d,e,f) = [a \quad b \quad c \quad d \quad e \quad f] \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & -1 & -w & -w^2 \\ 1 & w^2 & -w & 1 & w^2 & -w \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -w & w^2 & 1 & -w & w^2 \\ 1 & -w^2 & -w & -1 & w^2 & w \end{bmatrix}$$

here, w is the 6th root of unity,

Complexity

- # Multiplications = N^2; ($N = 6$ in the above example, so 36 multiplications needed)
- # Additions = $N(N-1)$

2 Towards Fast Fourier Transform

In the above example with N=6, we can see a possible improvement if somehow we could break the 6 point DFT to a 2 point DFT and a 3 point DFT. But how do we achieve? Let's

2.1 Properties of roots of unity:

- $w_N^{k+N/2} = -w_N^k$

- $w_N^{k+N} = w_N^k$

The above two properties will be used to reduce the no of computations.

2.2 Breaking N-DFT into two DFTs of size L and M

Now, if we have a N sized 1D array, and if N is multiple of at least two numbers, then we can transform the 1D array to a 2D array of size LxM, where N=LM. The indices would be changed according to the manner in which we store the 1-D array.

L = No of Rows
M = No of Columns

New indices for input side where (l,m) are indices of the 2-D representation:

- n = lM + m // for Row wise storage[L=3, M=2]
  $[x[0] \quad x[1] \quad x[2] \quad x[3] \quad x[4] \quad x[5]] \implies \begin{bmatrix} x[0] & x[1] \\ x[2] & x[3] \\ x[4] & x[5] \end{bmatrix} \implies \begin{bmatrix} x[0,0] & x[0,1] \\ x[1,0] & x[1,1] \\ x[2,0] & x[2,1] \end{bmatrix}$
- n = mL + l // for Column wise storage
  $[x[0] \quad x[1] \quad x[2] \quad x[3] \quad x[4] \quad x[5]] \implies \begin{bmatrix} x[0] & x[3] \\ x[1] & x[4] \\ x[2] & x[5] \end{bmatrix} \implies \begin{bmatrix} x[0,0] & x[0,1] \\ x[1,0] & x[1,1] \\ x[2,0] & x[2,1] \end{bmatrix}$

New indices for output side where (p,q) are indices of the 2-D representation:

- k = pM + q // for Row wise storage
  $\begin{bmatrix} X[0] & X[1] \\ X[2] & X[3] \\ X[4] & X[5] \end{bmatrix}$
- k = qL + p // for Column wise storage
  $\begin{bmatrix} X[0] & X[3] \\ X[1] & X[4] \\ X[2] & X[5] \end{bmatrix}$

Now let's manipulate the equation of DFT.

$$X(k) = \sum_{n=0}^{n=N-1} x(n) w_N^{kn} \tag{1}$$

w_N is the Nth root of unity.

$$X(k) = X(p,q) = \sum_{m=0}^{m=M-1} \sum_{l=0}^{l=L-1} x(l,m) w_N^{(Mp+q)(mL+l)} \tag{2}$$

Here we replaced k by Mp+q (row wise order) and n by mL+l (Column wise order). The reason for this is that we need to get LM after multiplication of kn in the w_N power. Then we could reduce LM by N. We could have used k (column wise order) and n (row wise order) indices as well. It would work just fine. But if we were to combine the column wise order for both k and n, then we would get L^2 or M^2, which is of no significant use for us at this point. Remind you that (p,q) are the output side indices and (l,m) are input side indices.

$$w_N{}^{(Mp+q)(mL+l)} = w_N{}^{MpLm} * w_N{}^{qmL} * w_N{}^{Mpl} * w_N{}^{ql} \tag{3}$$

$w_N{}^{MpLm} = w_N{}^{Npm} = 1$ // root of unity
$w_N{}^{qmL} = w_{N/L}{}^{mq} = w_M{}^{mq}$
$w_N{}^{Mpl} = w_{N/M}{}^{pl} = w_L{}^{pl}$

$$w_N{}^{(Mp+q)(mL+l)} = 1 * w_{N/L}{}^{mq} * w_{N/M}{}^{pl} * w_N{}^{ql} \tag{4}$$

$$w_N{}^{(Mp+q)(mL+l)} = 1 * w_M{}^{mq} * w_L{}^{pl} * w_N{}^{ql} \tag{5}$$

$$X(p,q) = \sum_{m=0}^{m=M-1} \sum_{l=0}^{l=L-1} x(l,m) w_M{}^{mq} * w_L{}^{pl} * w_N{}^{ql} \tag{6}$$

$$X(p,q) = \sum_{l=0}^{L-1} \left(\left(\sum_{m=0}^{M-1} x(l,m) w_M{}^{mq} \right) \cdot w_N{}^{ql} \right) \cdot w_L{}^{ql}$$

$$F(l,q) = \sum_{m=0}^{m=M-1} x(l,m) w_M{}^{mq} \tag{7}$$

$$G(l,q) = F(l,q) w_M{}^{mq} \tag{8}$$

$$X(p,q) = \sum_{l=0}^{L-1} \left((F(l,q)) * w_N{}^{ql} \right) * w_L{}^{ql} \tag{9}$$

$$X(p,q) = \sum_{l=0}^{L-1} (G(l,q)) * w_L{}^{ql} \tag{10}$$

By the above decimation, we can see that we have successfully broken a N=LM point DFT into two DFTs of DFTs of size L and M respectively. Here, F(l,q) is an M point DFT and X(p,q) is finally the summation of M many L point DFTs. $w_N{}^{ql}$ is the phase factor.

2.3 Complexity Analysis

Let's go step by step and calculate the complexity.

1. Computing L many F(l,q) M point DFT [$Mul = LM^2, Add = LM(M-1)$]
2. Multiplying phase factor [$Mul = LM$]
3. Computing M many X(p,q) L point DFT [$Mul = ML^2, Add = ML(L-1)$]

Total no of multiplications = LM (M + 1 + L) = N (M + L + 1)
Total no of additions = LM (M-1 + L-1) = LM (M + L -2) = N (M + L -2)

Remember the no of multiplications required before was N^2, but now it is N (L + M + 1) only. Similarly for additions, the count has reduced from N(N+1) to N(L+M-2)

2.4 Algorithm [input = x(n) , output = X(n)]

1. Store the input x(n) column wise in a 2-D array of L rows and M columns. [n = mL+l]
2. Compute row wise DFT [L many M-point-DFTs].
3. Multiply phase factor $w_N{}^{ql}$ in the whole 2D array.
4. Compute column-wise DFT [M many L-point-DFTs]. This can be broken into smaller DFTs.[Recursion]
5. Read the output row wise [k = pM + q].

3 Radix-2 FFT [N = 2^α]

3.1 Reduction: M=N/2, L=2

$$X(k) = \sum_{n=0}^{N-1} x(n) w_N^{kn} \qquad \text{where } k \in \{0, 1, \ldots, N-1\} \tag{11}$$

$$= \sum_{\text{n even}} x(n) w_N^{kn} + \sum_{\text{n odd}} x(n) w_N^{kn} \tag{12}$$

$$= \sum_{m=0}^{m=N/2-1} x(2m) w_N^{2km} + \sum_{m=0}^{m=N/2-1} x(2m+1) w_N^{k(2m+1)} \tag{13}$$

$$= \sum_{m=0}^{m=N/2-1} f_1(m) w_{N/2}^{km} + w_N^k \sum_{m=0}^{m=N/2-1} f_2(m) w_{N/2}^{km} \qquad \text{where } f_1(m) = x(2m), f_2(m) = x(2m+1) \tag{14}$$

$$= F_1(k) + w_N^k F_2(k) \qquad F_1(k), F_2(k) \text{ are M point DFTs} \tag{15}$$

$$\tag{16}$$

Now we know that $F_1(k)$ and $F_2(k)$ are both periodic with period M = N/2.

$$F_1(k + N/2) = F_1(k) \tag{17}$$

$$F_2(k + N/2) = F_2(k) \tag{18}$$

$$w_N^{k+N/2} = -w_N^k \tag{19}$$

$$\tag{20}$$

$$X(k) = F_1(k) + w_N^k F_2(k) \tag{21}$$

$$X(k + N/2) = F_1(k) + w_N^{k+N/2} F_2(k) \tag{22}$$

$$= F_1(k) - w_N^k F_2(k) \qquad \text{where } k \in \{0, 1, \ldots, N/2-1\} \tag{23}$$

$$\tag{24}$$

Complexity at this stage:

- Two N/2 point DFT and N/2 phase factor multiplications
- No of multiplications = $2 * (N/2)^2 + N/2$

Now, as we created G using F and the phase factor above, we create G_1 and G_2 here, which will be periodic as well.

$$G_1(k) = F_1(k) \tag{25}$$

$$G_2(k) = w_N^k F_2(k) \qquad \text{where } k \in \{0, 1, \ldots, N/2-1\} \tag{26}$$

$$\tag{27}$$

Then,

$$X(k) = G_1(k) + G_2(k) \tag{28}$$

$$X(k + N/2) = G_1(k) - G_2(k) \qquad \text{where } k \in \{0, 1, \ldots, N/2-1\} \tag{29}$$

$$\tag{30}$$

Now,

$$F_1(k) = V_{11}(k) + w_{N/2}^k V_{12}(k) \qquad \text{where } k \in \{0, 1, \ldots, N/4 - 1\} \tag{31}$$

$$F_2(k) = V_{21}(k) + w_{N/2}^k V_{21}(k) \tag{32}$$

$$F_1(k + N/4) = V_{11}(k) - w_{N/2}^k V_{12}(k) \tag{33}$$

$$F_2(k + N/4) = V_{21}(k) - w_{N/2}^k V_{21}(k) \tag{34}$$

V_{ij} are each $N/4$ point DFTs.
Complexity at this stage:

- Four $N/4$ point DFTs.
- $N/2$ phase factor multiplications for the first stage (multiply by w_N^{lq}).
- $N/2$ phase factor multiplications for the second stage (multiply by $w_{N/2}^{lq}$).
- Number of multiplications $= 4 \times \left(\frac{N}{4}\right)^2 + N/2 + N/2$

Similarly for N = 2^α, this reduction can be performed log(N) times.
Thus the total complexity of multiplication would be $\frac{N}{2} log(N)$
and the no of additions would be Nlog(N).

4 Example

4.1 8-point DFT of [1,2,3,4,5,6,7,8]

1. Store the input x(n) column wise in a 2-D array of L rows and M columns.
$$\begin{bmatrix} x[0] & x[4] \\ x[1] & x[5] \\ x[2] & x[6] \\ x[3] & x[7] \end{bmatrix} = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}$$
2. Compute row wise DFT [L many M-point-DFTs].
$$\begin{bmatrix} 6 & -4 \\ 8 & -4 \\ 10 & -4 \\ 12 & -4 \end{bmatrix}$$
3. Multiply phase factor ${w_8}^{ql}$ in the whole 2D array.
$$\begin{bmatrix} 6 & -4 \\ 8 & -4w_8 \\ 10 & -4w_8^2 \\ 12 & -4w_8^3 \end{bmatrix}$$
4. Compute column-wise DFT [M many L-point-DFTs]. Each column is 4 point so recursively computing DFT of this using 2 point DFTs. Mind that I am writing both the DFT calculations parrallelly, but their calculation is independent of each other. $\begin{bmatrix} 6 & 10 \\ 8 & 12 \end{bmatrix}, \begin{bmatrix} -4 & -4w_8^2 \\ -4w_8 & -4w_8^3 \end{bmatrix}$
5. 2 point row wise DFT
$\begin{bmatrix} 16 & -4 \\ 20 & -4 \end{bmatrix}, \begin{bmatrix} -4 - 4w_8^2 & -4 + 4w_8^2 \\ -4w_8 - 4w_8^3 & -4w_8 + 4w_8^3 \end{bmatrix}$
6. Multiply by phase factor ($w_4^{l'q'}$). Mind that previously we had phase factor w_8^{lq}, because we were calculating an 8 point DFT, here we are doing that for a 4 point DFT.
$\begin{bmatrix} 16 & -4 \\ 20 & -4w_4 \end{bmatrix}, \begin{bmatrix} -4 - 4w_8^2 & -4 + 4w_8^2 \\ -4w_8 - 4w_8^3 & (-4w_8 + 4w_8^3) * w_4 \end{bmatrix}$
   - $w_4 = w_8^2$

- $w_8^5 = -w_8$

$$\begin{bmatrix} 16 & -4 \\ 20 & -4w_8^2 \end{bmatrix}, \begin{bmatrix} -4-4w_8^2 & -4+4w_8^2 \\ -4w_8-4w_8^3 & -4w_8^3-4w_8 \end{bmatrix}$$

7. Perform column wise DFT for 4 point DFTs.

$$\begin{bmatrix} 36 & -4-4w_8^2 \\ -4 & -4+4w_8^2 \end{bmatrix}, \begin{bmatrix} -4-4w_8-4w_8^2-4w_8^3 & -4-4w_8+4w_8^2-4w_8^3 \\ -4+4w_8-4w_8^2+4w_8^3 & -4+4w_8+4w_8^2+4w_8^3 \end{bmatrix}$$

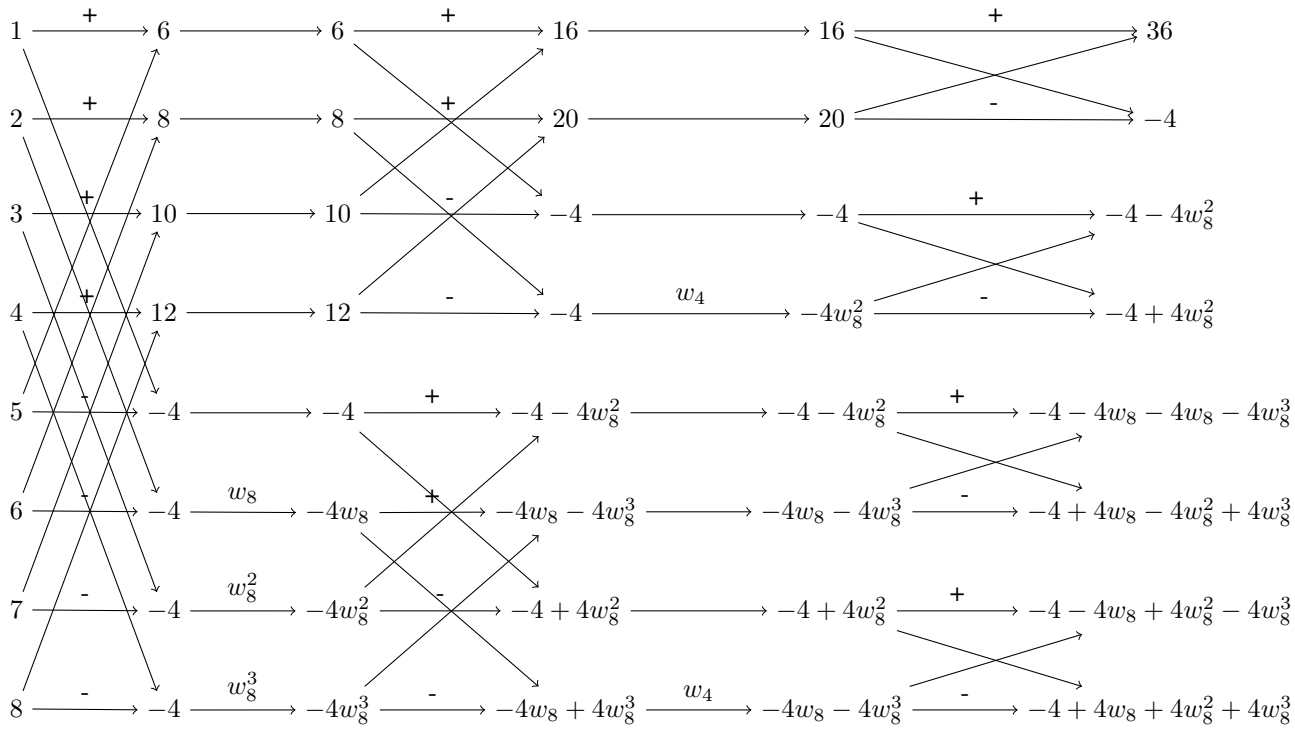
8. Rearrange back the output of 4 point DFTs back to their positions. Read the output of 4 point DFTs row wise and place them in proper index.

$$\begin{bmatrix} 6 & -4 \\ 8 & -4w_8 \\ 10 & -4w_8^2 \\ 12 & -4w_8^3 \end{bmatrix} \xrightarrow{columnwise4-pointDFT} \begin{bmatrix} 36 & -4-4w_8-4w_8^2-4w_8^3 \\ -4-4w_8^2 & -4-4w_8+4w_8^2-4w_8^3 \\ -4 & -4+4w_8-4w_8^2+4w_8^3 \\ -4+4w_8^2 & -4+4w_8+4w_8^2+4w_8^3 \end{bmatrix}$$

9. Read the output row wise for the 8-point DFT.

$$\begin{bmatrix} 36 \\ -4-4w_8-4w_8^2-4w_8^3 \\ -4-4w_8^2 \\ -4-4w_8+4w_8^2-4w_8^3 \\ -4 \\ -4+4w_8-4w_8^2+4w_8^3 \\ -4+4w_8^2 \\ -4+4w_8+4w_8^2+4w_8^3 \end{bmatrix}$$

4.2 Butterfly method



The output seems to be out of order. This happens because of the inherent property of DFT calculations, if you see the Divide and conquer approach above, the inputs are given in column wise order but the output is read row wise. Here also that happens and the trick to get this sequence in correct order is to use the bit reverse order of the 0-indexed indices.
For eg. 6 = 110 changes to 011 =3,
4 = 100 changes to 001 =1.
Thus the correct sequence is 000, 100, 010, 110, 001, 101, 011, 111

In decimal it is 0, 4, 2, 6, 1, 5, 3, 7.

5 Link to implementation code

- Github link to the implementation in C and Java

References

[1] John R. Buck Alan V. Oppenheim, Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall Signal Processing Series, 1999.

[2] Leo Chartrand Benoit Boulet. *Fundamentals of signals and systems*. Da Vinci Engineering Press, 2006.

[3] Dimitris K Manolakis John G. Proakis. *Digital Signal Processing: Principles, Algorithms and Applications*. Prentice Hall, 1995.

[3] [1] [2]