Fast Fourier Transform

Wu Yuhan

2020 Nov. 11

1 Introduction

This article is intended to introduce 1-dimensional 2^n-point FFT algorithm. Firstly, the definition of discrete fourier transform will be introduced. Then, we will study the easiest fast Fourier transform: Radix-2 DIT(decimation in time)-FFT algorithm. We will start from some simple examples: 2-point, 4-point and 8-point FFT. Then the 2^n-FFT will be discussed.

2 Discrete Fourier Transformation

Suppose $\{x(n)\}$ is a N-point array of data, where $n = 0, 1 \cdots, N-1$, and $\{X^F(k)\}$ is discrete fourier transformation of $\{x(n)\}$. The definition of $X^F(k)$ is

$$X^F(k) = \sum_{n=0}^{N-1} x(n) \exp(\frac{-2\pi kj}{N} \cdot n) \tag{1}$$

where j represents the imaginary number and $k = 0, 1, \cdots, N-1$. Here will introduce a notation W_N

$$W_N = \exp(\frac{-2\pi j}{N}) \tag{2}$$

Therefore,

$$\exp(\frac{-2\pi kj}{N} \cdot n) = W_N^{kn}$$

and

$$X^F(k) = \sum_{n=0}^{n-1} W_N^{kn} x(n)$$

For example, if N=4, 4-point DFT can be written in matrix form:

$$\begin{pmatrix} X^F(0) \\ X^F(1) \\ X^F(2) \\ X^F(3) \end{pmatrix} = \begin{pmatrix} W_4^{00} & W_4^{01} & W_4^{02} & W_4^{03} \\ W_4^{10} & W_4^{11} & W_4^{12} & W_4^{13} \\ W_4^{20} & W_4^{21} & W_4^{22} & W_4^{23} \\ W_4^{30} & W_4^{31} & W_4^{32} & W_4^{33} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{pmatrix} \tag{3}$$

3 Main algorithm

From the matrix representation of N-DFT, we can calculate its computational complexity is N^2. Our main motivation is to decompose N-point sequence into two $\frac{N}{2}$-point sequences. Therefore, we can reduce the computational complexity to $2 \cdot \frac{N^2}{4} + cN$, where c is a constant. Furthermore, we can apply this repeatedly.

The computational complexity will be reduced to $O(N \log N)$

$$\begin{aligned} X^F(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \\ &= \sum_{n=\text{even integer}} x(n) W_N^{kn} + \sum_{n=\text{odd integer}} x(n) W_N^{kn} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{(2r+1)k} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_N^2)^{rk} \end{aligned}$$

Noticed that,

$$W_N^2 = \exp(\frac{-2\pi j \cdot 2}{N}) = \exp(\frac{-2\pi j}{\frac{N}{2}}) = W_{\frac{N}{2}} \tag{4}$$

Therefore

$$X^F(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{\frac{N}{2}}^{rk} \tag{5}$$

Observe that

$$G^F(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{\frac{N}{2}}^{rk} \tag{6}$$

and

$$H^(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{\frac{N}{2}}^{rk} \tag{7}$$

are two $\frac{N}{2}$-point DFT. Therefore, we can conclude

$$X^F = G^F(k) + W_N^k H^F(k) \tag{8}$$

for $k = 0, 1, \cdots, \frac{N}{2} - 1$ When $k \geq \frac{N}{2}$,

$$\begin{aligned} X^F(k + \frac{N}{2}) &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{\frac{N}{2}}^{r(k+\frac{N}{2})} + W_N^(k + \frac{N}{2}) \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{\frac{N}{2}}^{r(k+\frac{N}{2})} \\ &= W_{\frac{N}{2}}^{\frac{N}{2}} \{ \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{\frac{N}{2}}^{rk} + W_N^{(k+\frac{N}{2})} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{\frac{N}{2}}^{rk} \} \end{aligned} \tag{9}$$

It is easy to verify that

$$W_{\frac{N}{2}}^{\frac{N}{2}} = \exp(\frac{-2\pi j \frac{N}{2}}{\frac{N}{2}}) = 1 \quad W_N^{\frac{N}{2}} = \exp(\frac{-2\pi j \frac{N}{2}}{N}) = -1 \tag{10}$$

Therefore

$$\begin{aligned} X^F(k + \frac{N}{2}) &= \{ \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{\frac{N}{2}}^{rk} - W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{\frac{N}{2}}^{rk} \} \\ &= G^F(k) - W_N^k H^F(k) \end{aligned} \tag{11}$$

4 Detailed examples

Let's start from the easiest situation 2-point DFT.

4.1 2-point DFT

N=2, $W_2 = \exp(\frac{-2\pi j}{2}) = -1$, we write it in matrix form:

$$\begin{pmatrix} X^F(0) \\ X^F(1) \end{pmatrix} = \begin{pmatrix} W_2^{00} & W_2^{01} \\ W_2^{10} & W_2^{11} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \end{pmatrix} \tag{12}$$

In other words $X^F(0) = x(0) + x(1)$ and $X^F(1) = x(0) - x(1)$.

4.2 4-point FFT

As we have proved, the 4-point DFT can be expressed as the sum of two 2-point DFT,

$$X^F(k) = A_1(k) + W_4^k A_2(k) \tag{13}$$

where $A_1(k) = DFT\{x(0), x(2)\}(k)$ and $A_2(k) = DFT\{x(1), x(3)\}$, explicitly

$$\begin{pmatrix} A_1(0) \\ A_1(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(2) \end{pmatrix} \tag{14}$$

and

$$\begin{pmatrix} A_2(0) \\ A_2(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x(1) \\ x(3) \end{pmatrix} \tag{15}$$

The 4-point DFT can be expressed as

$$\begin{pmatrix} X^F(0) \\ X^F(1) \\ X^F(2) \\ X^F(3) \end{pmatrix} = \begin{pmatrix} 1 & & W_4^0 & \\ & 1 & & W_4^1 \\ 1 & & W_4^2 & \\ & 1 & & W_4^3 \end{pmatrix} \begin{pmatrix} A_1(0) \\ A_1(1) \\ A_2(0) \\ A_2(1) \end{pmatrix} \tag{16}$$

And

$$\begin{pmatrix} A_1(0) \\ A_1(1) \\ A_2(0) \\ A_2(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(2) \\ x(1) \\ x(3) \end{pmatrix} \tag{17}$$

4.3 8-point FFT

To obtain 8-point FFT, we need to calculate two 4-point FFT first.

$$X^F(k) = B_0^F(k) + W_8^k B_1^F(k)$$

where $B_0(k) = DFT\{x(0), x(2), x(4), x(6)\}(k)$,and $B_1(k) = DFT\{x(1), x(3), x(5), x(7)\}$

Write them in matrix form:

$$\begin{pmatrix} X^F(0) \\ X^F(1) \\ X^F(2) \\ X^F(3) \\ X^F(4) \\ X^F(5) \\ X^F(6) \\ X^F(7) \end{pmatrix} = \begin{pmatrix} 1 & & & & W_8^0 & & & \\ & 1 & & & & W_8^1 & & \\ & & 1 & & & & W_8^2 & \\ & & & 1 & & & & W_8^3 \\ 1 & & & & W_8^4 & & & \\ & 1 & & & & W_8^5 & & \\ & & 1 & & & & W_8^6 & \\ & & & 1 & & & & W_8^7 \end{pmatrix} \begin{pmatrix} B_0(0) \\ B_0(1) \\ B_0(2) \\ B_0(3) \\ B_1(0) \\ B_1(1) \\ B_1(2) \\ B_1(3) \end{pmatrix} \tag{18}$$

and

$$\begin{pmatrix} B_0(0) \\ B_0(1) \\ B_0(2) \\ B_0(3) \end{pmatrix} = \begin{pmatrix} 1 & & W_4^0 & \\ & 1 & & W_4^1 \\ 1 & & W_4^2 & \\ & 1 & & W_4^3 \end{pmatrix} \begin{pmatrix} A_0(0) \\ A_0(1) \\ A_1(0) \\ A_1(1) \end{pmatrix} \tag{19}$$

where

$$\begin{pmatrix} A_0(0) \\ A_0(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(4) \end{pmatrix} \quad \begin{pmatrix} A_1(0) \\ A_1(1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x(2) \\ x(6) \end{pmatrix} \tag{20}$$

Let's work them together:

$$\begin{pmatrix} X^F(0) \\ X^F(1) \\ X^F(2) \\ X^F(3) \\ X^F(4) \\ X^F(5) \\ X^F(6) \\ X^F(7) \end{pmatrix} = \begin{pmatrix} 1 & & & & W_8^0 & & & \\ & 1 & & & & W_8^1 & & \\ & & 1 & & & & W_8^2 & \\ & & & 1 & & & & W_8^3 \\ 1 & & & & W_8^4 & & & \\ & 1 & & & & W_8^5 & & \\ & & 1 & & & & W_8^6 & \\ & & & 1 & & & & W_8^7 \end{pmatrix} \begin{pmatrix} 1 & & W_4^0 & & & & & \\ & 1 & & W_4^1 & & & & \\ 1 & & W_4^2 & & & & & \\ & 1 & & W_4^3 & & & & \\ & & & & 1 & & W_4^0 & \\ & & & & & 1 & & W_4^1 \\ & & & & 1 & & W_4^2 & \\ & & & & & 1 & & W_4^3 \end{pmatrix}$$

$$\times \begin{pmatrix} 1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & 1 & 1 & & & & \\ & & 1 & -1 & & & & \\ & & & & 1 & 1 & & \\ & & & & 1 & -1 & & \\ & & & & & & 1 & 1 \\ & & & & & & 1 & -1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{pmatrix}$$

5 Algorithm

To make the problem simple, we only study $N = 2^t$-point FFT.

5.1 Recursive method

Assume we are studying $N = 2^t$-point problem, and its correspond FFT algorithm is note as FFT[t]()

1. Input x[n] (A complex array of 2^t numbers)
2. Output X^F[k] (Complex array with 2^t numbers)
3. Required parameter: $t = log_2 N$

Our first step is to divide the input data into two array. Create two complex array a[r] and b[r] which have 2^{t-1} data respectively. Here is pseudo code.

```
    for(r=0;r<=N-1;r++){ /*N=2^t */
        a[r]=x[2*r];
        b[r]=x[2*r+1];
    }
```

Then we apply the recursive method calculate the DFT of a[n] and b[n].

$$\begin{aligned} a^F(k) &= FFT[t-1](a)(k) \\ b^F(k) &= FFT[t-1](b)(k) \quad k = 0, 1, \cdots, 2^{t-1} - 1 \end{aligned} \tag{21}$$

Here FFT[t-1] refers to the FFT algorithm that compute the DFT of the array with 2^{t-1} complex numbers. And here is pseudo code.

```
    a_F=FFT[t-1](a);
    b_F=FFT[t-1](b);
```

where a[n] and b[n] are two arrays with $\frac{N}{2} = 2^{t-1}$ complex numbers ,and a_F[k] and b_F[k] are the DFT of a[n] and b[n] respectively.

However a_F[k] and b_F[k] are only defined for $k \le 2^{t-1}-1$. We can expand them periodically,

$$\begin{aligned} a^F(2^{t-1}+k) &= a^F(k) \\ b^F(2^{t-1}+k) &= b^F(k) \quad k=0,1,\cdots,2^{t-1}-1 \end{aligned} \tag{22}$$

As we have showed in the previous chapter:

$$X^F(k) = a^F(k) + W_{2^t}^k b^F(k) \tag{23}$$

The pseudo code is

```
for(k=0;k<=N-1;k++){
    X_F[k]=a_F[k]+W^k*b_F[k]; /*W= exp(-2*pi*j/N) */
}
```

The computation has finished

5.2 A trick to improve the efficiency

We find that to calculate $W_{2^t}^k$ may take lots of computation. Actually, we can only compute once, and store them in a complex array.

1. Create an array $W_N[k] = \exp[\frac{-2\pi j}{N} \cdot k]$ where $N = 2^{t_0}$
2. Define a function $W(t,k) = \exp[\frac{-2\pi j \cdot k}{2^t}] = \exp[\frac{-2\pi j \cdot k}{2^{t_0}} \cdot 2^{t_0-t}] = W_N[k \cdot 2^{t_0-t}]$

5.3 Analyze computation complexity

First of all, the complexity of 2^t-point FFT is C(t). In each FFT[t], we need to calculate two FFT[t-1] to obtain $A^F[k]$ and $B^[k]$, then by

$$X^F(k) = A^F(k) + W_N^k B^F(k) \tag{24}$$

there will be 2^t complex multiplication and 2^t complex addition. To simplify, we note the complexity as $2 \cdot 2^t$.

Therefore:

$$C(t) = 2 \cdot C(t-1) + 2 \cdot 2^t \tag{25}$$

Our goal is to calculate $C(t)$.

To begin with, it is easy to evaluate that $C[1] = 4$. And, we divide 2^t on each side of the equation above.

$$\frac{C(t)}{2^t} = \frac{C(t-1)}{2^{t-1}} + 2 \tag{26}$$

Note $b(t) = \frac{C(t)}{2^t}$, and $b(1) = \frac{C(1)}{2} = 2$. Therefore,

$$b(t) = b(t-1) + 2 \tag{27}$$

And it is easy to calculate $b(t)$

$$b(t) = 2t \tag{28}$$

Therefore,

$$C(t) = b(t) * 2^t = 2t \cdot 2^t \tag{29}$$

Note that $N = 2^t$, so $t = log_2 N$, so

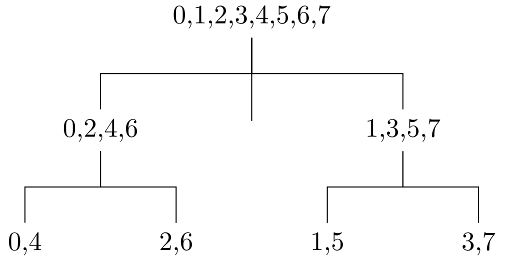
$$C(t) = 2N \log_2 N$$

6 Iteration method

6.1 Algorithm introduction

Although recursive method is easy to understand, there are many disadvantages of recursive method. For example, we cannot free the memory; we cannot apply parallel computation. Therefore, we will introduce another equivalent method: iteration method.

The main difference is that we can calculate from bottom to top. For example, in the case of 8-p FFT, we firstly calculate 4 2-p FFT; then compute 2 4-p FFT; last, finish the computation.



Our procedure is :

$$\begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix} \longrightarrow \begin{pmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{pmatrix} \longrightarrow \begin{pmatrix} A_0(0) \\ A_0(1) \\ A_1(0) \\ A_1(1) \\ A_2(0) \\ A_2(1) \\ A_3(0) \\ A_3(1) \end{pmatrix} \longrightarrow \begin{pmatrix} B_0(0) \\ B_0(1) \\ B_0(2) \\ B_0(3) \\ B_1(0) \\ B_1(1) \\ B_1(2) \\ B_1(3) \end{pmatrix} \longrightarrow \begin{pmatrix} X^F(0) \\ X^F(1) \\ X^F(2) \\ X^F(3) \\ X^F(4) \\ X^F(5) \\ X^F(6) \\ X^F(7) \end{pmatrix} \tag{30}$$

Here

1. $B_0 = DFT\{x(0), x(2), x(4), x(6)\}$
2. $B_1 = DFT\{x(1), x(3), x(5), x(7)\}$
3. $A_0 = DFT\{x(0), x(4)\}$
4. $A_1 = DFT\{x(2), x(6)\}$
5. $A_2 = DFT\{x(1), x(5)\}$
6. $A_3 = DFT\{x(3), x(7)\}$

6.2 Data rearrangement

As is shown above, a difficult part is to rearrange data. We need to change index {0,1,2,3,4,5,6,7} to {0,4,2,6,1,5,3,7}. Here I will only introduce an experience formula, I will write it in pseudo code. Assume $2^t = N$

```
int index[2^t];
index[0]=0;
for(i=0;i<=t-1;i++){
    for(l=0;l<=2^i-1;i++){
        index[2^i+l]=index[l]+2^(t-1-i)
    }
}
```

The new data array will be X_new[n]=x[index[n]]

Let's check this in the case of $N = 2^3$

```
index[0]=0

i=0
l=0: index[1]=index[2^0+0]=index[0]+2^(3-1-0)=4

i=1
l=0: index[2]=index[2^1+0]=index[0]+2^(3-1-1)=2
l=1: index[3]=index[2^1+1]=index[1]+2^(3-1-1)=6

i=2
l=0: index[4]=index[2^2+0]=index[0]+2^(3-1-2)=1
l=1: index[5]=index[2^2+1]=index[1]+2^(3-1-2)=5
l=2: index[6]=index[2^2+2]=index[2]+2^(3-1-2)=3
l=3: index[7]=index[2^2+3]=index[3]+2^(3-1-2)=7
```

6.3 Main program

Now let's finish our main program, I will show this in C language:

Define the function, with input complex xin[ ] and complex xout[ ]. "x" is the pointer of input data, "xf" is the pointer of an array that store the output.

```
void FFT(complex* xin,complex* xout)
```

Assume the input data is an array with $N = 2^t$ numbers, otherwise return error.

```
N=sizeof(x)/sizeof(x[0]);
/* Assume that we have define a function to calculate the logarithm.*/
t=log2(N);
```

Then let's calculate the

$$W_N[k] = \exp(\frac{-2\pi j \cdot k}{N}) \tag{31}$$

Then define a function

```
complex W(int i,int k){
    return W_N[k*2^i]
}
```

Then we continue to the key algorithm

```
complex x[t+1][N]
```

x[i] is an array to store the variable of each circle of calculation. For example:

$$x[0] = \begin{pmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{pmatrix} \quad x[1] = \begin{pmatrix} A_0(0) \\ A_0(1) \\ A_1(0) \\ A_1(1) \\ A_2(0) \\ A_2(1) \\ A_3(0) \\ A_3(1) \end{pmatrix} \quad x[2] = \begin{pmatrix} B_0(0) \\ B_0(1) \\ B_0(2) \\ B_0(3) \\ B_1(0) \\ B_1(1) \\ B_1(2) \\ B_1(3) \end{pmatrix} \quad x[3] = \begin{pmatrix} X^F(0) \\ X^F(1) \\ X^F(2) \\ X^F(3) \\ X^F(4) \\ X^F(5) \\ X^F(6) \\ X^F(7) \end{pmatrix}$$

In the real computation, when we calculate x[n+1], we can free the memory of x[n-1].

The main loop:

```
for(i=1;i<=t;i++){
    for(j=0;j<=(2^(t-i)-1);j++){
        for(l=0;l<=2^(i-1);l++){
            x[i][j*2^i+l]=x[i-1][j*2^i+l mod 2^(i-1)]
            +W(i,l)x[i-1][j*2^(i-1)+l mod 2^(i-1)]
```

```
                }
            }
        }
```

6.4 Explain

Let's explain this pseudo code in a detailed way. Assume $N = 8, t = \log_2 8 = 3$
In the case of i=2, we want to calculate the x[2]. Let's explain the loop:

```
for(j=0;j<=(2^(t-i)-1);j++)
```

In this case t=3,i=2, so it becomes

```
for(j=0;j<=1;j++)
```

As we can see,

$$x[2] = \begin{pmatrix} B_0(0) \\ B_0(1) \\ B_0(2) \\ B_0(3) \\ B_1(0) \\ B_1(1) \\ B_1(2) \\ B_1(3) \end{pmatrix}$$

For case j=0, refers that B_0 is stored in $\{x[2][0], x[2][1], x[2][2], x[2][3]\}$ and B_1 is stored in {x[2][4],x[2][5],x[2][6],x[2][7]}, which can also be expressed in $\{x[2][0+2^i], x[2][1+2^i], x[2][2+2^i], x[2][3+2^i]\}$, here i=2

As we have proved

$$B_0[k] = A_0[k] + W_4^k A_1[k]$$

$A_i[k]$ can be expanded in this way

$$A_i[k+2^1] = A_i[k]$$

This can be achieved by

```
for(k=0;k<=3,k++){
    A[k]=A[k mod 2]
}
```

Let's view this process again;

```
i=1, j=0,1,2,3
In case j=0, we calculate A0; j=1 A1;j=2 A2; j=3 A3

i=2 ,j=0,1;
j=0, we calculate B0;
j=1, we calculate B1;

i=3, j=0
We calculate XF.

And x[3] is the final result, output it.

for(i=0;i<=2^t-1;i++){
    xout[i]=x[3][i];
}
```