Lab 5 Report: Fast Fourier Transform With MATLAB

IEEEauthorblockN1st Qiu Kunyuan *EEE. Southern University of Science and Technology*
Shenzhen, PRC
11913019@mail.sustech.edu.cn

***Abstract*—This lab session conducts further discussions on the efficiency and computational cost of the discrete Fourier transformation(DFT). The matrix multiplication approach of the DFT is not only computationally intensive, but also consumes a lot of time for memory read and write operations when creating matrices. The theoretical complexity of DFT is $\mathcal{O}(n^2)$, and the memory allocations involved in actual execution increases the time complexity of DFT further. In order to cut down the time complexity of DFT operations, which is intensively adopted in modern digital signal processing(DSP), a new algorithm using divide-conquer strategy is implemented and the theoretical complexity of DFT is reduced to $\mathcal{O}(n \log_2 n)$. This algorithm is named Fast Fourier Transformation(FFT) and is used widely today.**

***Index Terms*—FFT, time complexity, high-performance computation, MATLAB**

I. Introduction

A. Frequency Range Shift

The FFT result is a sequence with subscript k and the corresponding frequency range of the result is $[0, 2\pi]$ rather than $[-\pi, \pi]$. For a bandpass signal with bandwidth B, its FFT result are not intuitive since the nonzero values locates in the interval $[0, B] \cup [B, 2\pi]$. This property of the *raw* FFT output make it difficult to process further.

To overcome this difficulty, a shift should be done so that the misplaced sequence is recovered. Let the desired shift be a map $fftshift : X[k] \mapsto X[k]$, this map is defined as

$$X[k - 2\pi] = X[k], k \in [\pi, 2\pi] \tag{1}$$

B. Zero Padding

The FFT requires the length of input sequence to be a integer power of 2. For a sequence with arbitrary length L, the input sequence are padded with zero so that the length requirement is satisfied:

$$N_Z = 2^{\lceil \log_2 L \rceil}, x_{in} = [x, \text{zeros}(1, N_Z - L)] \tag{2}$$

This technique can also be used to produce a finer sampling of the DTFT, for the frequency step is inversely proportional to the number of sampling points and the time domain multiplication with a step function equals to a frequency domain convolution with a sinc function.

C. FFT Algorithm

To boost the evaluation of the DFT, the divide-conquer strategy is tried since there are lots of preceding examples, such as Strassen method for matrix multiplication. The proportion of division is selected to 2, because multiplication or division of factor 2 is done by shift registers in digital circuit and the cost of each operation is one single clock pulse.

The DFT summation can be break into 2 parts, one is of odd indexes and the other is of even indexes:

$$\begin{aligned}\mathcal{F}(x[t]) &= \sum_{t=0}^{N-1} x[t] \exp\left(-j\frac{2\pi t}{N}k\right) \\ &= \sum_{m=0}^{\frac{N}{2}-1} x[2m] \exp\left(-j\frac{2\pi 2m}{N}k\right) \\ &+ \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] \exp\left(-j\frac{2\pi(2m+1)}{N}k\right)\end{aligned} \tag{3}$$

The equation above can be rearranged into a prettier form:

$$\begin{aligned}\mathcal{F}(x[n]) &= \sum_{m=0}^{\frac{N}{2}-1} x[2m] \exp\left(-j\frac{2\pi m}{N/2}k\right) \\ &+ \exp\left(-j\frac{2\pi}{N}k\right) \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] \exp\left(-j\frac{2\pi m}{N/2}k\right) \\ &= \mathcal{F}(x[2n]) + \mathcal{F}(x[2n+1])W_N^k\end{aligned} \tag{4}$$

With the deductions above, a recursion relation is established and the size of each recursion is half to the upper recursion step. The twiddle factor

$$W_N^k = \exp\left(-j\frac{2\pi k}{N}\right) \tag{5}$$

is a vector in equation (5) for

```
k=0:length(x(l*n))
```

is a vector of length N/l. Since the complex exponential factor, or the twiddle factor has the property that

$$\exp\left(-j\frac{2\pi k + N/2}{N}\right) = -\exp\left(-j\frac{2\pi k}{N}\right) \tag{6}$$

The recursion relation (4) can be simplified as

$$\begin{aligned}X[k] &= X_0[k] + W_k^N X_1[k] \\ X[k + N/2] &= X_0[k] - W_N^k X_1[k]\end{aligned}, k \in [0, N/2 - 1] \tag{7}$$

D. Complexity Analysis

Since the size of each sub-problem is halved after every single iteration, and the time consumption of the merging operation is $\mathcal{O}(n)$, the time complexity can be then obtained by solving the recursion below:

$$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \quad (8)$$

The fig.1 visualizes the vast complexity gap between FFT and DFT explicitly.

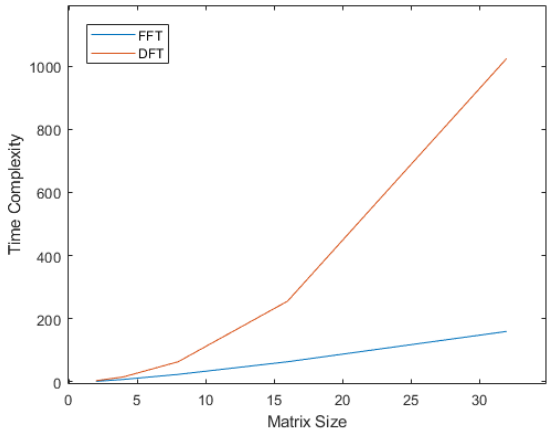

Fig. 1: Theoretical Time Complexity Plot

The recursion implementation of the FFT algorithm requires large stack memory due to the tail recursion. The avoidance of this problem leads to the conversion from tail recursion to nested loop, and sequential structure in terms of loop implementation is naturally suitable for MATLAB and FPGA programming.

II. Frequency Range Shifting and Zero Padding

A. Frequency Range Shift

The built-in function fftshift () is applied to recover the *misplaced* output sequence of original DFT function:

```
function [X,w]=DTFTsamples(x)
    N=length(x);
    w=linspace(-pi,pi,N);
    X=fftshift(DFTsum(x,N))/N;
end
```

Fig.2 clearly demonstrates that the shifting recovers the order of output sequence comparing to the original DFT output.

B. Zero Padding of Input

The spacing between samples of the DTFT is determined by the number of points in the DFT. This can lead to surprising results when the number of samples is too small. When the number of samples within a sampling interval is less than the Nyquist frequency

$$N > F_{\text{Nyquist}} = BT_S \quad (9)$$

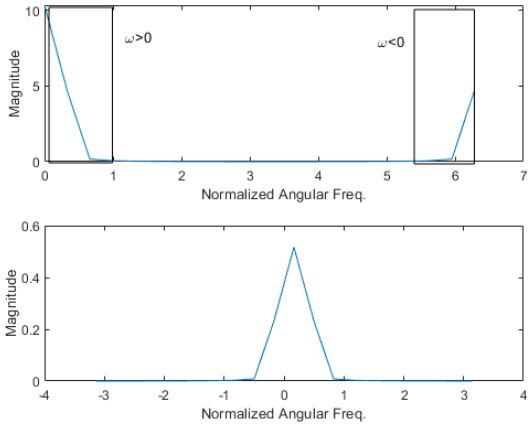

Fig. 2: DFT before shifting and after shifting

, the signal can no longer be recovered due to spectrum overlapping.

A larger sampling frequency increases the sampling frequency, and the frequency domain convolution acts as sinc interpolation that improves the similarity between the digital spectrum and the theoretical analog spectrum.

$$\mathcal{F}(x[n](u(n) - u(n+L))) = U[e^{j\omega}] * \text{sinc}(\omega L) \quad (10)$$

The demonstration Fig.3 from the following code is evident to this equation. A larger number of samples per interval produces a finer FFT spectrum.

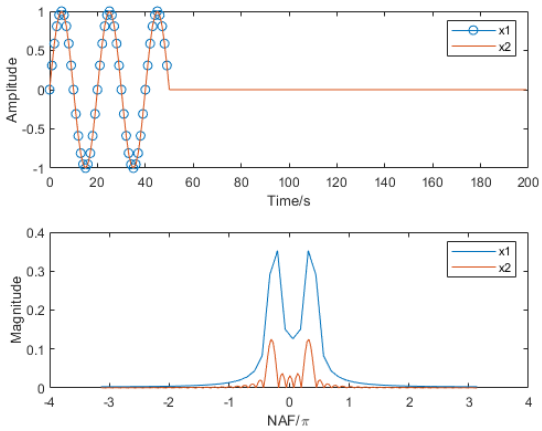

Fig. 3: 50 samples versus 200 samples

III. Fast Fourier Transformation

A. Single Stage Divide-Conquer

The divide-conquer method of FFT computation are composed by many consequentially stages of divide, conquer and merge. For the final and the largest stage, the steps can be programmed as following MATLAB code:

```
function X=dc1DFT(x)
```

```
    xe=x(1:2:length(x)-1);
    fxe=DFTsum(xe,length(xe));
    xo=x(2:2:length(x));
    fxo=DFTsum(xo,length(xo));
    twi=exp(-1j*2*pi*(0:length(x)/2-1)/length(x));
    X=[(fxe+fxo.*twi),(fxe-fxo.*twi)];
end
```

This code is a direct translation of equation (7) where the fxe , fxo are variable representation of the $N/2$ DFT of even and odd sequence X_0, X_1.

The number of multiplication can be easily obtained by analyzing the computations involved in the dc1DFT(x) function. The matrix implementation of DFT executes one real matrix multiplication to build the Vandermonde matrix, and then executes complex matrix multiplications twice to generate the twiddle matrix and apply the DFT transformation.

```
function Xk=DFTsum(xn,N)
    n=0:N-1;
    nk=n'*n; % n^2 real
    WNnk=exp(-1j*2*pi/N).^nk; % n^2 complex
    Xk=xn*WNnk; % n^2 complex
end
```

Therefore, one matrix DFT requires n^2 real multiplications and $2n^2$ complex multiplications, which accounts as $5n^2$ real multiplications eventually.

The number of multiplications executed in dc1DFT() can be divided into two parts, one is the butterfly computation and twiddle factor generation that costs $2n$ complex multiplications, the other is the two dcDFT() with size $n/2$ and requires $5n^2/4$ real multiplications. Merging the two parts together, the dcDFT() uses $4n + 5/4\ n^2$ real multiplications.

B. Recursive and Iterative Divide-Conquer

The subroutine DFTsum invoked by the single-stage divide-conquer FFT program can be substituted by another divide-conquer subroutine, and the only difference between the upper stage and the lower stage of recursion is the size of the input vector. Since the size of input vector can be conveniently controlled by specifying the initial and final indices, the iterative subroutine is reusable and hardware accelerations such as MKL are able to be applied.

The operations that are to be implemented in the reusable dcDFT(x) includes generation of corresponding twiddle factor W_N^k and the in-situ vector update. Both of the two operations are already optimized in many mathematical libraries, and the existing accelerations has been invoked automatically in MATLAB.

Based on the backward inference of the input sequence from the recursive results, it is clear that the initial input sequence needs to be rearranged in reverse binary order, i.e., the indices of the original sequence are converted to binary numbers, and then converted back to decimal after a bit inversion.

```
function [out,wout]=FFT1(a)
N=2^(ceil(log2(length(a))));%最近二次幂
a=[a,zeros(1,N-length(a))];
wout=linspace(0,2*pi,length(a));
S=1;
a=Reverse_Rader(a,N);
while S<=N
    T=N/S; %每层S循环的DFT问题规模
    Ws=exp(-1i*2*pi/S);
    for k=1:T% (对每个子段进行循环)
        l=(k-1)*S+(1:S/2);
        c1=a(l);
        c2=a(l+S/2).*(Ws.^(l-1));
        a(l)=c1+c2;
        a(l+S/2)=c1-c2;
    end
    S=S*2; %向上递归
end
out=a/N;
end
```

The Rader approach of bit inversion is of $\mathcal{O}(\log n)$ time complexity, exponentially faster than naively for loop:

```
function Y=Reverse_Rader(F,N)
LH=N/2;
J=LH;
N1=N-2;
for I=1:1:N1
    if I<J
        T=F(I+1);
        F(I+1)=F(J+1);
        F(J+1)=T;
    end
    K=LH; %K是比较位数的指示，一开始是二进制最高位，如果J的K位为1，则跳0，比较下一位
    while J>=K   %是0则不跳，退出循环
        J=J-K;
        K=K/2;
    end
    J=J+K; %K位跳1 还是不明白逻辑，太神棍了这段代码
end
Y=F;
end
```