The Quick Fourier Transform: An FFT Based on Symmetries

Group 20

Arshini Govindu: 2020102009
Shruti Kolachana: 2020102053

1 Abstract

The Fast Fourier transform(FFT) is used for the calculation of a DFT of length-2^M. The computations involved in this for complex data are lengthy and cumbersome if done directly. The algorithm to reduce the number of operations by a factor of two or four is described in this paper. The basic QFT is defined by splitting the data into real and complex parts and further into their odd and even parts. This enables us to reduce the number of operations as the odd integrals end up becoming zero so we can ignore those calculations. Extending this to the calculation of 2^M length data, we apply recursion similar to the algorithm used in the Cooley-Tukey FFT. DCT and DST are used in this algorithm.

2 Important Concepts and Short Discussion About The Existing Method

DFT has a finite input vector length while the DTFT processes an infinitely long signal Due to this reason, DTFT is mainly used for analytical purposes. Hence, we end up using the DFT for signal processing.

2.1 Fast Fourier Transform

The calculations involved in directly computing the discrete Fourier transform

$$F(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2jnk\pi}{N}} \qquad fork = 0, 1, ..., N-1 \tag{1}$$

are $O(N^2)$ as we can clearly see that the calculation is similar to that of a double sigma.

There are other algorithms like the Cooley-Tukey FFT which has a $O(N\log N)$ complexity.
Number of real multiplications = $2N.2N = 4N^2$
Number of real additions = $4N^2$

2.2 The Basic QFT for Arbitrary Length Data

The function x(n) has a real part($r(n)$) and an imaginary part($c(n)$), and these can be further divided into even and odd parts.

$$\begin{aligned} x(n) &= r(n) + jc(n) \\ &= (r_e(n) + r_o(n)) + j(c_e(n) + c_o(n)) \end{aligned} \tag{2}$$

Therefore, the DFT equation becomes

$$X(k) = \sum_{n=0}^{N-1} [(r_e(n) + r_o(n)) + j(c_e(n) + c_o(n))][cos(\frac{2kn\pi}{N}) - jsin(\frac{2kn\pi}{N})]$$

The sum over an integral number of periods of an odd function is zero:

$$r_e(n)sin(\frac{2kn\pi}{N}) = r_o(n)cos(\frac{2kn\pi}{N}) = c_e(n)\frac{2kn\pi}{N}) = c_o(n)cos(\frac{2kn\pi}{N}) = 0$$

$$X(k) = \sum_{n=0}^{N-1} [r_e(n)cos(\frac{2kn\pi}{N}) + c_e(n)sin(\frac{2kn\pi}{N})][c_e(n)cos(\frac{2kn\pi}{N}) - r_o(n)sin(\frac{2kn\pi}{N})]$$

For an even function, sum over half of the period = $\frac{1}{2}$(sum over the whole period)

$$X(k) = 2\sum_{n=0}^{\frac{N}{2}-1} [r_e(n)cos(\frac{2kn\pi}{N}) + c_e(n)sin(\frac{2kn\pi}{N})][c_e(n)cos(\frac{2kn\pi}{N}) - r_o(n)sin(\frac{2kn\pi}{N})]$$

$$\text{for } k = 0, 1, ..., N-1$$

$$cos(\frac{2k(N-n)\pi}{N}) = cos(2k\pi - \frac{2kn\pi}{N}) = cos(\frac{2kn\pi}{N}) \tag{3}$$

$$sin(\frac{2k(N-n)\pi}{N}) = sin(2k\pi - \frac{2kn\pi}{N}) = -sin(\frac{2kn\pi}{N}) \tag{4}$$

From (3) and (4),

$$X(N-k) = 2\sum_{n=0}^{\frac{N}{2}-1}[r_e(n)cos(\frac{2kn\pi}{N}) - c_e(n)sin(\frac{2kn\pi}{N})][c_e(n)cos(\frac{2kn\pi}{N}) + r_o(n)sin(\frac{2kn\pi}{N})]$$

$$\text{for } k = 0, 1, ..., \tfrac{N}{2} - 1$$

This allows us to use the same values for computing both X(k) and X(N-k). Number of real multiplications $= 2\frac{N}{2}\frac{N}{2} = N^2$

Number of real additions $= N^2 + 4N$
Analysis between these two show that basic QFT is the most efficient when we need to calculate the DFT for an arbitrary length N as it involves lesser number of operations.

3 The QFT for Length-2^M Data

We define the discrete Fourier transform as

$$DFT(k, N, x) = \sum_{n=0}^{N-1} x(n)e^{-\frac{j2kn\pi}{N}}$$

$$\text{for } k = 0, 1, ..., N-1$$

Discrete Cosine Transform of a length (N+1) sequence x(n):

$$DCT(k, N-1, x) = \sum_{n=0}^{N-1} x(n)cos(\frac{nk\pi}{N})$$

$$\text{for } k = 0, 1, ..., N$$

Discrete Sine Transform of a length (N-1) sequence x(n):

$$DST(k, N-1, x) = \sum_{n=0}^{N-1} x(n)sin(\frac{nk\pi}{N})$$

$$\text{for } k = 1, ..., N-1$$

These equations have been predefined. We reduce the $x_e(n)$ sequence by half as follows:

$$x_e(n) = x(n) + x(N-n) \qquad n = 1, ..., \frac{N}{2} - 1$$

and

$$x_e(0) = x(0), \quad x_e(\frac{N}{2}) = x(N/2)$$
$$x_o(n) = x(n) - x(N-n)$$

Similarly for the odd part of $x(n)$

$$x_o(n) = x(n) - x(N-n) \qquad n = 1, ..., \frac{N}{2} - 1$$

What we do now is split an N-length DFT into an $\frac{N}{2}+1$ length DCT and length $\frac{N}{2}-1$ DST for the first half from 1 to $N/2$

$$DFT(k, N, x) = DCT(k, N/2{+}1, x_e){-}jDST(k, N/2{-}1, x_o) \qquad k = 1, ..., N/2{-}1$$

and we define the DFT for the second half from $N/2+1$ to N as

$$DFT(N{-}k, N, x) = DCT(k, N/2{+}1, x_e){+}jDST(k, N/2{-}1, x_o) \qquad k = 1, ..., N/2{-}1$$

Now we move to the recursive decomposition of the DCT and DST, we use the symmetry relations

$$cos(\frac{k(N-n)\pi}{N}) = cos(\frac{nk\pi}{N}) \qquad k = 0, 2, ..., N$$

$$cos(\frac{k(N-n)\pi}{N}) = -cos(\frac{nk\pi}{N}) \qquad k = 1, 3, ..., N-1$$

to get

$$DCT(2k{+}1, N{+}1, x) = DCT(k, N/2{+}1, x'_o){+}DCT(k{+}1, N/2{+}1, x'_o) \qquad k = 0, 1, ...N/2{-}1$$

where

$$x'_o(n) = \frac{x(n) - x(N-n)}{2cos(\frac{n\pi}{N})}$$

Similarly we calculate the recursive decomposition of DST,

$$DST(N{-}1, N{-}1, x) = DST(N/2{-}1, N/2{-}1, x'_e){+}x(N/2)(-1)^{(N/2)+1} \qquad k = 1, ..., N/2{-}1$$

where

$$x'_e(n) = \frac{x(n) + x(N-n)}{2cos(\frac{n\pi}{N})}$$

3.1 Real Data QFT and computational complexity

In the QFT computation, the complex calculations occur at then end, so we can calculate the real and complex calculations separately

For real data:

$$O_M = \frac{N}{2}\log_2(N) - \frac{11}{8}N + 1$$

$$O_A = \frac{7}{4}\log_2(N) - 3N + 2$$

For complex data

$$O_M = N\log_2(N) - \frac{11}{4} + 2$$

$$O_A = \frac{7}{2}N\log_2(N) - 4N$$

The calculations involved for complex data is double that of real calculations.

3.2 Mixed Radix QFT

To find the N-point DFT, where $N = p * 2^m$ and p is prime, a combined version of the recursive radix-2 QFT terminated by a prime length QFT. In a radix-2 QFT, the last recursion ends with a half length of p.

4 Simulations and Results

The code for the normal DFT computation is in the github repository. The code has a complexity of $O(N^2)$. The basic QFT code also has similar complexity. Therefore, we can compare these two FFTs. On comparing, we can see that the QFT graph has little spikes while the FFT does not. The graphs are in the poster.
The length-2^M QFT has a complexity of $O(N\log N)$ which is comparable to the Cooley-Tukey algorithm which is also a $O(N\log N)$ algorithm. The Cooley-Tukey algorithm is less efficient than the QFT.

5 Conclusion

The basic QFT algorithm is more efficient than the direct methods. The length-2^M QFT is a lot faster than the radix-2 FFT as we had hoped to prove. It is much more suitable for real data as it involves nearly less than half the operations. Errors mostly occur due to inverse trigonometry multiplications. QFT for real data can be pruned more easily.

6 References

[1] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing. Englewood Cliffs, NJ: Prentice-Hall, 1989. vol. 34, pp. 265–277, 1985.
[2] H. Guo, G. A. Sitton, and C. S. Burrus, "The quick discrete Fourier transform," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process., Adelaide, Australia, Apr. 19–22, 1994, pp. III:445–448.