CM2208: Scientific Computing
Fourier Transform 2:
Digital Signal and Image Processing
Fast Fourier Transform

Prof. David Marshall

School of Computer Science & Informatics

The Fast Fourier Transform

How may the **DFT** may be computed **efficiently**?

Naive 1D Case: $O(N^2)$

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-2\pi i x u / N} \tag{1}$$

has to be evaluated for N values of u, which if done in the obvious way clearly takes N^2 multiplications.

It is possible to calculate the DFT more efficiently than this, using the **fast Fourier transform** or **FFT** algorithm, which reduces the number of operations to $O(N \log_2 N)$.

Historical Note

History of the Fast Fourier Transform (FFT)

1805 : Carl Friedrich Gauss tried to determine the orbit of certain asteroids from sample locations.

- Did not publish the results — other methods seemed to be more useful to solve this problem at the time!
- Gauss' collected works were published in **1866**
- Written in old form of latin!
- This was two years before Charles Fourier

1966 : Cooley and Tukey

- One the great inventions of 20th Century
- Revolutionised Digital Signal Processing, Image Processing
- Ubiquitous algorithm.

Between Gauss and Cooley and Tukey others touched on the FFT (e.g. Runge (1903), Yates 1932 (Other similar transforms), Danielson and Lanczos (1942).

The Fast Fourier Transform Algorithm

- We shall assume **for simplicity**[1] that N is a power of 2, $N = 2^n$.
- If we define ω_N to be the N^{th} root of unity given by $\omega_N = e^{-2\pi i/N}$, and set $M = N/2$, we have

$$F(u) = \frac{1}{2M} \sum_{x=0}^{2M-1} f(x)\omega_{2M}^{xu}. \tag{2}$$

- This can be split **apart** into two **separate** sums of **alternate terms** from the original sum:

$$F(u) = \frac{1}{2}\left(\frac{1}{M} \sum_{x=0}^{M-1} f(2x)\omega_{2M}^{(2x)u} + \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1)\omega_{2M}^{(2x+1)u} \right). \tag{3}$$

[1] In practice always **pad out** data to the nearest power of 2

The Fast Fourier Transform Algorithm

- Now, since the **square of a $2M^{th}$** root of unity is an **M^{th} root of unity,** we have that

$$\omega_{2M}^{(2x)u} = \omega_M^{xu} \tag{4}$$

and hence

$$F(u) = \frac{1}{2}\Bigg(\underbrace{\frac{1}{M}\sum_{x=0}^{M-1} f(2x)\omega_M^{xu}}_{F_{even}(u)} + \underbrace{\frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)\omega_M^{xu}\,\omega_{2M}^{u}}_{F_{odd}(u)}\Bigg). \tag{5}$$

- If we call the two sums demarcated above $F_{even}(u)$ and $F_{odd}(u)$ respectively, then we have:

$$F(u) = \frac{1}{2}\left(F_{even}(u) + F_{odd}(u)\omega_{2M}^{u}\right). \tag{6}$$

The Fast Fourier Transform Algorithm

- Note that each of $F_{even}(u)$ and $F_{odd}(u)$ for $u = 0, \ldots, M-1$ is in itself a **discrete Fourier transform** over $N/2 = M$ points.
- **How does this help us?**

Well

$$\omega_M^{M+u} = \omega_M^u \qquad \text{and} \qquad \omega_{2M}^{M+u} = -\omega_{2M}^u, \tag{7}$$

and we can also write

$$F(u+M) = \frac{1}{2}\left(F_{even}(u) - F_{odd}(u)\omega_{2M}^u\right). \tag{8}$$

A **shift** in M ($F(u+M)$) now becomes a **multiplication** of $F_{odd}(u)$ by ω_{2M}^u.

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

The Fast Fourier Transform Algorithm

- Thus, we can compute an N-point DFT by dividing it into two parts:
  - The **first** half of $F(u)$ for $u = 0, \ldots, M-1$ can be found from Eqn. 6,
  - The **second** half for $u = M, \ldots, 2M-1$ can be found simply be **reusing** the same terms differently as shown by Eqn. 8.
  - This is obviously a **divide and conquer** method.

The Fast Fourier Transform Algorithm

To show how many operations this requires:

- Let $T(n)$ be the **time taken** to perform a transform of size $N = 2^n$, measured by the **number of multiplications** performed.
- The above analysis shows that

$$T(n) = 2T(n-1) + 2^{(n-1)}, \tag{9}$$

  - **first** term on the right hand side coming from the **two transforms of half** the **original size**, and
  - **second** term coming from the **multiplications** of F_{odd} **by** ω_{2M}^{u}.
- **Induction** can be used to prove that

$$T(n) = 2^{(n-1)} \log_2 2^n = \frac{1}{2} N \log_2 N, \tag{10}$$

The Fast Fourier Transform Algorithm

- A similar argument can also be applied to the **number of additions** required, to show that the algorithm as a whole takes time $O(N \log_2 N)$.
- **Also Note** that the same algorithm can be used with a little modification to perform the inverse DFT too.
  - Going back to the definitions of the DFT and its inverse,

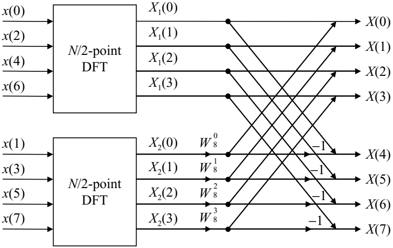
$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-2\pi i x u / N} \tag{11}$$

and

$$f(x) = \sum_{x=0}^{N-1} F(u) e^{2\pi i x u / N}, \tag{12}$$

Illustration of the FFT (Butterfly Network): 8 Point Example

Decomposition of an **N-point DFT** into two **$N/2$DFTs**, $\mathbf{N = 8}$:



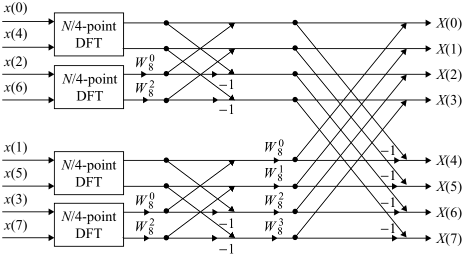
- Each **butterfly**[2] involves just a **single complex multiplication**, one addition, and one subtraction.

[2]The structure above is called the **butterfly network** because of its crisscross appearance

Illustration of the FFT (Butterfly Network):8 Point Example Cont.

Since N is a **power of 2**, $N/2$ is an even number.

- Each of these **$N/2$-point** DFTs can be computed by **two smaller** $N/4$-point DFTs.



By repeating the same process, we will finally obtain a set of 2-point DFTs since N is a power of 2.

Inverse Fast Fourier Transform Algorithm

- If we take the **complex conjugate** of the **second equation**, we have that

$$f^*(x) = \sum_{x=0}^{N-1} F^*(u) e^{-2\pi i x u / N}. \quad (13)$$

- This now looks (**apart** from a factor of $1/N$) like a forward DFT, rather than an inverse DFT.

Thus to compute an inverse DFT:

- take the conjugate of the Fourier space data,
- put conjugate through a **forward** DFT algorithm,
  - **Multiply** all complex parts by -1.
- take the conjugate of the result, at the same time multiplying each value by N.

2D Fast Fourier Transform Algorithm

- The same fast **1D Fourier transform algorithm** can be used — applying the **separability** property of the 2D transform.
- **Rewrite** the **2D DFT** as:

$$
\begin{aligned}
F(u,v) &= \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi i(xu+yv)/N} \\
&= \frac{1}{N}\sum_{x=0}^{N-1} e^{-2\pi i xu/N}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi i yv/N}. \quad (14)
\end{aligned}
$$

  - **right hand** sum is basically just a one-dimensional DFT if x is held constant.
  - **left hand** sum is then another one-dimensional DFT performed with the numbers that come out of the first set of sums.

Order of the 2D Fast Fourier Transform Algorithm

We can compute a **two-dimensional DFT** by:

- performing a one-dimensional DFT for each value of x, *i.e.* for each column of $f(x, y)$, then
- performing a one-dimensional DFT in the opposite direction (for each row) on the resulting values.

This requires a **total** of $2N$ **one dimensional** transforms, so the overall process takes time $O(N^2 \log_2 N)$.