Fast Fourier Transform

6.046 Design and Analysis of Algorithms
MIT
Michael Kapralov
11 Feb 2014

Fast Fourier Transform

One of the top 10 algorithms of the 20th century, a major tool in scientific computing and computer science.

Applications:
- Discrete Fourier Transform
  - signal processing (e.g. image, video compression)

Polynomial multiplication:

$$A(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_d z^d$$
$$B(z) = b_0 + b_1 z + b_2 z^2 + \ldots + b_d z^d$$
$$A(z) \cdot B(z) = a_0 b_0 + (a_0 b_1 + b_0 a_1) z + \ldots + a_d b_d z^{2d}$$
$$= c_0 + c_1 z + \ldots + c_{2d} z^{2d} = C(z)$$

What is the k-th coefficient of the product?

$$c_k =$$
Polynomial multiplication:

\[ A(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_d z^d \]
\[ B(z) = b_0 + b_1 z + b_2 z^2 + \ldots + b_d z^d \]
\[ A(z) \cdot B(z) = a_0 b_0 + (a_0 b_1 + a_1 b_0) z + \ldots + a_d b_d z^{2d} \]
\[ = c_0 + c_1 z + \ldots + c_{2d} z^{2d} = C(z) \]

What is the k-th coefficient of the product?
\[ c_k = \sum_{j=0}^{k} a_j b_{k-j} \text{ for all } k = 0, 1, \ldots, 2d, \]
where \(a_{d+1} = a_{d+2} = \ldots = a_{2d} = 0, b_{d+1} = b_{d+2} = \ldots = b_{2d} = 0\) (padding by zeros)
c is the convolution of \(a = (a_0, a_1, \ldots, a_{2d})\) and \(b = (b_0, b_1, \ldots, b_{2d})\)

How many arithmetic operations?
\[ \Theta(d^2) \]

Operations on polynomials in point/value form
Given: \((x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_{N-1}, A(x_{N-1}))\)

Point/value representation: choose \(x_0, \ldots, x_{N-1}\) and write down values of \(A\) at \(x_i, i = 0, \ldots, N-1\):

\[ A(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_d z^d \]

\[ (x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_{N-1}, A(x_{N-1})) \]

Evaluation: compute \(A(x_0) = a_0 + x_0 (a_1 + x_0 (a_2 + \ldots + a_d x_0 \ldots ))\)

Number of operations? Horner’s rule, \(O(\text{degree})\):

\[ O(d^2) \] by solving linear system, can do better \((O(d^2) \text{ time})\)
Given coeffs of $A(x), B(x)$, compute coeffs of $C(x)$

Strategy:
1. evaluate $A(x), B(x)$ at some points $x_0, x_1, \ldots, x_{N-1}, N > 2d(\text{why?})$
2. then $C(x_k) = A(x_k)B(x_k), k = 0, \ldots, N-1$
3. polynomial interpolation to recover coefficients of $(c_0, c_1, \ldots, c_{N-1})$ from $C(x_0), C(x_1), \ldots, C(x_{N-1})$

Choose $N$ a power of 2.

Disturbing question: would need to evaluate $A(x), B(x)$ at $x_0, x_1, \ldots, x_{N-1}$ at a cost of almost $O(1)$ per point...

$\Theta(N^2)$ via the naive approach...

Divide and conquer

To evaluate $A(z)$ on $\{x_0, x_1, \ldots, x_{N-1}\}$, divide $A(z) = A_{\text{even}}(z^2) + z \cdot A_{\text{odd}}(z^2)$ of polynomials halves at each step!

Let $d$ denote the degree of our polynomials. We have

$T(d) \leq 2T(d/2) + O(?)$

Degree of polynomials halves at each step!

Let $d$ denote the degree of our polynomials. We have

$T(d) \leq 2T(d/2) + O(N)$

At the leaves: evaluate polynomials of degree 1 at $N$ points!

So $\Theta(N^2)$ time...
To evaluate $A(z)$ on $\{x_0, x_1, \ldots, x_{N-1}\}$,

**Divide** Split $A(z)$ as $A(z) = A_{\text{even}}(z^2) + z \cdot A_{\text{odd}}(z^2)$

**Conquer** Evaluate $A_{\text{even}}(z), A_{\text{odd}}(z)$ on $x_0^2, x_1^2, \ldots, x_{N-1}^2$

**Combine** Let $A(x_j) = A_{\text{even}}(x_j^2) + x_j \cdot A_{\text{odd}}(x_j^2)$, $j = 0, \ldots, N-1$

We need

$\{x_0^2, x_1^2, \ldots, x_{N-1}^2\}$

to be twice smaller than

$\{x_0, x_1, \ldots, x_{N-1}\}$

Take $x_0 = -x_1, x_2 = -x_3, \ldots, x_{N-2} = -x_{N-1}$!

At the next step, all numbers are positive, so cannot repeat...

Will choose points $x_0, x_1, \ldots, x_{N-1}$ in the complex plane

Complex numbers

$a + b \cdot i = r e^{i \phi}$,

where

$\r = \sqrt{a^2 + b^2}$

phase $\phi = \arctan(b/a)$

Definition

$\omega$ is an $N$-th root of unity if $\omega^N = 1$. A primitive root of unity if $\omega^k \neq 1$ for all $k < N$.

We evaluate $A(z)$ on the set $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$, where $\omega$ is a primitive root of unity of order $N$. Let $\omega = e^{2\pi i/N}$

Will choose points $x_0, x_1, \ldots, x_{N-1}$ in the complex plane

Complex numbers

$a + b \cdot i = r e^{i \phi}$,

where

$\r = \sqrt{a^2 + b^2}$

Will have $r = 1$, i.e. points on the unit circle

phase $\phi = \arctan(b/a)$

Definition

$\omega$ is an $N$-th root of unity if $\omega^N = 1$. A primitive root of unity if $\omega^k \neq 1$ for all $k < N$.

We evaluate $A(z)$ on the set $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$, where $\omega$ is a primitive root of unity of order $N$. Let $\omega = e^{2\pi i/N}$
Let \(\omega_N = e^{2\pi i/N}\), i.e. a primitive root of unity of order \(N\).

**Lemma**

\[
\left\{\omega^{2k}_N\right\}_{k=0}^{N/2-1} = \left\{\omega^{k}_{N/2}\right\}_{k=0}^{N/2-1}
\]

**Proof.**

\(\omega_{N/2}^{2k} = e^{2\pi i(2k)/N} = e^{2\pi i k/(N/2)} = \omega_k^{N/2}\)

Squares of all roots of unity of order \(N\) = all roots of unity of order \(N/2\)

---

### Fast evaluation at roots of unity

**Divide** Split \(A(z)\) as \(A(z) = A_{\text{even}}(z^2) + z \cdot A_{\text{odd}}(z^2)\)

**Conquer** Evaluate \(A_{\text{even}}(z), A_{\text{odd}}(z)\) on \(\left\{1, \omega_{N/2}^2, \omega_{N/2}^4, \ldots, \omega_{N/2}^{2(N-1)}\right\}\)

**Combine** Let \(A(\omega^j) = A_{\text{even}}(\omega_{N/2}^{2j}) + \omega^j \cdot A_{\text{odd}}(\omega_{N/2}^{2j})\), \(j = 0, \ldots, N - 1\)

**Theorem**

Can evaluate a polynomial \(A(z) = a_0 + a_1 z + \ldots + a_{N-1} z^{N-1}\) at \(\{1, \omega_N, \omega_{N}^2, \ldots, \omega_{N}^{N-1}\}\) in \(O(N \log N)\) time.

Cooley-Tukey’1965, Gauss’1805

---

### Fast matrix-vector product

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & \omega & \omega^2 & \ldots & \omega^{N-1} & \omega^{N/2}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{N-1}
\end{pmatrix} =
\begin{pmatrix}
A(\omega^0) \\
A(\omega^1) \\
A(\omega^2) \\
\vdots \\
A(\omega^{N-1})
\end{pmatrix}
\]

\(O(N \log N)\) time

---

Are we done? Partially.

Our strategy was:

1. evaluate \(A(x), B(x)\) at \(1, \omega^1, \omega^2, \ldots, \omega^{N-1}\) \(\text{OK}\)
2. then \(C(\omega^k) = A(\omega^k) B(\omega^k), k = 0, \ldots, 2d\) \(\text{OK}\)
3. polynomial interpolation to recover coefficients of \((c_0, c_1, \ldots, c_{N-1})\) from \(C(1), C(\omega^1), \ldots, C(\omega^{N-1})\) \(?\)

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & \omega & \omega^2 & \ldots & \omega^{N-1} \\
1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\
1 & \omega^3 & \omega^6 & \ldots & \omega^{3(N-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{N-1} & \omega^{2(N-1)} & \ldots & \omega^{(N-1)(N-1)}
\end{pmatrix}
\begin{pmatrix}
c_0 \\
c_1 \\
c_2 \\
\vdots \\
c_{N-1}
\end{pmatrix} =
\begin{pmatrix}
C(\omega^0) \\
C(\omega^1) \\
C(\omega^2) \\
\vdots \\
C(\omega^{N-1})
\end{pmatrix}
\]

denote this by \(M\) want this know this
Need to solve
$$M \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} C(\omega^0) \\ C(\omega^1) \\ C(\omega^2) \\ \vdots \\ C(\omega^{N-1}) \end{bmatrix}$$

Multiply by $M^{-1}$!
$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix} = M^{-1} \cdot \begin{bmatrix} C(\omega^0) \\ C(\omega^1) \\ C(\omega^2) \\ \vdots \\ C(\omega^{N-1}) \end{bmatrix}$$

How do we find $M^{-1}$?

We have $M = (\omega^j)_{j=0}^{N-1}$; guess that $M^{-1} = \frac{1}{N}(\omega^{-j})_{j=0}^{N-1}$

We have $M = (\omega^j)_{j=0}^{N-1}$; guess that $M^{-1} = \frac{1}{N}(\omega^{-j})_{j=0}^{N-1}$

We have $M = (\omega^j)_{j=0}^{N-1}$; guess that $M^{-1} = \frac{1}{N}(\omega^{-j})_{j=0}^{N-1}$

Claim
$(\omega^i) \cdot (\omega^{-j}) = N \cdot I$, where $I$ is the identity matrix

Proof.
The $(i,j)$th entry of the product $(\omega^i) \cdot (\omega^{-j})$ is
$$\sum_{k=0}^{N-1} \omega^{ik} \cdot \omega^{-kj} = \sum_{k=0}^{N-1} \omega^{(i-j)k}$$

- If $i = j$, then $\sum_{k=0}^{N-1} \omega^{ik} \cdot \omega^{-kj} = N$
- If $i \neq j$, we have $\sum_{k=0}^{N-1} \omega^{(i-j)k} = \frac{\omega^{(i-j)N-1}}{\omega^{i-j}-1} = 0$

How long will this product take?
$$M = \frac{1}{N}(\omega^{-j}) = \frac{1}{N}(\omega^{(N-j)})$$
Any real function \( f(t) \) on 0, 1, ..., \( N - 1 \) can be represented as a sum of cosines and sines. Let

\[
\begin{pmatrix}
\hat{f}(0) \\
\hat{f}(1) \\
\hat{f}(2) \\
\vdots \\
\hat{f}(N-1)
\end{pmatrix} = (\omega^j \cdot \omega^{-jt}) = \begin{pmatrix}
f(0) \\
f(1) \\
f(2) \\
\vdots \\
f(N-1)
\end{pmatrix}
\]

So that

\[ f(t) = \frac{1}{N} \sum_{j=0}^{N-1} \hat{f}(j) \cdot \omega^{-jt}, \quad c_j = \frac{1}{N} \hat{f}(j) = a_j + i \cdot b_j \]

Take real parts:

\[ \text{Re}(f(t)) = \text{Re} \left( \sum_{j=0}^{N-1} c_j \cdot \omega^{-jt} \right) \]

Fast polynomial multiplication

Our strategy was:

1. evaluate \( A(x), B(x) \) at 1, \( \omega^1, \omega^2, \ldots, \omega^{N-1} \) OK
2. then \( C(\omega^k) = A(\omega^k)B(\omega^k), k = 0, \ldots, N - 1 \) OK
3. polynomial interpolation to recover coefficients of \( (c_0, c_1, \ldots, c_{N-1}) \) from \( C(1), C(\omega^1), \ldots, C(\omega^{N-1}) \) OK

Steps 1 and 3 take \( O(N \log N) \) time, step 2 takes \( O(N) \) time.

Discrete Fourier Transform

Definition

For a vector \( x \) of length \( N \) with complex coefficients the Discrete Fourier Transform of \( x \) is given by

\[ \hat{x} = F \cdot x, \text{ where } F = (\omega^j_N) \]

Definition

FFT is an \( O(N \log N) \) time algorithm for computing the DFT

We proved: \( F(x \ast y) = Fx \cdot Fy \) (convolution in time domain is multiplication in Fourier domain)

Applications in signal processing

Any real function \( f(t) \) on 0, 1, ..., \( N - 1 \) can be represented as a sum of cosines and sines. Let

\[
\begin{pmatrix}
\tilde{f}(0) \\
\tilde{f}(1) \\
\tilde{f}(2) \\
\vdots \\
\tilde{f}(N-1)
\end{pmatrix} = (\omega^j \cdot \omega^{-jt}) = \begin{pmatrix}
f(0) \\
f(1) \\
f(2) \\
\vdots \\
f(N-1)
\end{pmatrix}
\]

So that

\[ f(t) = \frac{1}{N} \sum_{j=0}^{N-1} \tilde{f}(j) \cdot \omega^{-jt}, \quad c_j = \frac{1}{N} \tilde{f}(j) = a_j + i \cdot b_j \]

Take real parts:

\[ \text{Re}(f(t)) = \text{Re} \left( \sum_{j=0}^{N-1} c_j \cdot \omega^{-jt} \right) \]
Any real function \( f(t) \) on \( 0, 1, \ldots, N - 1 \) can be represented as a sum of cosines and sines.

Let

\[
\begin{pmatrix}
  f(0) \\
f(1) \\
f(2) \\
\vdots \\
f(N-1)
\end{pmatrix}
= \frac{1}{N} \mathbf{\omega}^{-j\mathbf{\tilde{f}}^T}.
\]

So that

\[
f(t) = \frac{1}{N} \sum_{j=0}^{N-1} \tilde{f}(j) \cdot \omega^{-jt}, \quad c_j = \frac{1}{N} \tilde{f}(j) = a_j + i \cdot b_j
\]

Take real parts:

\[
f(t) = \sum_{j=0}^{N-1} \text{Re}(c_j \cdot (\cos(2\pi j \cdot t/N) + i \cdot \sin(2\pi j \cdot t/N))).
\]

We have

\[
f(t) = \sum_{j=0}^{N-1} a_j \cdot \cos(2\pi j \cdot t/N) - b_j \cdot \sin(2\pi j \cdot t/N).
\]

If \( f \) does not change fast, most of the mass is on low frequencies.

Image size \( \approx 1200 \times 900 \) pixels

3 bytes per pixel: \( \geq 3 \)MB on disk!

In reality: 74KB on disk, i.e. 15-fold compression!

2d Fourier transform!

Can decompose a function of two variables into sum of products of complex exponentials (or sines and cosines):

\[
f(x, y) = \sum_{a=0}^{N-1} \sum_{b=0}^{N-1} c_{a,b} \omega^{ax} \cdot \omega^{by}
\]

If function is smooth, most Fourier mass is on low frequencies!

- partition image into small square blocks (e.g. 16 \times 16)
- take Fourier transform
- only keep large coefficients

- partition image into small square blocks (e.g. 16 \times 16)
- take Fourier transform
- only keep large coefficients
If function is smooth, most Fourier mass is on low frequencies!

Only keep dominant Fourier coefficients!

Conclusions and final remarks

- FFT computes the DFT in $O(N\log N)$ time
- Very efficient implementations are known, e.g. FFTW (developed at MIT)

- Is it optimal? Nobody knows...
- Can do better if the signal has few 'large' coefficients
- Can do better if the signal has $k$ 'large' coefficients, $k \ll N$
  - Can run in time $O(k \log^2 N)$
  - Time is sub-linear in $N$ (do not even read the whole input!)