Fast Fourier Transform: Brief History

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.


Importance not fully realized until advent of digital computers.

Fast Fourier Transform: Applications

Applications.
- Optics, acoustics, quantum physics, telecommunications, control systems, signal processing, speech recognition, data compression, image processing.
- DVD, JPEG, MP3, MRI, CAT scan.
- Numerical solutions to Poisson’s equation.

Polynomials: Coefficient Representation

Polynomial. [coefficient representation]
\[ A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \]
\[ B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} \]

Add: \( O(n) \) arithmetic operations.
\[ A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1) x + \cdots + (a_{n-1} + b_{n-1}) x^{n-1} \]

Evaluate: \( O(n) \) using Horner’s method.
\[ A(x) = a_0 + (x (a_1 + x (a_2 + \cdots + x (a_{n-2} + x (a_{n-1}) \cdots)) \cdots) \]

Multiply (convolve): \( O(n^2) \) using brute force.
\[ A(x) \times B(x) = \sum_{j=0}^{2n-2} c_j x^j, \text{ where } c_j = \sum_{j=0}^{n} a_j b_{n-j} \]
Polynomials: Point-Value Representation


Corollary. A degree n-1 polynomial A(x) is uniquely specified by its evaluation at n distinct values of x.

\[ y_j = A(x_j) \]

Converting Between Two Polynomial Representations

Tradeoff. Fast evaluation or fast multiplication. We want both!

<table>
<thead>
<tr>
<th>Representation</th>
<th>Multiply</th>
<th>Evaluate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficient</td>
<td>( O(n^2) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Point-value</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

Goal. Make all ops fast by efficiently converting between two representations.

Coefficient representation \( a_0, a_1, \ldots, a_{n-1} \)

Point-value representation \( (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \)

Polynomials: Point-Value Representation

Polynomial. [point-value representation]

\[
A(x) : (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})
\]

\[
B(x) : (x_0, z_0), \ldots, (x_{n-1}, z_{n-1})
\]

Add: \( O(n) \) arithmetic operations.

\[
A(x) + B(x) : (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1})
\]

Multiply: \( O(n) \), but need 2n-1 points.

\[
A(x) \times B(x) : (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})
\]

Evaluate: \( O(n^2) \) using Lagrange’s formula.

\[
A(x) = \sum_{i=0}^{n-1} y_i \prod_{j \neq i} \frac{(x - x_j)}{(x_i - x_j)}
\]

Converting Between Two Polynomial Representations: Brute Force

Coefficient to point-value. Given a polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), evaluate it at n distinct points \( x_0, \ldots, x_n \).

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  \vdots \\
  y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
  1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\
  1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1}
\end{bmatrix}
\begin{bmatrix}
  a_0 \\
  a_1 \\
  \vdots \\
  a_{n-1}
\end{bmatrix}
\]

\( O(n^2) \) for matrix-vector multiply

\( O(n^3) \) for Gaussian elimination

Point-value to coefficient. Given n distinct points \( x_0, \ldots, x_{n-1} \) and values \( y_0, \ldots, y_{n-1} \), find unique polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \) that has given values at given points.
Coefficient to Point-Value Representation: Intuition

Coefficient to point-value. Given a polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \).

Divide. Break polynomial up into even and odd powers.
- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \).
- \( A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^4 \).
- \( A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3 \).
- \( A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2) \).
- \( A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2) \).

Intuition. Choose two points to be \( \pm 1 \).
- \( A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1) \). Can evaluate polynomial of degree \( \leq n \) at 2 points by evaluating two polynomials of degree \( \geq \frac{n}{2} \) at 1 point.
- \( A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1) \).

Discrete Fourier Transform

Coefficient to point-value. Given a polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \).

Key idea: choose \( x_k = \omega^k \) where \( \omega \) is principal \( n \)th root of unity.

\[
\begin{bmatrix}
 y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
 1 & 1 & 1 & \cdots & 1 \\
 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\
 1 & \omega^3 & \omega^6 & \cdots & \omega^{2(n-1)} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

\( \Downarrow \quad \text{Discrete Fourier transform} \)

\( \Downarrow \quad \text{Fourier matrix } F_n \)

Roots of Unity

Def. An \( n \)th root of unity is a complex number \( x \) such that \( x^n = 1 \).

Fact. The \( n \)th roots of unity are: \( \omega^0, \omega^1, \ldots, \omega^{n-1} \) where \( \omega = e^{2\pi i / n} \).

Pf. \( (\omega^k)^n = (e^{2\pi i k / n})^n = (e^{2\pi i})^{2k} = (-1)^{2k} = 1 \).

Fact. The \( \frac{n}{2} \)th roots of unity are: \( v^0, v^1, \ldots, v^{n/2-1} \) where \( v = e^{4\pi i / n} \).

Fact. \( v^2 = v \) and \( (v^2)^k = v^k \).
**Fast Fourier Transform**

**Goal.** Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \cdots + a_{n-1} x^{n-1}$ at its $n$'th roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

**FFT Summary**

**Theorem.** FFT algorithm evaluates a degree $n-1$ polynomial at each of the $n$'th roots of unity in $O(n \log n)$ steps.

**Running time.** $T(2n) = 2T(n) + O(n) \Rightarrow T(n) = n \log_2 n$.

**Divide:** break polynomial up into even and odd powers.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{(n-2)/2}$,
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{(n-1)/2}$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.

**Conquer:** Evaluate degree $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2}n$'th roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n/2-1}$.

**Combine.**
- $A(\omega^k) = A_{even}(\omega^k) + \omega^k A_{odd}(\omega^k), \ 0 \leq k < n/2$
- $A(\omega^{kn}) = A_{even}(\omega^k) - \omega^k A_{odd}(\omega^k), \ 0 \leq k < n/2$

**FFT Algorithm**

```c
fft(n, a0, a1, ..., an-1) {
  if (n == 1) return a0
  (e0, e1, ..., en/2-1) % FFT(n/2, a0, a2, a4, ..., an-2)
  (d0, d1, ..., dn/2-1) % ...
  for k = 0 to n/2 - 1 {
    y_k = a0 + \omega^k d_k
    y_{k+n/2} = a0 - \omega^k d_k
  }
  return (y0, y1, ..., yn-1)
}
```

**Recursion Tree**

- The recursion tree shows the division of the polynomial into even and odd parts, which are recursively evaluated.
- The tree structure reflects the divide-and-conquer approach of the FFT algorithm.

- Coefficient representation: $a_0, a_1, \ldots, a_{n-1}$
- Point-value representation: $(\omega^0, y_0), (\omega^1, y_1), \ldots, (\omega^{n-1}, y_{n-1})$
Inverse FFT: Proof of Correctness

Claim. $F_n$ and $G_n$ are inverses.

Pf.

\[
\left\{ F_n \ G_n \right\}_{k,k'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega_j^{k k'} = \left\{ \begin{array}{ll} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{array} \right. \]

Summation lemma. Let $\omega$ be a principal $n^{th}$ root of unity. Then

\[
\sum_{j=0}^{n-1} \omega_j^k = \begin{cases} n & \text{if } k \equiv 0 \mod n \\ 0 & \text{otherwise} \end{cases}
\]

Pf.

- If $k$ is a multiple of $n$ then $\omega^k = 1 \Rightarrow$ sums to $n$.
- Each $n^{th}$ root of unity $\omega^k$ is a root of $x^n - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{n-1})$.
- If $\omega^k = 1$ we have: $1 + \omega^k + \omega^{2k} + \ldots + \omega^{(n-1)k} = 0 \Rightarrow$ sums to $0$.

Inverse FFT: Algorithm

\[
\text{ifft}(n, a_0, a_1, \ldots, a_{n-1}) \{ \\
\quad \text{if } (n == 1) \text{ return } a_0 \\
\quad (e_0, e_1, \ldots, e_{n/2-1}) \leftarrow \text{FFT}(n/2, a_0, a_2, a_4, \ldots, a_{n-2}) \\
\quad (d_0, d_1, \ldots, d_{n/2-1}) \leftarrow \text{FFT}(n/2, a_1, a_3, a_5, \ldots, a_{n-1}) \\
\quad \text{for } k = 0 \text{ to } n/2 - 1 \{ \\
\quad \quad \omega^k \leftarrow \omega^{2\pi i k/n} \\
\quad \quad Y_k \leftarrow (e_k + \omega^k d_k) / n \\
\quad \quad Y_{k+n/2} \leftarrow (e_k - \omega^k d_k) / n \\
\quad \}\end{array}
\]

\[
\text{return } (Y_0, Y_1, \ldots, Y_{n-1}) \}
\]
**Inverse FFT Summary**

**Theorem.** Inverse FFT algorithm interpolates a degree n-1 polynomial given values at each of the n\(^n\) roots of unity in O(n \log n) steps.

1. Assumes n is a power of 2.

![Diagram of Inverse FFT Summary]

---

**Polynomial Multiplication**

**Theorem.** Can multiply two degree n-1 polynomials in O(n \log n) steps.

![Diagram of Polynomial Multiplication]

---

**FFT in Practice**

**Fastest Fourier transform in the West.** [Frigo and Johnson]
- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

**Implementation details.**
- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to “shape” of the problem.
- Core algorithm is nonrecursive version of Cooley-Tukey radix 2 FFT.
- O(n \log n), even for prime sizes.

Reference: [http://www.fftw.org](http://www.fftw.org)