Chapter 5
Divide and Conquer

5.6 Convolution and FFT

Fast Fourier Transform: 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.

Fast Fourier Transform: Brief History

- Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.
- Importance not fully realized until advent of digital computers.

The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. -Charles van Loan
Polynomials: Coefficient Representation

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

Add: \( O(n) \) arithmetic operations.
\[ A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1) x + \cdots + (a_{n-1} + b_{n-1}) x^{n-2} \]

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

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

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

Multiply: \( O(n) \), but need \( 2n-1 \) points.
\[ A(x) \times B(x) = (x_0, y_0 z_0), \ldots, (x_{2n-1}, y_{2n-1} z_{2n-1}) \]

Evaluate: \( O(n^2) \) using Lagrange’s formula.
\[ A(x) = \sum_{j=0}^{n-1} y_j \prod_{j \neq k} \frac{x - x_j}{x_k - x_j} \]

Polynomials: Point-Value Representation

Fundamental theorem of algebra. [Gauss, PhD thesis] A degree \( n \) polynomial with complex coefficients has \( n \) complex roots.

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

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.
Converting Between Two Polynomial Representations: Brute Force

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

\[
\begin{bmatrix}
y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
x_0 & x_0^2 & ... & x_0^{n-1} \\ x_1 & x_1^2 & ... & x_1^{n-1} \\ x_2 & x_2^2 & ... & x_2^{n-1} \\ \vdots \\ x_{n-1} & x_{n-1}^2 & ... & x_{n-1}^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1}
\end{bmatrix}
\]

O\((n^3)\) for matrix-vector multiply

Vandermonde matrix is invertible iff \( x_i \) distinct

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

Intuition. Choose four points to be \( \pm 1, \pm i \).
- \( A(1) = A_{\text{even}}(1) + A_{\text{odd}}(1) \).
- \( A(-1) = A_{\text{even}}(-1) - A_{\text{odd}}(-1) \).
- \( A(i) = A_{\text{even}}(-1) + i A_{\text{odd}}(-1) \).
- \( A(-i) = A_{\text{even}}(-1) - i A_{\text{odd}}(-1) \).

Can evaluate polynomial of degree \( \leq n \) at 4 points by evaluating two polynomials of degree \( \leq \frac{n}{2} \) at 2 points.

Coefficient to Point-Value Representation: Intuition

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

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

Intuition. Choose two points to be \( \pm 1 \).
- \( A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1) \).
- \( A(-1) = A_{\text{even}}(-1) - 1 A_{\text{odd}}(-1) \).

Can evaluate polynomial of degree \( \leq n \) at 2 points by evaluating two polynomials of degree \( \leq \frac{n}{2} \) at 1 point.

Discrete Fourier Transform

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

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

\[
\begin{bmatrix}
y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & ... & 1 \\ 1 & \omega & \omega^2 & ... & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & ... & \omega^{2(n-1)} \\ \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & ... & \omega^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1}
\end{bmatrix}
\]

\( \uparrow \) Discrete Fourier transform \( \uparrow \) 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: $\omega^0, \omega^1, ..., \omega^{n-1}$ where $\omega = e^{\frac{2\pi i}{n}}$.

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

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

Fact. $\omega^2 = v$ and $(\omega^2)^k = \sqrt{k}$.

FFT Algorithm

```plaintext
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) & FFT(n/2, a1,a3,a5,...,an-1)
    for k = 0 to \(n/2 - 1\) {
        \(\omega^k = e^{\frac{2\pi ik}{n}}\)
        \(y_k = e_k + \omega^k \ d_k\)
        \(y_{k+n/2} = e_k - \omega^k \ d_k\)
    }
    return (y0,y1,...,yn-1)
}
```

Fast Fourier Transform

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

Divide. Break polynomial up into even and odd powers.

- $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + ... + a_{n/2-2} x^{(n-2)/2}$.
- $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + ... + a_{n/2-1} x^{(n-1)/2}$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.

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

Combine.

- $A(\omega^k) = A_{\text{even}}(\sqrt{k}) + \omega^k A_{\text{odd}}(\sqrt{k}), \ 0 \leq k \leq n/2$
- $A(\omega^{kn}) = A_{\text{even}}(\sqrt{k}) - \omega^k A_{\text{odd}}(\sqrt{k}), \ 0 \leq k \leq n/2$

\[\omega^k = (\omega^k)^2 = (\omega^{kn})^2\]

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.  This assumes $n$ is a power of 2.

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

\[a_0, a_1, ..., a_{n-1} \rightarrow (\omega^0, y_0), ..., (\omega^{n-1}, y_{n-1})\]
Inverse FFT

Claim. Inverse of Fourier matrix is given by following formula.

Consequence. To compute inverse FFT, apply same algorithm but use $\omega^{-1} = e^{-2\pi i/n}$ as principal $n^{th}$ root of unity (and divide by $n$).

Inverse FFT: Proof of Correctness

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

Pf.

$$
(F_nG_n)_{ij} = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{ik} \omega^{-jk} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-j)j} = \begin{cases} 
1 & \text{if } k = k' \\
0 & \text{otherwise}
\end{cases}
$$

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

$$
\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} 
n & \text{if } k \equiv 0 \text{ mod } n \\
0 & \text{otherwise}
\end{cases}
$$

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: $1 + \omega^k + \omega^{2k} + \ldots + \omega^{k(n-1)} = 0 \Rightarrow$ sums to 0.
Inverse FFT: Algorithm

```c
ifft(n, a0, a1, ..., an-1) {
    if (n == 1) return a0
    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a1, ..., an-2)
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a3, ..., an-1)
    for k = 0 to n/2 - 1 {
        ωk ← e^{-2πik/n}
        yk ← (ek + ωk dk) / n
        yk+n/2 ← (ek - ωk dk) / n
    }
    return (y0, y1, ..., yn-1)
}
```

Inverse FFT Summary

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

This assumes n is a power of 2.

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)

Polynomial Multiplication

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

---

**Inverse FFT: Algorithm**

```c
ifft(n, a0, a1, ..., an-1) {
    if (n == 1) return a0
    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a1, ..., an-2)
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a3, ..., an-1)
    for k = 0 to n/2 - 1 {
        ωk ← e^{-2πik/n}
        yk ← (ek + ωk dk) / n
        yk+n/2 ← (ek - ωk dk) / n
    }
    return (y0, y1, ..., yn-1)
}
```

**Inverse FFT Summary**

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

- Assumes n is a power of 2.

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)

**Polynomial Multiplication**

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

---

**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)