# FFT: Fast Polynomial Multiplications

Jie Wang

University of Massachusetts Lowell
Department of Computer Science

# Overview

- So far we have learned five basic algorithm design techniques: (1) sorting and searching; (2) divide-and-conquer; (3) greedy selection; (4) dynamic programming, and (5) linear programming. Another common technique is to use math tricks (actually DP and LP are in this category).
- Fast Fourier transform for computing polynomial multiplications is a typical math trick.

# Polynomials

- General form:

$$A(x) = a_0 x^0 + a_1 x + \cdots a_{n-1} x^{n-1} = \sum_{j=0}^{n-1} a_j x^j.$$

- $A(x)$ has **degree** $k$ if its highest nonzero coefficient is $a_k$, denoted by $\text{degree}(A) = k$.
- The degree of a polynomial of **degree-bound** $n$ may be any integer between 0 and $n-1$.
- Unless otherwise stated, all polynomials we will consider have degree-bound $n$.

# Evaluation and Summation

- Point evaluation (**Horner's rule**):

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))\cdots)).$$

- Runtime of point evaluation: $\Theta(n)$.
- Summation:

$$A(x) + B(x) = \sum_{j=0}^{n-1}(a_j + b_j)x_j.$$

- Runtime of summation: $\Theta(n)$.

# Multiplication and Convolution

- Multiplication: $C(x) = A(x)B(x)$, where $C(x)$ has degree-bound $2n-1$,

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j$$

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}.$$

- $\mathbf{c} = (c_1, c_2, \ldots, c_{2n-2})$ is called the **convolution** of $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ and $\mathbf{b} = (b_1, b_2, \ldots, b_{n-1})$, denoted by $\mathbf{c} = \mathbf{a} \otimes \mathbf{b}$.
- Runtime: $\Theta(n^2)$.
- Note: Since $C(x)$ belongs to polynomials of degree-bound $2n$, for computational convenience, we will treat $C(x)$ as a polynomial of degree bound $2n$ (rather than $2n-1$).

# Representations

- **Coefficient representation**: A polynomial of degree-bound $n$ can be represented by $(a_0, a_1, \ldots, a_{n-1})$.
- **Point-value representation**: $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$, where $y_i = A(x_i)$ and $x_i \neq x_j$ if $i \neq j$.
- Given a polynomial, computing a point-value representation is straightforward and the runtime is $\Theta(n^2)$.

# Interpolation

- **Interpolation**: The process from point-value representation to coefficient representation.
- **Lagrange's formula**: For any set $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ of $n$ point-value pairs with pairwise different $x_k$, there is a unique polynomial $A(x)$ of degree-bound $n$ such that $y_k = A(x_k)$ for $k = 0, \ldots, n-1$, where

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}.$$

This indicates the **uniqueness of interpolating polynomials**.

# Point-value Representations and Operations

- Suppose

$$A : \{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$$
$$B : \{(x_0, y_0'), \ldots, (x_{n-1}, y_{n-1}')\}.$$

- Addition:

$$C(x) = A(x) + B(x) : \{(x_0, y_0 + y_0'), \ldots, (x_{n-1}, y_{n-1} + y_{n-1}')\}.$$

- Multiplication needs extended point-value representations on $2n$ points:

$$A : \{(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})\}$$
$$B : \{(x_0, y_0'), \ldots, (x_{2n-1}, y_{2n-1}')\}.$$

  Then

$$C(x) = A(x)B(x) : \{(x_0, y_0 y_0'), \ldots, (x_{2n-1}, y_{2n-1} y_{2n-1}')\}.$$

# Multiplications

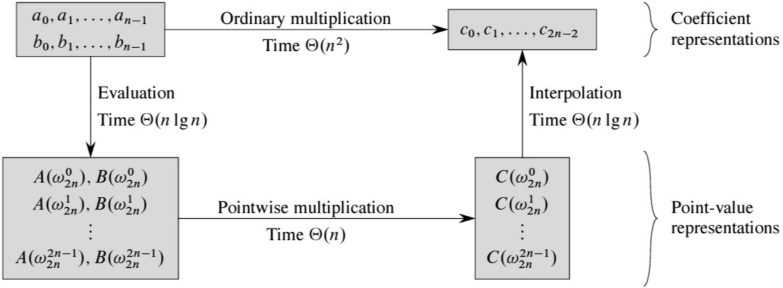


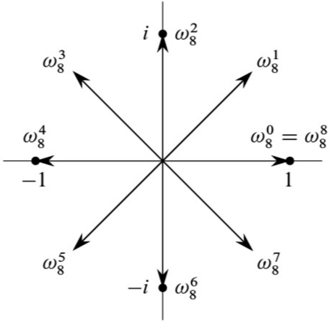

**Figure 30.1** A graphical outline of an efficient polynomial-multiplication process. Representations on the top are in coefficient form, while those on the bottom are in point-value form. The arrows from left to right correspond to the multiplication operation. The $\omega_{2n}$ terms are complex $(2n)$th roots of unity.

# Discrete and Fast Fourier Transforms (DFT, FFT)

- **Complex root of unity**: Let $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ be complex roots of $\omega^n = 1$, where

$$\omega_n^k = e^{2\pi ik/n}, \;\; k = 0, 1, \ldots, n-1.$$

- Recall that $e^{iu} = \cos(u) + i\sin(u)$.



# Lemmas

- **Lemma 30.3 (Cancellation lemma)**

$$\omega_{dn}^{dk} = \omega_n^k$$

  for any integers $n \geq 0$, $k \geq 0$, and $d > 0$.
  **Proof**. $\omega_{dn}^{dk} = (e^{2\pi i/dn})^{dk} = (e^{2\pi i/n})^k = \omega_n^k$.
- **Corollary 30.4**.

$$\omega_n^{n/2} = \omega_2 = -1.$$

- **Lemma 30.5 (Halving lemma)** If $n > 0$ is even, then

$$(\omega_n^{k+n/2})^2 = (\omega_n^k)^2.$$

  **Proof**. By Corollary 30.4 that $\omega_n^{n/2} = \omega_2 = -1$, we have $\omega_n^{k+n/2} = -\omega_n^k$, and so $(\omega_n^{k+n/2})^2 = (\omega_n^k)^2$.

# Summation Lemma

- **Summation lemma**

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0$$

for any integer $n > 0$ and any integer $k \neq 0$ with $k$ not divisible by $n$.
**Proof**.

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^n - 1}{\omega_n^k - 1} = 0.$$

# The DFT

- Let

$$A(x) = \sum_{j=0}^{n-1} a_j x^j,$$

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}.$$

- $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$ is the DFT of $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$, denoted by $\mathbf{y} = \mathrm{DFT}_n(\mathbf{a})$.

# The FFT

- Assume that $n$ is a power of 2.
- Divide-and conquer. Rewrite $A(x)$ as

$$
\begin{aligned}
A(x) &= A^{[0]}(x^2) + xA^{[1]}(x^2), \text{ where} \\
A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}, \\
A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}.
\end{aligned}
$$

- Only need to evaluate two "half-size" polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at

$$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2,$$

and combine the results.

# An Important Observation

By the halving lemma, there are only $n/2$ distinct values in $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$, where

$$\begin{aligned}
(\omega_n^0)^2 &= (\omega_n^{n/2})^2, \\
(\omega_n^1)^2 &= (\omega_n^{1+n/2})^2, \\
&\ldots, \\
(\omega_n^{n/2-1})^2 &= (\omega_n^{n-1})^2.
\end{aligned}$$

# Recursive FFT

RECURSIVE-FFT$(a)$

$n = a.length$ // $n$ is a power of 2
**if** $n == 1$
    **return** $a$
$\omega_n = e^{2\pi i/n}$
$\omega = 1$
$a^{[0]} = (a_0, a_2, \ldots, a_{n-2})$
$a^{[1]} = (a_1, a_3, \ldots, a_{n-1})$
$y^{[0]} =$ RECURSIVE-FFT$(a^{[0]})$
$y^{[1]} =$ RECURSIVE-FFT$(a^{[1]})$
**for** $k = 0$ **to** $n/2 - 1$
    $y_k = y_k^{[0]} + \omega\, y_k^{[1]}$
    $y_{k+(n/2)} = y_k^{[0]} - \omega\, y_k^{[1]}$
    $\omega = \omega\, \omega_n$
**return** $y$ // $y$ is assumed to be a column vector

Runtime: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$.

# Line 11

For $y_0, y_1, \ldots, y_{n/2} - 1$, line 11 yields

$$\begin{aligned} y_k &= y_k^{[0]} + \omega_n^k \cdot y_k^{[1]} \\ &= A^{[0]}(\omega_{n/2}^k) + \omega_n^k \cdot A^{[1]}(\omega_{n/2}^k) \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^k \cdot A^{[1]}(\omega_n^{2k}) && \text{by the cancellation lemma} \\ &= A(\omega_n^k). \end{aligned}$$

# Line 12

For $y_{n/2}, y_{n/2+1}, \ldots, y_{n-1}$, line 12 yields

$$\begin{aligned}
y_{k+n/2} &= y_k^{[0]} - \omega_n^k \cdot y_k^{[1]} \\
&= y_k^{[0]} + \omega_n^{k+n/2} \cdot y_k^{[1]} && \text{since } \omega_n^{k+n/2} = -\omega_n^k \\
&= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} \cdot A^{[1]}(\omega_n^{2k}) && \text{by the cancellation lemma} \\
&= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} \cdot A^{[1]}(\omega_n^{2k+n}) && \text{since } \omega_n^{2k+n} = \omega_n^{2k} \\
&= A(\omega_n^{k+n/2}).
\end{aligned}$$

# Interpolation

- Rewrite $\mathbf{y} = \mathrm{DFT}_n(\mathbf{a})$ as

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

- The $(j,k)$ entry of the inverse of the Vandermonde matrix $V_n$ is $\omega_n^{-kj}/n$.
  **Proof**.

$$[V_n^{-1}V_n]_{jj'} = \sum_{k=0}^{n-1}(\omega_n^{-kj/n}(\omega_n^{kj'}) = \sum_{k=0}^{n-1}\omega_n^{k(j'-j)}/n = \begin{cases} 1, & \text{if } j = j' \\ 0, & \text{otherwise.} \end{cases}.$$

- $\mathbf{a} = \mathrm{DFT}_n^{-1}(\mathbf{y})$.

# Coefficients

- For $j = 0, 1, \ldots, n-1$,

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}.$$

- Modify recursive FFT to replace **a** with **y** and $\omega_n$ with $\omega_n^{-1}$, we can computer $\mathrm{DFT}_n^{-1}$ in $\Theta(n \log n)$ time.
- This means that we can transform the coefficient representation to the point-value representation in $O(n \log n)$ time, and so is the other direction.
- **Theorem 30.8 (Convolution theorem)** For any two vectors **a** and **b** of length $n$, where $n$ is a power of 2,

$$\mathbf{a} \otimes \mathbf{b} = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(\mathbf{a}) \cdot \mathrm{DFT}_{2n}(\mathbf{b})),$$

where **a** and **b** are padded with 0s to length $2n$ and $\cdot$ denotes the componentwise product of two $2n$-element vectors.