Polynomials and the FFT

My T. Thai @ UF
Polynomials

- A polynomial in the variable x:

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

- Polynomial addition

\[ A(x) + B(x) = \sum_{j=0}^{n-1} a_j x^j + \sum_{j=0}^{n-1} b_j x^j = \sum_{j=0}^{n-1} (a_j + b_j) x^j = C(x) \]

- Polynomial multiplication

\[ A(x) B(x) = \sum_{j=0}^{n-1} a_j x^j \sum_{j=0}^{n-1} b_j x^j = \sum_{j=0}^{2n-2} c_j x^j = C(x) \]

where \[ c_j = \sum_{k=0}^{j} a_k b_{j-k} \]
Representing polynomials

- A coefficient representation of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ of degree bound $n$ is a vector of coefficients $a = (a_0, a_1, \ldots, a_{n-1})$.

- Horner’s rule to compute $A(x_0)$
  
  $$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))))$$

  Time: $O(n)$

- Given $a = (a_0, a_1, \ldots, a_{n-1})$, $b = (b_0, b_1, \ldots, b_{n-1})$
  
  - Sum: $c = a + b$, takes $O(n)$ time
  - Product: $c = a \otimes b$ (convolution of $a$ and $b$), takes $O(n^2)$ time
A point-value representation of a polynomial \( A(x) \) of degree-bound \( n \) is a set of \( n \) point-value pairs
\[
\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}
\]
- All of the \( x_k \) are distinct
- \( y_k = A(x_k) \)

**Theorem 30.1 (Uniqueness of an interpolating polynomial)**
For any set \( \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\} \) of \( n \) point-value pairs such that all the \( x_k \) values are distinct, there is a unique polynomial \( A(x) \) of degree-bound \( n \) such that \( y_k = A(x_k) \) for \( k = 0, 1, \ldots, n-1 \).

**Proof:**
\[
\begin{pmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-1}
\end{pmatrix}
= 
\begin{pmatrix}
y_0 \\
y_1 \\
\vdots \\
y_{n-1}
\end{pmatrix}
\]

\[ Da = y \]
\[ |D| = \prod_{0 \leq j < k \leq n-1} (x_k - x_j) \neq 0 \]
\[ \Rightarrow a = D^{-1}y \]
Operations in point-value representation

A: \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}

B: \{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}

- Addition:

C: \{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})\}

- Multiplication:
  - Extend A, B to 2n points:
    \{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\}
    \{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\}
  - Product: \{(x_0, y_0 y'_0), (x_1, y_1 y'_1), \ldots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\}
Fast multiplication of polynomials in coefficient form

- **Evaluation**: coefficient representation \(\rightarrow\) point-value representation

- **Interpolation**: point-value representation \(\rightarrow\) coefficient form of a polynomial

My T. Thai
mythai@cise.ufl.edu
Compute evaluation and interpolation

- Evaluation: using Horner method takes $O(n^2)$ => not good
- Interpolation: computing inversion of Vandermonde matrix takes $O(n^3)$ time => not good
- How to complete evaluation and interpolation in $O(n \log n)$ time?
  - Choose evaluation points: complex roots of unity
  - Use Discrete Fourier Transform for evaluation
  - Use inverse Discrete Fourier Transform for interpolation

My T. Thai
mythai@cise.ufl.edu
Complex roots of unity

- A complex nth root of unity is a complex number $\omega$ such that $\omega^n = 1$
- There are exactly $n$ complex nth roots of unity:
  $$e^{2\pi i k/n} \text{ for } k = 0, 1, \ldots, n - 1$$
- Principal nth root of unity $\omega_n = e^{2\pi i / n}$
- All other complex nth roots of unity are powers of $\omega_n$
  $$\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$$
Properties of unity’s Complex roots

Lemma 30.3 (Cancellation lemma)
For any integers $n \geq 0$, $k \geq 0$, and $d > 0$,
\[
\omega_{dn}^{dk} = \omega_{n}^{k}.
\]

Proof:
\[
\omega_{dn}^{dk} = (e^{2\pi i / dn})^{dk} = (e^{2\pi i / n})^{k} = \omega_{n}^{k}.
\]

Corollary 30.4
For any even integer $n > 0$,
\[
\omega_{n}^{n/2} = \omega_{2} = -1.
\]
**Halving lemma**

**Lemma 30.5 (Halving lemma)**
If $n > 0$ is even, then the squares of the $n$ complex $n$th roots of unity are the $n/2$ complex $(n/2)$th roots of unity.

**Proof:**

1. 
   \[ (\omega_n^{k+n/2})^2 = \omega_n^{2k+n} \]
   \[ = \omega_n^{2k} \omega_n^n \]
   \[ = \omega_n^{2k} \]
   \[ = (\omega_n^k)^2 \]

2. **Cancellation lemma:** \[ (\omega_n^k)^2 = \omega_n^{k/2} \]
Summation lemma

Lemma 30.6 (Summation lemma)
For any integer $n \geq 1$ and nonzero integer $k$ not divisible by $n$,
\[
\sum_{j=0}^{n-1} \left( \omega_n^k \right)^j = 0 .
\]

Proof:

Note:
- $k$ is not divisible by $n$
- $\omega_n^k = 1$ only when $k$ is divisible by $n$

\[
\sum_{j=0}^{n-1} \left( \omega_n^k \right)^j = \frac{\left( \omega_n^k \right)^n - 1}{\omega_n^k - 1}
\]

\[
= \frac{\left( \omega_n^k \right)^k - 1}{\omega_n^k - 1}
\]

\[
= \frac{(1)^k - 1}{\omega_n^k - 1}
\]

\[
= 0 .
\]
Discrete Fourier Transform

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

The values of \( A \) at evaluation points \( \omega_0^n, \omega_1^n, \omega_2^n, \ldots, \omega_n^{n-1} \) are

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

Vector \( y = (y_0, y_1, \ldots, y_{n-1}) \) is discrete Fourier transform (DFT) of the coefficient vector \( a = (a_0, a_1, \ldots, a_{n-1}) \). Denote \( y = \text{DFT}_n(a) \)
Fast Fourier Transform

- **Divide** $A$ into two polynomials based on the even-indexed and odd-indexed coefficients:

  \[
  A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}
  \]

  \[
  A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}
  \]

- **Combine:** $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$

- Evaluation points of $A^{[0]}$ and $A^{[1]}$

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

are actually the $n/2$th roots of unity
Fast Fourier Transform

\begin{verbatim}
RECURSIVE-FFT(a)
1    n = a.length  // n is a power of 2
2    if n == 1
3        return a
4    \omega_n = e^{2\pi i / n}
5    \omega = 1
6    a[0] = (a_0, a_2, \ldots, a_{n-2})
7    a[1] = (a_1, a_3, \ldots, a_{n-1})
8    y[0] = RECURSIVE-FFT(a[0])
9    y[1] = RECURSIVE-FFT(a[1])
10    for k = 0 to n/2 - 1
11       y_k = y_k[0] + \omega y_k[1]
12       y_{k+(n/2)} = y_k[0] - \omega y_k[1]
13       \omega = \omega \omega_n
14    return y  // y is assumed to be a column vector
\end{verbatim}

Time: \( T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n) \)
Inversion of Vandermonde matrix

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

**Theorem 30.7**
For \( j, k = 0, 1, \ldots, n - 1 \), the \((j, k)\) entry of \( V_n^{-1} \) is \( \omega_n^{-kj} / n \).

**Proof:**

\[
[V_n^{-1} V_n]_{jj'} = \sum_{k=0}^{n-1} \left( \omega_n^{-kj} / n \right) \left( \omega_n^{kj'} \right) = \sum_{k=0}^{n-1} \omega_n^{k(j'-j)} / n
\]

This summation equals 1 if \( j' = j \), and it is 0 otherwise by the summation lemma.
Interpolation at the complex roots of unity

\[ a = V_n^{-1} y \]

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

- Compute by modifying FFT algorithm
  - Switch the roles of \( a \) and \( y \)
  - Replace \( \omega_n \) by \( \omega_n^{-1} \)
  - Divide each element by \( n \)
Theorem 30.8 (Convolution theorem)
For any two vectors $a$ and $b$ of length $n$, where $n$ is a power of 2,

$$a \otimes b = \text{DFT}_{2n}^{-1} \left( \text{DFT}_{2n}(a) \cdot \text{DFT}_{2n}(b) \right),$$

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

Theorem 30.2
We can multiply two polynomials of degree-bound $n$ in time $\Theta(n \lg n)$, with both the input and output representations in coefficient form.
Efficient FFT implementations

Recursive-FFT($a$)

1. $n = a.length$  \hspace{1cm} // $n$ is a power of 2
2. if $n == 1$
   3. return $a$
4. $\omega_n = e^{2\pi i/n}$
5. $\omega = 1$
6. $a[0] = (a_0, a_2, \ldots, a_{n-2})$
7. $a[1] = (a_1, a_3, \ldots, a_{n-1})$
8. $y[0] = $Recursive-FFT($a[0]$)
10. for $k = 0$ to $n/2 - 1$
   11. $y_k = y_k[0] + \omega y_k[1]$
   12. $y_{k+(n/2)} = y_k[0] - \omega y_k[1]$
   13. $\omega = \omega \cdot \omega_n$
14. return $y$ \hspace{1cm} // $y$ is assumed to be a column vector

- Line 10 – 12: change the loop to compute $\omega_n^k y_k^{[1]}$ only once storing it in $t$ (Butterfly operation)

\[
\begin{align*}
  t &= \omega y_k^{[1]} \\
  y_k &= y_k[0] + t \\
  y_{k+(n/2)} &= y_k[0] - t \\
  \omega &= \omega \cdot \omega_n
\end{align*}
\]

Butterfly operation
Structure of RECURSIVE-FFT

- Each RECURSIVE-FFT invocation makes two recursive calls
- Arrange the elements of a into the order in which they appear in the leaves (take $\log n$ for each element)
- Compute bottom up
An iterative FFT implementation

**Iterative-FFT**(a)

1. **Bit-Reverse-Copy**(a, A)
2. \( n = a\. \text{length} \)  // \( n \) is a power of 2
3. \( \text{for } s = 1 \text{ to } \lg n \)
4. \( m = 2^s \)
5. \( \omega_m = e^{2\pi i / m} \)
6. \( \text{for } k = 0 \text{ to } n - 1 \text{ by } m \)
7. \( \omega = 1 \)
8. \( \text{for } j = 0 \text{ to } m/2 - 1 \)
9. \( t = \omega \ A[k + j + m/2] \)
10. \( u = A[k + j] \)
11. \( A[k + j] = u + t \)
12. \( A[k + j + m/2] = u - t \)
13. \( \omega = \omega \omega_m \)
14. **return** A

**Time:**

\[
L(n) = \sum_{s=1}^{\lg n} \frac{n}{2^s} \cdot 2^{s-1} = \Theta(n \lg n).
\]
A parallel FFT circuit

stage s = 1

stage s = 2

stage s = 3