The Fast Fourier Transform and Applications to Multiplication

Analysis of Algorithms

Prepared by
John Reif, Ph.D.
Topics and Readings:

- The Fast Fourier Transform

  • Reading Selection:
  • CLR, Chapter 30

Advanced Material:

- Using FFT to solve other Multipoint Evaluation Problems

- Applications to Multiplication
Nth Roots of Unity

• Assume Commutative Ring \((R, +, \cdot, 0, 1)\)

• \(\omega\) is principal \(n\)th root of unity if
  
  \(- \omega^k \neq 1 \text{ for } k = 1, \ldots, n-1\)
  
  \(- \omega^n = 1, \text{ and} \)

\[
\sum_{j=0}^{n-1} \omega^{jp} = 0 \text{ for } 1 \leq p \leq n
\]

• Example:
  
  \(\omega = e^{2\pi i/n}\) for complex numbers
Example of $n$th Root of Unity for Complex Numbers

$\omega = e^{2\pi i/8}$ is the 8th root of unity
Fourier Matrix

\[ M_n(\omega) = \begin{pmatrix}
1 & 1 & \ldots & 1 \\
1 & \omega & \ldots & \omega^{n-1} \\
1 & \omega^2 & \ldots & \omega^{2(n-1)} \\
\vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \ldots & \omega^{(n-1)(n-1)}
\end{pmatrix} \]

so \[ M(\omega)_{ij} = \omega^{ij} \text{ for } 0 \leq i, j < n \]

given \( a = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} \)
Discrete Fourier Transform

**Input** a column n-vector \( a = (a_0, ..., a_{n-1})^T \)

**Output** an n-vector which is the product of the Fourier matrix times the input vector

\[
\text{DFT}_n (a) = M(\omega) \times a
\]

\[
\begin{pmatrix}
    f_0 \\
    \vdots \\
    f_{n-1}
\end{pmatrix}
\]

where

\[
f_i = \sum_{k=0}^{n-1} a_k \omega^{ik}
\]
Inverse Fourier Transform

\[
\text{DFT}_n^{-1}(a) = M(\omega)^{-1} \times a
\]

**Theorem**

\[
M(\omega)^{-1}_{ij} = \frac{1}{n} \omega^{-ij}
\]

**proof**

We must show \(M(\omega) \cdot M(\omega)^{-1} = I\)

\[
\frac{1}{n} \sum_{k=0}^{n-1} \omega^{ik} \omega^{-kj} = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{k(i-j)}
\]

\[
= \begin{cases} 
0 & \text{if } i-j \neq 0 \\
1 & \text{if } i-j = 0 
\end{cases}
\]

using identity \(\sum_{k=0}^{n-1} \omega^{kp} = 0, \text{ for } 1 \leq p < n\)
Fourier Transform is Polynomial Evaluation at the Roots of Unity

**Input** a column n-vector \( a = (a_0, ..., a_{n-1})^T \)

**Output** an n-vector \( (f_0, ..., f_{n-1})^T \) which are the values polynomial \( f(x) \) at the \( n \) roots of unity

\[
\text{DFT}_n(a) = \begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix}
\]

where

\[
f_i = f(\omega^i) \quad \text{and}
\]

\[
f(x) = \sum_{j=0}^{n-1} a_j \cdot x^j
\]
Fast Fourier Transform

- Viewed as **Evaluation Problem**: naïve algorithm takes $n^2$ ops
- **Divide and Conquer** gives FFT with $O(n \log n)$ ops for $n$ a power of 2

**Key Idea:**
- If $\omega$ is $n$th root of unity then $\omega^2$ is $n/2$th root of unity
- So can reduce the problem to two subproblems of size $n/2$
Algorithm $\text{FFT}_n$

- Input $a = (a_0, ..., a_{n-1})^T$, $n$ a power of 2

1. If $n=1$ then output

2. $\left( f'_0, ..., f'_{\frac{n}{2}-1} \right)^T \leftarrow \text{FFT}_n \left( (a_0, a_2, ..., a_{n-2})^T \right)$

3. $\left( f''_0, ..., f''_{\frac{n}{2}-1} \right)^T \leftarrow \text{FFT}_n \left( (a_1, a_3, ..., a_{n-1})^T \right)$

4. For $i=0, ..., \frac{n}{2} - 1$ do

   $f_i \leftarrow f'_i + \omega^i f''_i$

   $f_{i+\frac{n}{2}} \leftarrow f'_i - \omega^i f''_i$

5. Output $(f_0, f_1, ..., f_{n-1})$
FFT Circuit
(also known as Butterfly Network)

- Total Recursion depth = $\log n$
- Communication Distance $2^d$ at depth $d$
\( f_i = a_0 + a_1 \omega^i + a_2 \omega^{2i} + \ldots + a_{n-1} \omega^{(n-1)i} \)

\( f_i = f'_i + \omega^i f''_i \) where

\( f'_i = a_0 + a_2 (\omega^2)^i + a_4 (\omega^2)^{2i} + \ldots + a_{n-2} (\omega^2)^{i(n-2)} \)

\( f''_i = a_1 + a_3 (\omega^2)^i + \ldots + a_{n-1} (\omega^2)^{i(n-2)} \)

\[
\begin{bmatrix}
    f'_0 \\
    f'_{\frac{n}{2}} \\
    \vdots \\
    f'_{\frac{n-1}{2}}
\end{bmatrix} = M \frac{n}{2} (\omega^2) \begin{bmatrix}
    a_0 \\
    a_2 \\
    M \\
    a_{n-2}
\end{bmatrix} = DFT_n \left( (a_0, a_2, \ldots, a_{n-2})^T \right)
\]

\[
\begin{bmatrix}
    f''_0 \\
    f''_{\frac{n}{2}} \\
    \vdots \\
    f''_{\frac{n-1}{2}}
\end{bmatrix} = M \frac{n}{2} (\omega^2) \begin{bmatrix}
    a_1 \\
    a_3 \\
    M \\
    a_{n-1}
\end{bmatrix} = DFT_n \left( (a_1, a_3, \ldots, a_{n-1})^T \right)
\]
Note: $f_{\frac{n}{2}+1}^{i} = f_{i}^{i}$, $f_{\frac{n}{2}+1}^{i+1} = f_{i}^{i}$, $i=0, ..., \frac{n}{2} - 1$

But $\omega^{n}=1$, so $(\omega^{2})^{\frac{n}{2}+1} = \omega^{n} \cdot (\omega^{2})^{i} = \omega^{2i}$

for $i=0, ..., \frac{n}{2} - 1$

Thus, $f_{i} = f_{i}^{i} + \omega^{i} f_{i}^{i}$ for $i=0, ..., \frac{n}{2} - 1$

and $f_{i+\frac{n}{2}} = f_{i}^{i} + \omega^{\frac{i+n}{2}} \cdot f_{i}^{i}$

$= f_{i}^{i} - \omega^{i} f_{i}^{i}$ for $i=0, ..., \frac{n}{2} - 1$

since $(\omega^{2})^{2} = \omega^{n} = 1$, so $\omega^{2} = -1$
Operation Counts for FFT Algorithm

- Assume $n = 2^k$
- # additions
  \[ \text{Add}(n) = 2 \cdot \text{Add}(n/2) + n \]
  \[ = n \log n \]
- # multiplications
  \[ \text{Mult}(n) = 2 \cdot \text{Mult}(n/2) + n/2 \]
  \[ = \frac{1}{2} n \log n \]
- Total Time \( O(n \log n) \)
  - Note in complex FFT,
    # real ops is \( 5 \ n \log n \)
Multipoint Polynomial Evaluation

- **Input** polynomial

- **Problem** evaluate $f(x)$ at $x_0, x_1, \ldots, x_{n-1}$

- **Easy Cases**: eval at roots of unity

\[
f(x) = \sum_{i=0}^{n-1} a_i x^i
\]

FFT Case $x_k = \omega^k$ for $k=0,\ldots,n$
Multipoint Polynomial Evaluation (cont’ d)

Summary of FFT:

method \[ f(x) = f'(y) + x f''(y) \]

where \( y = x^2 \)

\( f'(x), f''(x) \) both degree halved

⇒ needed to only evaluate at half as many points
Other Polynomial Evaluation Problems Solved by FFT

Each costs $O(n \log n)$ time

- Evaluate at points $X_i = ba^i + d$ for $i=0,\ldots, n-1$ (Chirp Transform)
  - Reduced to FFT

- Single point evaluation of all derivatives of a polynomial
  - Solve by reduction to above Chirp Transform of case 2)

- Evaluate at points $X_i = b(a^i)^2 + ca^i + d$ for $i=0,\ldots, n-1$
  - Solve by divide and conquer similar to FFT
Single Point Evaluation of all Derivatives of Polynomial

- **Input**
  \[ f(x) = \sum_{i=0}^{n-1} a_i x^i \]
  and point \( x_0 \)

- **Output**
  \[ f^k(x) = \frac{d^k f(x)}{dx} \quad x = x_0 \text{ for } k = 0, \ldots, n - 1 \]
Single Point Evaluation of all Derivatives of Polynomial (cont’d)

- Taylor Series Representation of

\[ f(x) = \sum_{i=0}^{n-1} c_i (x - x_0)^i \]

Then

\[ f^{(k)}(x_0) = k! c_k \]

This reduces to case of evaluation at points

\[ x_i = ab^i \text{ for } i = 0, \ldots, n-1 \]

- Solve this Chirp Transform problem by reduction FFT
Advanced Material:
Further Applications of FFT

1) **Convolution**: Products and Powers of Polynomials
   - Used for Integer Multiplication Algorithms
   - Also used for **Filtering** on infinite input streams

2) Division and Inverse of Polynomials

3) Multipoint Evaluation and Interpolation
Advanced Material: Products and Powers of Polynomials

• Input vectors
  \[ a = (a_0, a_1, \ldots, a_{n-1})^T \]
  \[ b = (b_0, b_1, \ldots, b_{n-1})^T \]

• Definition of Convolution \[ c = a \otimes b \]

Where

\[ c_i = \sum_{j=0}^{n-1} a_j b_{i-j} \]

for \( i = 0, \ldots, 2n-1 \)

define \( a_k = b_k = 0 \) if \( k < 0 \) or \( k \geq n \)
Products and Powers of Polynomials (cont’ d)

- Convolution Theorem

\[ a \otimes b = \text{FFT}_{2n}^{-1} \left( \text{FFT}_{2n} (a) \cdot \text{FFT}_{2n} (b) \right) \]

- Application to Polynomial Products:

\[
p(x) = \sum_{i=0}^{n-1} a_i x^i \\
q(x) = \sum_{j=0}^{n-1} b_j x^j \\
p(x) \cdot q(x) = \sum_{i=0}^{2n-2} c_i x^i \text{ where } c_i = \sum_{j=0}^{n-1} a_j b_{i-j}
\]
Products of $m$ Polynomials

for $k=1, \ldots, m$ let $P_k(x) = \sum_{i=0}^{n-1} a_{k,i} x^i$

\[
\prod_{k=1}^{m} P_k(x) = \sum_{i=0}^{m(n-1)} c_i x^i, \text{ where } c_i = \sum_{j_1=1}^{m} \prod_{k=1}^{m} a_{k,j_k} \sum j_k = 1
\]

- Generalized Convolution Theorem

\[
a_1 \otimes a_2 \otimes \ldots \otimes a_m = \text{FFT}_{nm}^{-1}\left(\text{FFT}_{nm}(a_1) \cdot \text{FFT}_{nm}(a_2) \cdot \ldots \cdot \text{FFT}_{nm}(a_m)\right)
\]
Wrapped Convolutions

- \( a = (a_0, a_1, ..., a_{n-1})^T, \ b = (b_0, b_1, ..., b_{n-1})^T \)
- Positive wrapped convolution is
  \( c = (c_0, c_1, ..., c_{n-1})^T \)
  \[
  c_i = \sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{n-1} a_j b_{n+i-j}
  \]
- Negative wrapped convolution is
  \( d = (d_0, d_1, ..., d_{n-1})^T \)
  \[
  d_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}
  \]
Application of Wrapped Convolution to Modular Polynomial Products

\[ p(x) = \sum_{i=0}^{n-1} a_i x^i \]

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

\[ p(x) \cdot q(x) \mod (x^n + 1) = \sum_{i=0}^{n-1} d_i x^i \text{ since } x^n = -1 \mod (x^n + 1) \]
Computing Positive Wrapped Convolution

- Let $\omega$ = principal $n$th root of unity
- Assume $n$ has multiplicative inverse,

**Theorem**

$$c = \text{FFT}^{-1}_n \left( \text{FFT}_n(a) \cdot \text{FFT}_n(b) \right)$$

is the positive wrapped convolution of $n$-vectors $a$ and $b$. 
Computing Negative Wrapped Convolution

- Also

\[ \hat{d} = \text{FFT}_n^{-1} \left( \text{FFT}_n(\hat{a}) \cdot \text{FFT}_n(\hat{b}) \right) \]

is the negatively wrapped convolution of n-vectors \( \hat{a} \) and \( \hat{b} \)

where

\[ \hat{a} = (a_0, \Psi a_1, ..., \Psi^{n-1} a_{n-1})^T \]

\[ \hat{b} = (b_0, \Psi b_1, ..., \Psi^{n-1} b_{n-1})^T \]

and \( \Psi^2 = \omega = \text{principal nth root of unity} \)
Integer Multiplication by Polynomial Product (solved via FFT)

- Input n bit integers $a, b$
define polynomials degree $k = n/L$

\[
a(x) = \sum_{i=0}^{k-1} a_i x^i, \quad 0 \leq a_i \leq 2^L
\]

\[
b(x) = \sum_{i=0}^{k-1} b_i x^i, \quad 0 \leq b_i \leq 2^L
\]

so $a = a(2^L)$, $b = b(2^L)$
Integer Multiplication by Polynomial Product (cont’d)

• Idea
  1) Compute $c(x) = a(x) \cdot b(x)$ by convolution

  2) Evaluate $c(2^L) = a \cdot b$
Integer Multiplication Algorithms using Reduction to Polynomial Product

- Pollard Mult Algorithm
  \[ O(n(\log n)^2)(\log \log n)^\epsilon \] use \[ L = \log n \]

- Karp Mult Algorithm
  \[ O(n(\log n)^2) \] use \[ L = \sqrt{n} \]

- Schönage-Strassen Mult Algorithm
  \[ O(n(\log n)(\log \log n)) \] use \[ L = \sqrt{n} \]
  and wrapped convolution
Pollard Multiplication Algorithm

- \( n = kL, \ L = 1 + \log k \)

1) Choose primes \( P_1, P_2, P_3 \) where

\[
P_1 \cdot P_2 \cdot P_3 \geq 4 \cdot k^3
\]

and \( P_i = \alpha_i \cdot 2^L + 1, \alpha_i = O(1) \)

2) Compute \( C(x) \) by convolution over finite field \( \mathbb{Z}_{p_i} \) for \( i = 1, 2, 3 \)

(Requires \( k \) mults on \( 2L \) bit integers)
Pollard Multiplication Algorithm (cont’ d)

3) Evaluate $C(2^L)$

- Time Bounds

\[
T(n) = 3kT(2L) + O(k \log k) \cdot O(L)
\]
\[
= 3kT \left(2(1 + \log k)\right) + O(k(\log k)^2)
\]
\[
\leq O(n(\log n)^2 (\log \log n)^\epsilon) \text{ for any } \epsilon > 0
\]
Korp Multiplication Algorithm

1) Compute $C(x)$ modulo $k$ by convolution
2) Compute $C(x)$ modulo $(2^{2L}+1)$ by convolution
3) Compute $C(x)$ coefficients from $C(x)$ mod $k$, $C(x)$ mod $(2^{2L}+1)$ by Chinese remaindering

$n = 2^s = kL$

$$k = \begin{cases} 
2^{\frac{s}{2}} & \text{if } s \text{ even} \\
2^{\frac{(s-1)}{2}} & \text{else}
\end{cases}$$
Korp Multiplication Algorithm (cont’d)

4) Compute $C(2^L)$

- **Time**

\[
T(n) = 2kT(2L) + O(k \log k)O(L)
\]

\[
= 2\sqrt{n}T (2\sqrt{n}) + O(n \log n)
\]

\[
= O(n(\log n)^2)
\]
Schönage-Strassen Multiplication Algorithm

(2’ ) Compute $C(x)$ mod $(x^k+1)$ modulo $(2^{2L}+1)$ by wrapped convolution
Requires only $k$ recursive mults on $2L$ bit numbers

- Time

\[
T(n) = kT(2L) + O(k \log k)O(L) \\
= \sqrt{n}T (2\sqrt{n}) + O(n \log n) \\
= O(n \log n)(\log \log n)
\]
Still Open Problem: How Fast Can You Multiply Integers?

- Can you mult n bit integers in $O(n \log n)$ time?
The Fast Fourier Transform and Applications to Multiplication

Analysis of Algorithms

Prepared by
John Reif, Ph.D.