Algorithm Design and Analysis

Victor Adamchik — CS 15-451 — Spring 2014

Lecture 4 — Jan 22, 2014 — Carnegie Mellon University

# Fast Fourier Transform - II

Gauss
(1777 – 1855)

Lagrange
(1736 –1813)

Fourier
(1768 –1830)

## Applications

Signal Processing
Image Compression
Statistics, Finance
Pattern Matching

## History

Cooley and Tukey's paper 1965
It was known to Gauss, 1805.

Tukey derived the basic reduction while in a meeting of President Kennedy's Science Advisory Committee for off-shore detection of nuclear tests in the Soviet Union.

The idea was to analyze time series obtained from seismometers. Other possible applications to national security included the long-range acoustic detection of nuclear submarines.

## High Level Idea

To compute the product A(x)B(x) of polynomials

O(n log n)

1) evaluate A(x) and B(x) at roots of unity, using the Vandermonde matrix
2) multiply $A(x_k)B(x_k)$,

O(n)

3) then find the polynomial using Lagrange's interpolation via the Vandermonde matrix

O(n log n)

Lagrange's Interpolation formula can be represented via the Vandermonde Matrix.

Polynomial evaluation is also computed via the Vandermonde Matrix.

## The Lagrange Interpolation

The Lagrange formula defines a polynomial

$$A(x) = \sum_{k=0}^{n-1} a_k x^k$$

where coefficients $a_k$ can be found by solving this

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix}$$

or in short $V.a = y$ Thus, $a = V^{-1}.y$

## Complexity of Interpolation

$$\begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix}$$

It follows that the complexity of interpolation depends on how fast can we inverse the Vandermonde matrix.

## Determinant of the Vandermonde Matrix

$$\det \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} = \prod_{k=0}^{n-1} \prod_{j=0}^{k-1} (x_k - x_j)$$

Since the n points are distinct, the determinant can't be zero.

The proof (by induction on n) is left as an exercise to a reader ☺

## Computing Polynomials

$$A(x) = \sum_{k=0}^{n-1} a_k x^k$$

$$\begin{pmatrix} A(1) \\ A(w) \\ \dots \\ A(w^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & w^{n-1} & w^{2(n-1)} & \dots & w^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix}$$

We will prove that this matrix multiplication can be done in O(n log n)

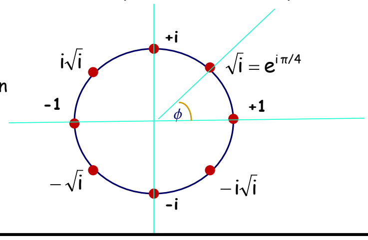
## Roots of Unity

are solutions to $z^n = 1$

The n-th roots of unity are points on the complex unit circle every $2\pi/n$ radians apart

$$z = e^{i\,2\pi k/n}$$

k=0,1,…,n-1




## Primitive Roots of Unity

<u>Definition:</u> A complex number w is called a n-th primitive root of unity if

1) $w^n = 1$

2) $w^p \neq 1$, for p = 1, .2, …, n-1

## Roots of Unity

<u>Claim 1:</u> Let w be a primitive root of $z^n = 1$ then

$$\sum_{k=0}^{n-1} w^k = 0$$

Proof. Multiply it by w

$$w\sum_{k=0}^{n-1} w^k = w(1 + w + \dots + w^{n-1}) =$$

$$= w + w^2 + \dots + w^{n-1} + w^n = \sum_{k=0}^{n-1} w^k$$

## Roots of Unity

Claim 2: Let w be a primitive root of $z^n = 1$ and $p = 1, \ldots, n-1$ then

$$\sum_{k=0}^{n-1} w^{kp} = 0$$

Proof.

$$\sum_{k=0}^{n-1} w^{kp} = \sum_{k=0}^{n-1} (w^p)^k = \frac{(w^p)^n - 1}{w^p - 1} = \frac{1^p - 1}{w^p - 1} = 0$$

## Modular Arithmetic

Consider a set of powers of 2

1,2,4,8,16,32,64,128

modulo 17

1,2,4,8,-1,-2,-4,-8

Square and then do mod 17 again

{1,2,4,8}^2 = {1, 4, 16, 64} ={1,4,-1,-4}

## Computing Polynomials

$$A(x) = \sum_{k=0}^{n-1} a_k x^k$$

$$\begin{pmatrix} A(1) \\ A(w) \\ \ldots \\ A(w^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{pmatrix}$$

Consider n = 4

## Computing Polynomials, n = 4

$$V_4 \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \boxed{\begin{matrix} 1 & 1 \\ 1 & -1 \end{matrix}} & \begin{matrix} 1 & 1 \\ i & -i \end{matrix} \\ \boxed{\begin{matrix} 1 & 1 \\ 1 & -1 \end{matrix}} & \begin{matrix} -1 & -1 \\ -i & i \end{matrix} \end{pmatrix} \begin{pmatrix} a_0 \\ a_2 \\ a_1 \\ a_3 \end{pmatrix} \begin{matrix} \text{even} \\ \\ \text{odd} \end{matrix}$$

swap

$V_2$

$$= \begin{pmatrix} V_2 & \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} V_2 \\ V_2 & -\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} V_2 \end{pmatrix}$$

## Computing Polynomials

$$V_{2n} = \begin{pmatrix} V_n & D_n V_n \\ V_n & -D_n V_n \end{pmatrix} \begin{pmatrix} a_{even} \\ a_{odd} \end{pmatrix} = \begin{pmatrix} V_n a_{even} + D_n V_n a_{odd} \\ V_n a_{even} - D_n V_n a_{odd} \end{pmatrix}$$

where $D_n$ is a diagonal matrix

$$D_n = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & w & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & w^{n-1} \end{pmatrix}.$$

## Proof

$$A(x) = \sum_{k=0}^{n-1} a_k x^k$$

$$\begin{pmatrix} A(1) \\ A(w) \\ \ldots \\ A(w^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & \ldots & w^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w^{n-1} & w^{2(n-1)} & \ldots & w^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{pmatrix}$$

Consider j-th row

$$A(w^j) = \sum_{k=0}^{n-1} w^{jk} a_k \qquad = \sum_{k\,\text{is even}} + \sum_{k\,\text{is odd}}$$

## Computing Polynomials

$$A(w^j) = \sum_{k=0}^{n/2-1} w^{j2k} a_{2k} + \sum_{k=0}^{n/2-1} w^{j(2k+1)} a_{2k+1}$$

Let $w_n$ denote a root of $z^n = 1$.

Since $w_n^2 = w_{n/2}$, (it follows from $(z^2)^{n/2} = z^n$

$$A(w^j) = \sum_{k=0}^{n/2-1} w_{n/2}^{jk} a_{2k} + w_n^j \sum_{k=0}^{n/2-1} w_{n/2}^{jk} a_{2k+1} = F_1(j) + w_n^j F_2(j$$

here $F_j$ is a n/2 size problem.

## Computing Polynomials

$$A(w^j) = \sum_{k=0}^{n/2-1} w_{n/2}^{jk} a_{2k} + w_n^j \sum_{k=0}^{n/2-1} w_{n/2}^{jk} a_{2k+1}$$

Let us compute $A(w^{j+n/2})$

$$A(w^{j+n/2}) = \sum_{k=0}^{n/2-1} w_{n/2}^{(j+n/2)k} a_{2k} + w_n^{j+n/2} \sum_{k=0}^{n/2-1} w_{n/2}^{(j+n/2)k} a_{2k+1}$$

Observe $w_{n/2}^{j+n/2} = w_{n/2}^j$ and $w_n^{n/2} = -1$ for even n

Periodic property

Symmetry property

## Computing Polynomials

$$A(w^j) = F_1(j) + w_n^j F_2(j),\ j = 0, 1, \ldots, n/2-1$$

$$A(w^{j+n/2}) = F_1(j) - w_n^j F_2(j),\ j = 0, 1, \ldots, n/2-1$$

This outlines the divide and conquer algorithm.

Therefore, V.a can be computed in O(n log n)

## Complexity of Interpolation

$$\begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \ldots \\ y_{n-1} \end{pmatrix}$$

We know that the complexity of interpolation depends on how fast can we inverse the Vandermonde matrix.

We will show that this step is also O(n log n)

## Inverse Vandermonde

Theorem.

$$V^{-1}(w) = \frac{1}{n} V(\frac{1}{w})$$

where $w^n = 1$.

## Inverse Vandermonde

$$V(w) = \begin{pmatrix} 1 & w_0 & w_0^2 & \ldots & w_0^{n-1} \\ 1 & w_1 & w_1^2 & \ldots & w_1^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w_{n-1} & w_{n-1}^2 & \ldots & w_{n-1}^{n-1} \end{pmatrix}$$

Let V* be V where w -> 1/w.

Compute V*.V   Each element of the product is

$$\{1, w^{-i}, w^{-2i}, \ldots, w^{-i(n-1)}\} . \{1, w^j, w^{2j}, \ldots, w^{j(n-1)}\}$$

$$= 1 + w^{j-i} + w^{2(j-i)} + \ldots + w^{(n-1)(j-i)}$$

## Inverse Vandermonde

$$(V^*.V)(i,j) = 1 + w^{j-i} + w^{2(j-i)} + \ldots + w^{(n-1)(j-i)}$$

Recall

$$1 + x + x^2 + \ldots + x^{n-1} = \sum_{k=0}^{n-1} x^k = \frac{1-x^n}{1-x}$$

It follows, that

$$(V^*.V)(i,i) = n$$

$$(V^*.V)(i,j \neq i) = 0$$

V*.V = n I

V(1/w).V(w) = n I

## Polynomial multiplication

$$a_0, a_1, \ldots, a_{n-1}$$
$$b_0, b_1, \ldots, b_{n-1}$$

FFT

$$A(1), A(w), A(w^2), \ldots, A(w^{2n-1})$$
$$B(1), B(w), B(w^2), \ldots, B(w^{2n-1})$$

Point-value multiplication

$$A(1)B(1), A(w)B(w), \ldots, A(w^{2n-1})B(w^{2n-1})$$

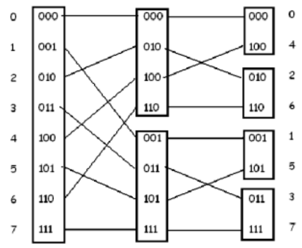
Inverse FFT

$$C(x) = \sum_{k=0}^{2n-2} c_k x^k$$

## FFT in place

Let n = 8

The right column is a bit reversal!.




The recursive algorithm can simply call on the left and right halves, rather than on the odd and even indices.

All DSP processors include a hardware bit reversal capability

## Discrete Fourier Transform

DFT converts a set of sample points into another set ordered by frequencies. It reveals periodicities in input data.

A DFT of $\{a_0, a_1, \ldots, a_{n-1}\}$ is defined by

$$b_j = \sum_{k=0}^{n-1} a_k w^{k\,j}$$

where $w^n = 1$. In a matrix form V.a = b

FFT is an algorithm for computing DFT.

## Convolution

The convolution of two vectors $a_k$ and $b_k$ is a third vector $c = a \otimes b$ which represents an overlap between the two vectors.

$$c_j = \sum_{k=0}^{n-1} a_k b_{j-k}$$

<u>The Convolution Theorem</u> says that the DFT of a convolution of two vectors is the point-wise product of the DFT of the two vectors

$$DFT(a \oplus b) = DFT(a)\,DFT(b)$$

## Convolution

$$DFT(a \oplus b) = DFT(a)\,DFT(b)$$

$$a \oplus b = DFT^{-1}(DFT(a)\,DFT(b))$$

It follows, using FFT we can compute convolution in O(n log n).

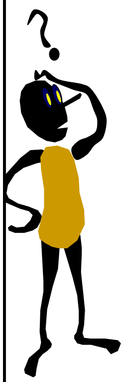
Note that inverse DFT is just a regular DFT with w replaced by $w^{-1}$.

## Polynomial multiplication

$$A(x) = \sum_{k=0}^{n-1} a_k x^k \qquad B(x) = \sum_{k=0}^{n-1} b_k x^k$$

$$A(x)B(x) = \sum_{k=0}^{2n-2} c_k x^k \qquad c_k = \sum_{j=0}^{k} a_j b_{k-j}$$

this is just a convolution of two vectors a and b



Given two n-bit integers

$a = a_{n-1} \ldots a_0$

$b = b_{n-1} \ldots b_0$

compute their product

$A(x) = a_0 + \ldots + a_{n-1} x^{n-1}$

$B(x) = b_0 + \ldots + b_{n-1} x^{n-1}$

$a = A(2), \; b = B(2)$

Compute $C(x) = A(x)\, B(x)$

Evaluate $C(2)$