2.6 The Fast Fourier Transform

Algorithms (S.Dasgupta, C.H.Papadimitriou, U.V.Vazirani)

Natalia Kotsani

Algorithms and Complexity I, MPLA

January 16, 2014

Table of contents

History of the FFT

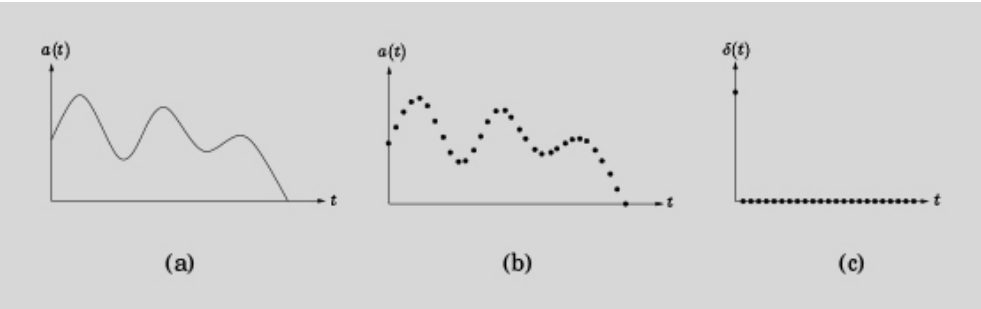
- 1965, publication (J.W. Culey, J.W. Tukey)
- 1963, presentation at IBM (J.W. Tukey)
- late 1930s, usage for hand calculations
- early 1800s, paper on interpolation, in Latin (C.F.Gauss)

J.W Tukey was reluctant to publish FFT because he felt that this was a simple observation that was probably already known. Typical of the period: algorithms were considered second class mathematical objects

Why multiplying polynomials?

Multiplying polynomials is crucial for signal processing.

A signal is any quantity that is a function of time (capture a human voice by measuring fluctuations in air pressure, the pattern of stars in the night sky, by measuring brightness as a function of angle etc.).



In order to extract information from a signal, we need to first digitize it by sampling and, then, to put it through a system that will transform it. The output is called the response of the system:

Why multiplying polynomials?

An important class of systems are those that are *linear*—the response to the sum of two signals is just the sum of their individual responses—and *time invariant*—shifting the input signal by time t produces the same output, also shifted by t. Any system with these properties is completely characterized by its response to the simplest possible input signal: the *unit impulse* $\delta(t)$, consisting solely of a "jerk" at $t = 0$ (Figure (c)). To see this, first consider the close relative $\delta(t - i)$, a shifted impulse in which the jerk occurs at time i. Any signal $a(t)$ can be expressed as a linear combination of these, letting $\delta(t - i)$ pick out its behavior at time i,

$$a(t) = \sum_{i=0}^{T-1} a(i)\delta(t - i)$$

(if the signal consists of T samples). By linearity, the system response to input $a(t)$ is determined by the responses to the various $\delta(t - i)$. And by time invariance, these are in turn just shifted copies of the *impulse response* $b(t)$, the response to $\delta(t)$.

In other words, the output of the system at time k is

$$c(k) = \sum_{i=0}^{k} a(i)b(k - i),$$

exactly the formula for polynomial multiplication!

Multiplying polynomials: Brute-force

Divide-and-conquer: fast algorithms for multiplying integers and matrices; next target is polynomials.

$$(a_0+a_1x+\ldots+a_dx^d)\cdot(b_0+b_1x+\ldots+b_dx^d)=c_0+c_1x+\ldots+c_{2d}x^{2d}$$
$$A(x)\cdot B(x)=C(x)$$

$$(1+2x+3x^2)\cdot(2+x+4x)=2+5x+12x^2+11x^3+12x^4$$
$$c_0=a_0b_0=2$$
$$c_1=a_0b_1+a_1b_0=5$$
$$c_2=a_0b_2+a_1b_1+a_2b_0=12$$

$$c_k=a_0b_k+a_1b_{k-1}+\ldots+a_kb_0=\sum_{i=0}^{k}a_ib_{k-1}$$

(convolution of the input vectors a and b, denoted $c=a\otimes b$)

Complexity: $\Theta(d^2)$

An important property

Fact

A degree-d polynomial is uniquely characterized by its values at any $d + 1$ distinct points.

We can specify a degree-d polynomial $A(x) = a_0 + a_1 x + ... + a_d x_d$ by either one of the following:

- Its coefficients $a_0, a_1, ..., a_d$
- The values $A(x_0), A(x_1), ..., A(x_d)$

The second is the more attractive for polynomial multiplication!

$$C(x) = A(x) \cdot B(x) \Rightarrow C(x_k) = A(x_k) \cdot B(x_k), \text{ for any point } x_k$$

Extended representations

We must face the problem, however, that
$degree(C) = degree(A) + degree(B)$

- if A and B are of degree-bound d, then C is of degree-bound 2d

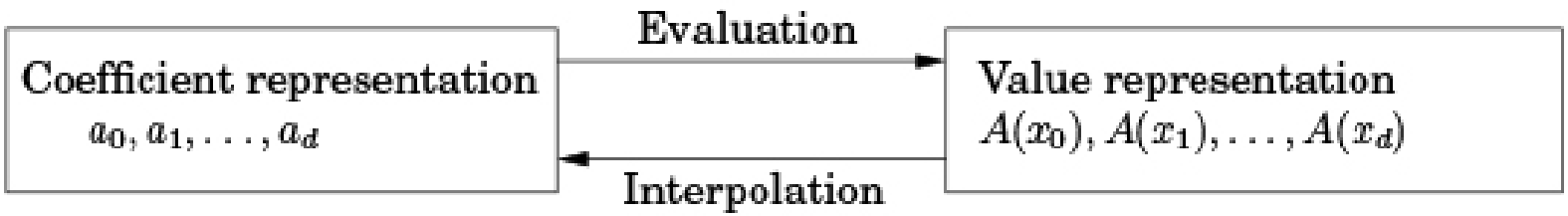
We must therefore begin with extended point-value representations for A and for B consisting of 2d point-value pairs each.

Complexity: $\Theta(d)$

Sketch of the FFT

We expect the input polynomials, and also their product, to be specified by coefficients. So we need:

- evaluation: translate from coefficients to values, which is just a matter of evaluating the polynomial at the chosen points,
- multiplication: in the value representation,
- interpolation: translate back to coefficients.

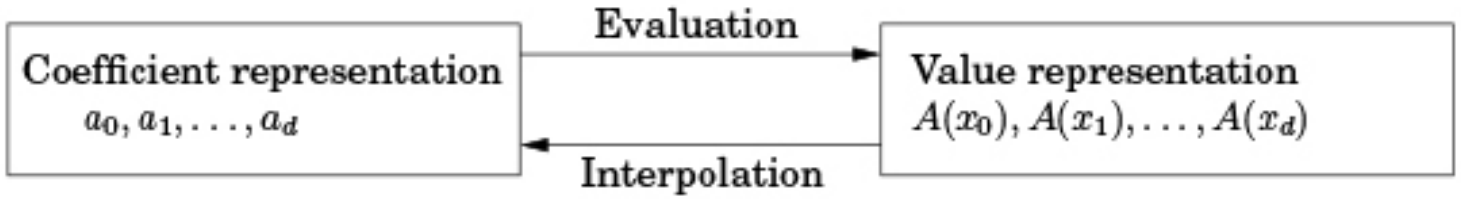


Sketch of the FFT

Figure 2.5 Polynomial multiplication

Input: Coefficients of two polynomials, $A(x)$ and $B(x)$, of degree d
Output: Their product $C = A \cdot B$

Selection
Pick some points $x_0, x_1, \ldots, x_{n-1}$, where $n \geq 2d+1$
Evaluation
Compute $A(x_0), A(x_1), \ldots, A(x_{n-1})$ and $B(x_0), B(x_1), \ldots, B(x_{n-1})$
Multiplication
Compute $C(x_k) = A(x_k)B(x_k)$ for all $k = 0, \ldots, n-1$
Interpolation
Recover $C(x) = c_0 + c_1 x + \cdots + c_{2d} x^{2d}$



Evaluation

Theorem (Uniqueness of an interpolating polynomial)

For any set $\{(x_0, y_0), (x_1, y_1)..., (x_n, y_n)\}$ 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 $k = 0, ..., n-1$.

Proof The proof relies on the existence of the inverse of a certain matrix. Equation (30.3) is equivalent to the matrix equation

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} . \tag{30.4}$$

The matrix on the left is denoted $V(x_0, x_1, \ldots, x_{n-1})$ and is known as a Vandermonde matrix. By Problem D-1, this matrix has determinant

$$\prod_{0 \le j < k \le n-1} (x_k - x_j) ,$$

and therefore, by Theorem D.5, it is invertible (that is, nonsingular) if the x_k are distinct. Thus, we can solve for the coefficients a_j uniquely given the point-value representation:

$$a = V(x_0, x_1, \ldots, x_{n-1})^{-1} y .$$

■

Alexandre-Théophile Vandermonde (1735-1796)

A.T.Vandermonde was a French musician, mathematician and chemist who worked with Bézout and Lavoisier; his name is now principally associated with determinant theory in mathematics. He was born in Paris, and died there.

Vandermonde was a violinist, and became engaged with mathematics only around 1770. In Mémoire sur la résolution des équations (1771) he reported on symmetric functions and solution of cyclotomic polynomials. In Remarques sur des problemes de situation (1771) he studied knight's tours: "Whatever the twists and turns of a system of threads in space, one can always obtain an expression for the calculation of its dimensions, but this expression will be of little use in practice. The craftsman who fashions a braid, a net, or some knots will be concerned, not with questions of measurement, but with those of position: what he sees there is the manner in which the theads are interlaced".

The same year he was elected to the French Academy of Sciences. Mémoire sur des irrationnelles de différents ordres avec une application au cercle (1772) was on combinatorics, and Mémoire sur l'élimination (1772) on the foundations of determinant theory. These papers were presented to the Académie des Sciences, and constitute all his published mathematical work.

Point value representation

Computing a point-value representation for a polynomial given in coefficient form is in principle straightforward:

- select n distinct points $x_0, x_1, ..., x_n$
- evaluate $A(x_k)$ for $k = 0, 1, ..., n$

Horners method

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

Complexity: $O(n^2)$.

Picking the n points

Computing a point-value representation for a polynomial given in coefficient form is in principle straightforward:

- select n distinct points $x_0, x_1, ..., x_n$
- evaluate $A(x_k)$ for $k = 0, 1, ..., n$

Horners method

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

Horners method Complexity: $\Theta(n)$
Evaluation Complexity: $\Theta(n^2)$

IF we choose the points x_k cleverly, we can accelerate this computation to run in time $\Theta(nlgn)$!

Picking the n points

Heres an idea for how to pick the n points at which to evaluate a polynomial $A(x)$ of degree $n-1$: positive-negative pairs

$$\pm x_1, \pm x_2, \ldots, \pm x_{n/2-1}$$

the even powers of x_j coincide with those of x_j (overlapping).

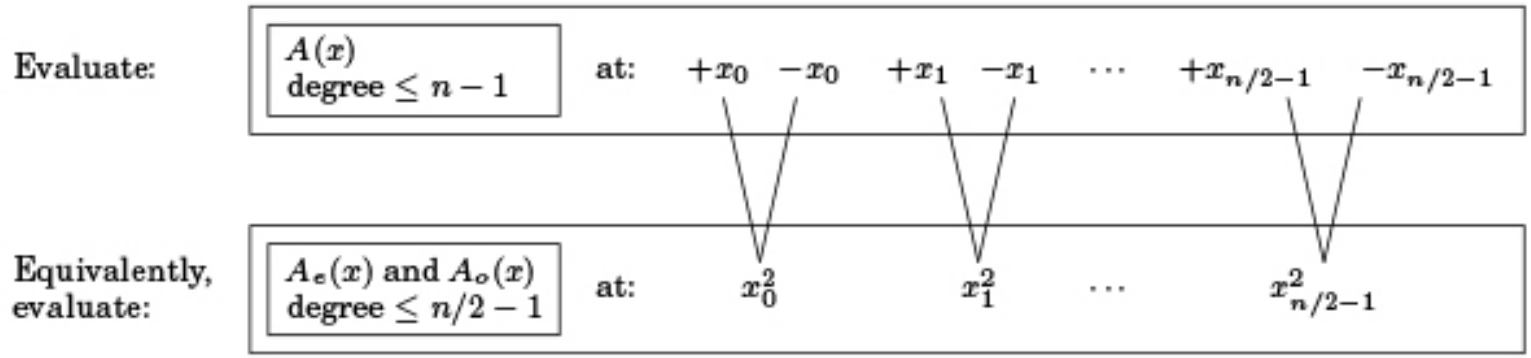
- $3 + 4x + 6x^2 + 2x^3 + x^4 + 10x^5 = (3 + 6x^2 + x^4) + x(4 + 2x^2 + 10x^4)$

$$A(x) = A_e(x^2) + xA_o(x^2)$$

- $A_e(\cdot)$ polynomial with the even-numbered coefficients (degree $\leq n/2 - 1$)
- $A_o(\cdot)$ with the odd-numbered coefficients (degree $\leq n/2 - 1$)
- the terms polynomial in parentheses are polynomials in x^2

evaluating A(x) at n paired points $\pm x_1, \pm x_2, \ldots, \pm x_{n/2-1}$ reduces to evaluating $A_e(x)$ and $A_o(x)$ (which each have half the degree of $A(x)$) at $n/2$ points, $x_0^2, \ldots, x_{n/2-1}^2$

The recursion



The original problem of size n is in this way recast as two subproblems of size n/2, followed by some linear-time arithmetic.

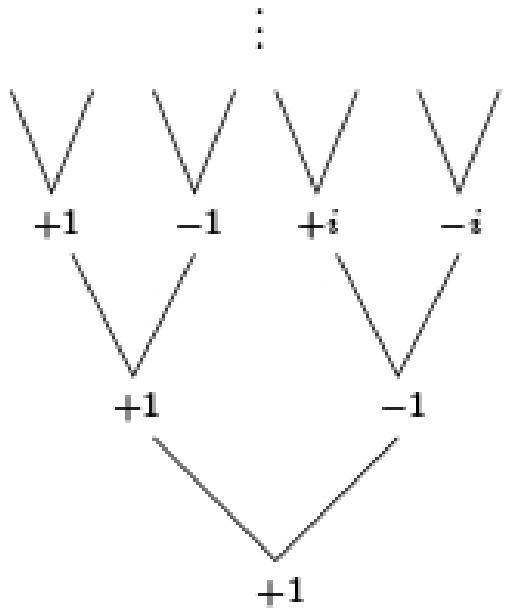
$$T(n) = 2T(\tfrac{n}{2}) + O(n)$$

Complexity: $O(nlogn)$

The recursion: a problem

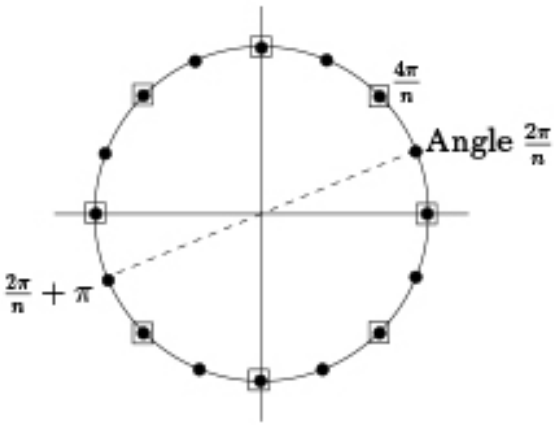
The plus-minus trick only works at the top level of the recursion!
How can a square be negative?

The reverse engineering of the process



The complex n_{th} roots of unity

- the n complex solutions to the equation $z_n = 1$
- the complex numbers $1, \omega, \omega^2, ..., \omega^{n-1}$ where $\omega = e^{2pi/n}$
- the n_{th} roots are plus-minus paired: $\omega^{n/2+j} = -\omega^j$
- squaring them produces the $(n/2)_{nd}$ roots of unity.



The nth complex roots of unity

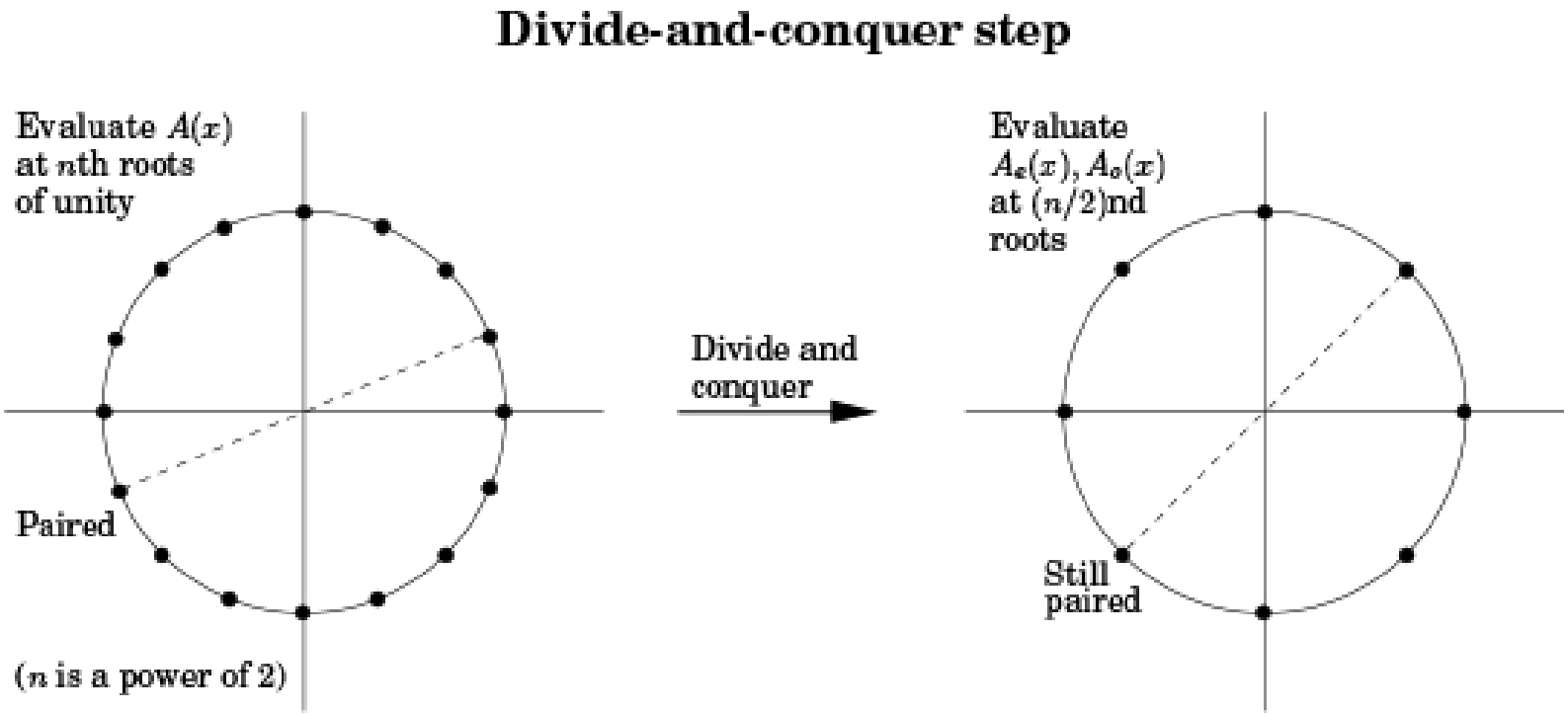
Solutions to the equation $z^n = 1$.

By the multiplication rule: solutions are $z = (1, \theta)$, for θ a multiple of $2\pi/n$ (shown here for $n = 16$).

For even n:

- These numbers are *plus-minus paired*: $-(1, \theta) = (1, \theta + \pi)$.
- Their squares are the $(n/2)$nd roots of unity, shown here with boxes around them.

The complex n_{th} roots of unity



Inversion Formula

- evaluation is multiplication by $M = M_n(\omega)$
- interpolation is multiplication by $\mathrm{M}^{-1} = \frac{1}{n} M_n(\omega^{-1})$.

The FFT multiplies an arbitrary n-dimensional vector (which we have been calling the coefficient representation) by the $n \times n$ matrix:

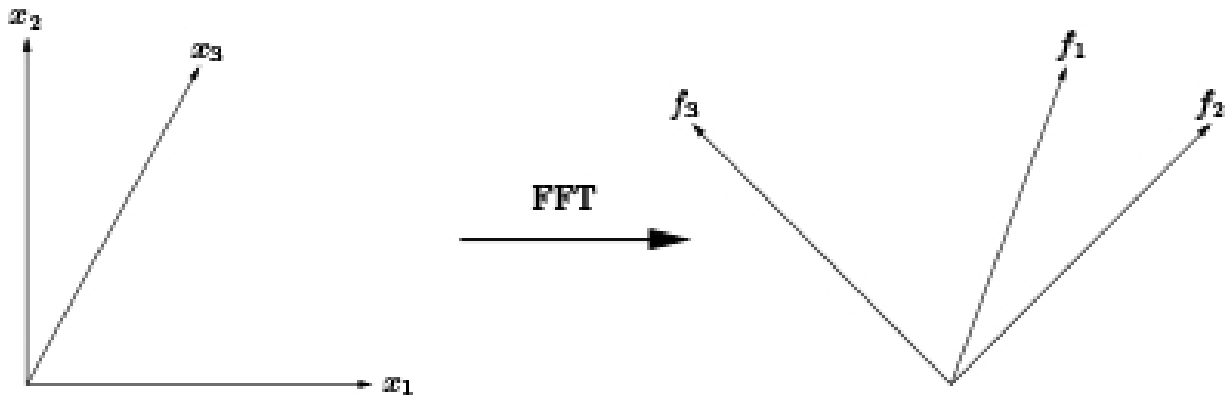
$$M_n(\omega) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ & & \vdots & & \\ 1 & \omega^j & \omega^{2j} & \cdots & \omega^{(n-1)j} \\ & & \vdots & & \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{array}{l} \longleftarrow \text{ row for } \omega^0 = 1 \\ \longleftarrow \omega \\ \longleftarrow \omega^2 \\ \\ \longleftarrow \omega^j \\ \\ \longleftarrow \omega^{n-1} \end{array}$$

where ω is a complex n_{th} root of unity, and n is a power of 2.
Its $(j, k)_{th}$ entry (starting row-count and column-count at zero) is ω^{jk}.

Inversion Formula

- multiplication by $M = M_n(\omega)$ maps the k_{th} coordinate axis (the vector with all zeros except for a 1 at position k) onto the k_{th} column of M
- the columns of M are orthogonal (at right angles) to each other
- the axes of an alternative coordinate system, (Fourier basis, FFT is a rigid rotation)

Figure 2.8 The FFT takes points in the standard coordinate system, whose axes are shown here as x_1, x_2, x_3, and rotates them into the Fourier basis, whose axes are the columns of $M_n(\omega)$, shown here as f_1, f_2, f_3. For instance, points in direction x_1 get mapped into direction f_1.



Interpolation - The FFT

$$\text{coefficients} \Leftrightarrow \text{values}$$

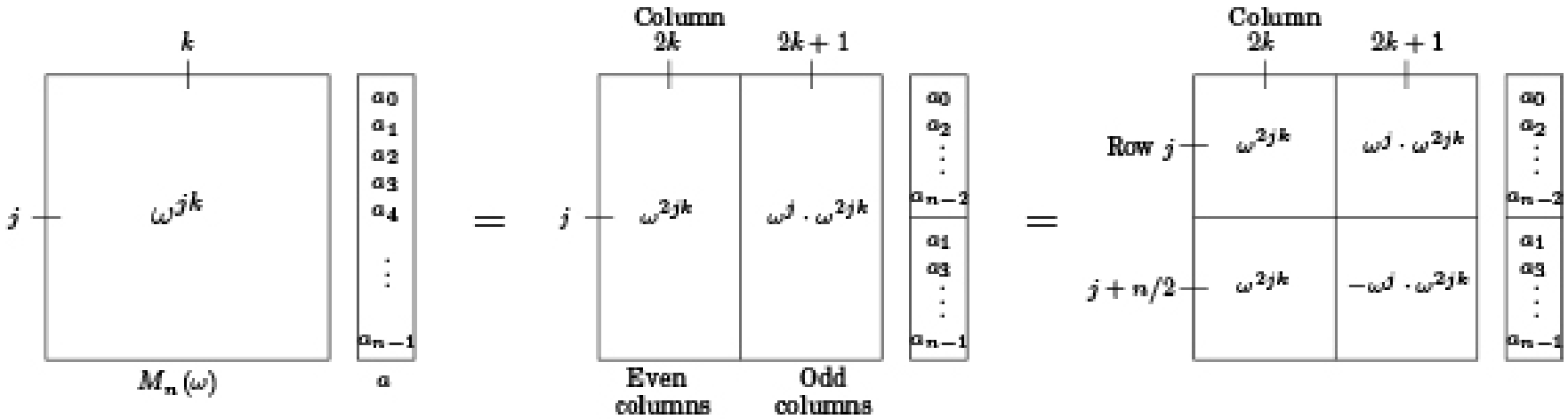
Complexity: $O(nlogn)$, when $\{x_i\}$ are complex n_{th} roots of unity $(1, \omega, ..., \omega^{n-1})$

$$\langle values \rangle = FFT(\langle coefficients \rangle, \omega)$$

Interpolation is the inverse operation:

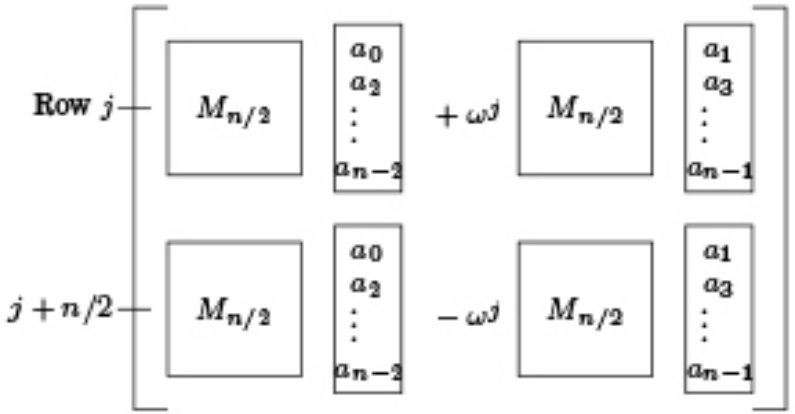
$$\langle coefficients \rangle = \frac{1}{n} FFT(\langle values \rangle, \omega^{-1})$$

The FFT algorithm



- M's columns are segregated into evens and odds
- simplified entries in the bottom half of the matrix using $\omega^{n/2} = -1$ and $\omega^n = 1$

The FFT algorithm



the product of $M_n(\omega)$ with vector $(a_0, ..., a_{n-1})$, a size-n problem, can be expressed in terms of two size-n/2 problems: the product of $M_{n/2}(\omega^2)$ with $(a_0, a_2, ..., a_{n-2})$ and with $(a_1, a_3, ..., a_{n-1})$.

Running time is $T(n) = 2T(n/2) + O(n) = O(nlogn)$.

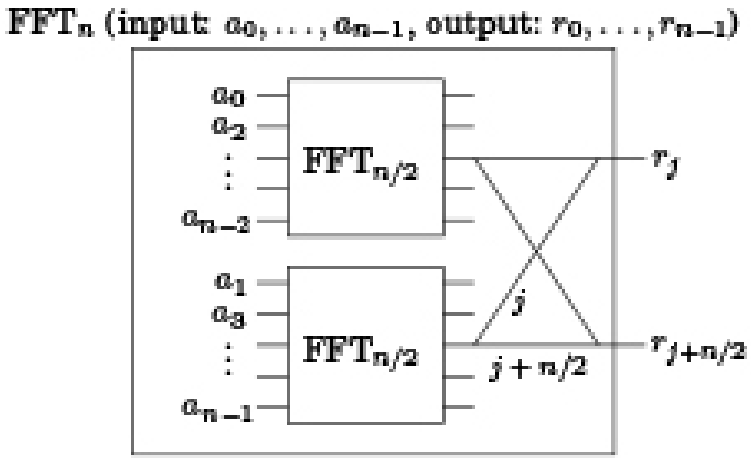
Transform circuit

The divide-and-conquer step of the FFT can be drawn as a very simple circuit.

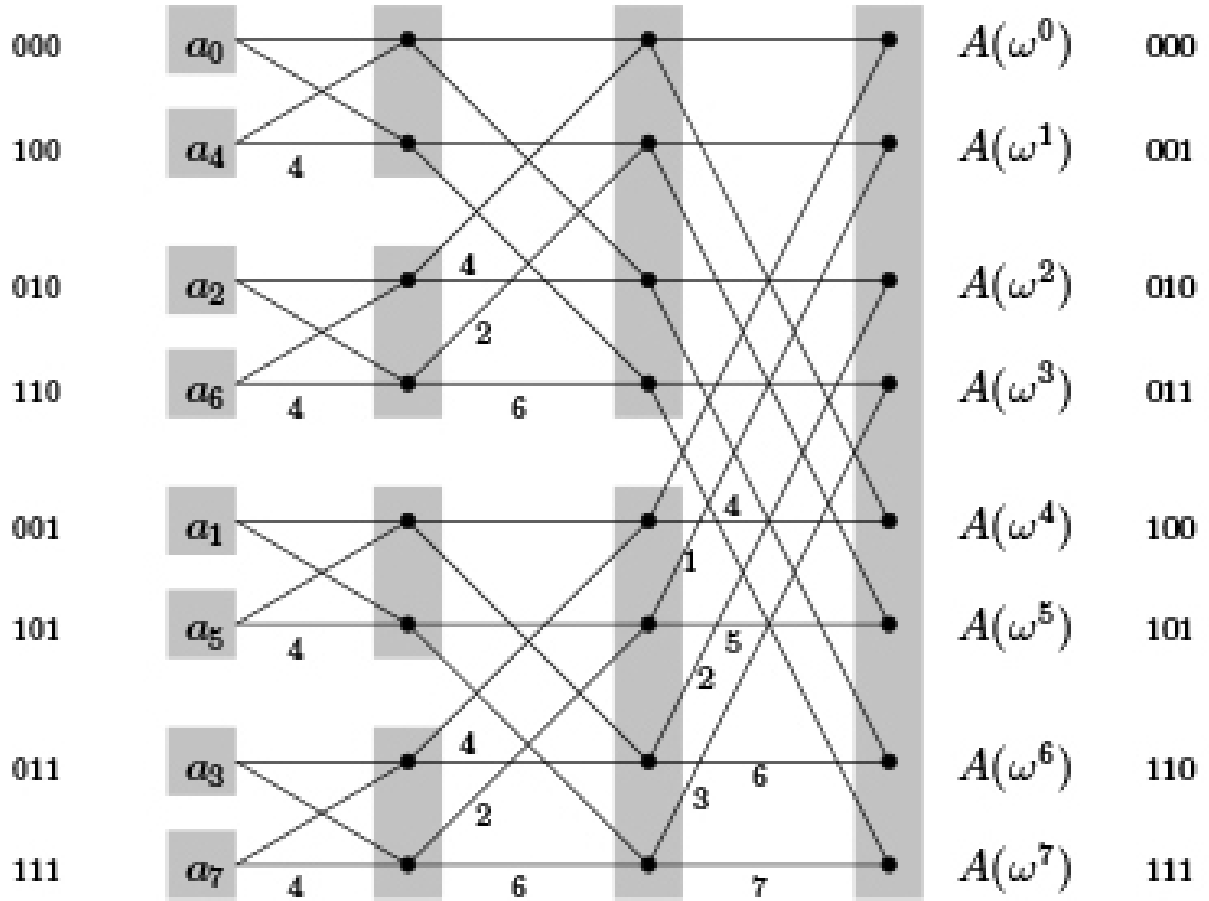
- the edges are wires carrying complex numbers from left to right
- a weight of j means multiply the number on this wire by ω^j
- when two wires come into a junction from the left, the numbers they are carrying get added up

So the two outputs depicted are executing the commands:

$$\begin{aligned} r_j &= s_j + \omega^j s'_j \\ r_{j+n/2} &= s_j - \omega^j s'_j \end{aligned}$$



Transform circuit



Transform circuit

- For n inputs there are $log_2 n$ levels, each with n nodes, for a total of n log n operations
- The inputs are arranged in the order specified by the recursion
- The resulting order in binary (000, 100, 010, 110, 001, 101, 011, 111) is the same as the natural one (000, 001, 010, 011, 100, 101, 110, 111) except the bits are mirrored
- There is a unique path between each input a_j and each output $A(\omega^k)$
- On the path between a_j and $A(\omega^k)$, the labels add up to jkmod8
- the FFT circuit is a natural for parallel computation and direct implementation in hardware.

Bibliography


T. Cormen and C. Leiserson and R. Rivest, *Introduction to Algorithms*. MIT Press, 1990.


Sanjoy Dasgupta and Christos H. Papadimitriou and Umesh V. Vazirani, *Algorithms*. McGraw-Hill, 2008.


Jon Kleinberg and Eva Tardos, *Algorithm design*. Pearson Education, 2006.