Transforms, Finite Fields, and Fast Multiplication

PATRICK CHIU
Stanford University
Stanford, CA 94305

Introduction The theme of a transform arises frequently. It entails transforming a problem into a setting where the new problem can be solved more easily and then applying the inverse transform to bring the solution back to the desired form. This is the general idea behind the fast Fourier transform (FFT) in a finite field for doing fast multiplication of large integers with absolute precision. Possible applications are to primality testing and cryptography where perfect accuracy is crucial.

Long ago in the seventeenth century, one highly effective use of a transform to make multiplication more tractable had been discovered, and was indispensable up to the computer age. In 1614 John Napier invented logarithms which transformed the multiplication problem into simple addition of exponents. Shortly after, Edward Gunter and William Oughtred built the powerful mechanical realization: the slide rule ([4, p. 247]).

The problem of multiplying big integers With the advent of the silicon chip, the celluloid slide rule became obselete. However, the problem of efficient multiplication lingered. It is now much more ambitious: given a computer capable of performing the operations $(+, -, *, \text{div})$ on the integers having up to l binary digits, how fast can it multiply two large n-digit numbers?

First, we need to state a decent method for measuring speed. Given a multiplication algorithm for large numbers, which invariably reduces to a bunch of l-digit size basic operations, one good method is to count the number of l-digit multiplications it takes to carry out the algorithm. As multiplications are slower than additions and occur more often in the algorithms we shall examine, they are a reasonable indication of speed.

Before discussing the fast Fourier transform, the classic grade school algorithm deserves a bit of analysis. When we multiply 123 by 456, we do this:

$$\begin{array}{r} 456 \\ \underline{123} \\ 1368 \\ 912 \\ \underline{456} \\ 56088 \end{array}$$

If we think of ourselves as computers that can multiply one-digit numbers, then the above takes $3^2 = 9$ multiplications. In general, to multiply two n-digit numbers requires n^2 multiplications. Extending the argument one step further, a computer capable of multiplying l-digit numbers would separate the n-digit numbers into $[n/l] + 1$ chunks, and multiplying chunk by chunk requires $([n/l] + 1)^2$ or $O(n^2)$ multiplications to perform the classic grade school algorithm.[1]

[1]A function $f(x)$ is $O(g(x))$ read "big oh of g," if $|f(x)| \leqslant Cg(x)$ for some constant C and all large x; it is used to indicate how fast a function grows.

In contrast, the fast Fourier transform will essentially allow us to multiply in $O(n \log n)$ multiplications!

Roots of the fast Fourier transform... In 1965 James Cooley and John Tukey published their famous paper [3], "An Algorithm for the Machine Calculation of Complex Fourier Series," which revolutionized that branch of mathematics. The method had been discovered forty years earlier by Runge and Konig, and independently by Stumpff, but in the precomputer days it was simpler to compute with the old $O(n^2)$ methods (for details, see [2]). It was a case of an idea coming before its time; only after the birth of the computer which can follow complicated instructions perfectly and rapidly did the fast Fourier transform realize its great potential. With so many applications in areas like spectral analysis, signal processing, and solution of differential equations, it was deemed as a major breakthrough ([1]).

...Transplantation to the world of algebra Although the fast Fourier transform was originally developed in an analytic setting of complex variables, in dealing with the concept of multiplication, an algebraic approach seems more appropriate. The following treatment is due to Lipson [5].

In the grade school algorithm, the integers are represented by the familiar positional notation. This is equivalent to polynomials to be evaluated at the base; for example, $456 = 4x^2 + 5x + 6$ at $x = 10$. *Therefore, if we can multiply polynomials quickly, then we can multiply large numbers quickly.* The fast Fourier transform provides a shortcut for multiplying polynomials.

What determines a polynomial? Obviously the coefficients do. More subtly, so do the evaluations of the polynomial at n distinct points where $n-1$ is the degree.

THEOREM 1 (Lagrange Interpolation Formula). *Let $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ be n elements in a field F, and $\beta_0, \beta_1, \ldots, \beta_{n-1}$ be distinct in F. Then there exists a unique polynomial $f(x)$ in $F[x]$ of degree less than n such that $f(\beta_i) = \alpha_i$, $i = 0, 1, \ldots, n-1$.*

Proof. The polynomial

$$f(x) = \sum_{i=0}^{n-1} \alpha_i (x-\beta_0)\cdots(x-\beta_{i-1})(x-\beta_{i+1})\cdots(x-\beta_{n-1})/$$

$$(\beta_i-\beta_0)\cdots(\beta_i-\beta_{i-1})(\beta_i-\beta_{i+1})\cdots(\beta_i-\beta_{n-1})$$

is of degree less than n which works neatly. To establish uniqueness, suppose $g(x)$ in $F[x]$ is of degree less than n which also works. Then $g-f$ is a polynomial of degree less than n having n zeros since $g-f$ is zero at each distinct β_i. But in general, every non-trivial polynomial of degree d has at most d zeros in the field. Hence $g-f$ must be identically 0, implying $g=f$.

Let $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$ be polynomials corresponding to $(a_{n-1}\cdots a_0)_B$ and $(b_{n-1}\cdots b_0)_B$, the two n-digit integers in base B to be multiplied. B is chosen to be small enough so that the computer can handle the a_i's, large enough to be efficient; B is typically a large integer less than 2^l. Hereafter, no confusion should arise if "multiplication of l-digit integers (or coefficients)" is simplified to just "multiplication." Now $c(x) = a(x)b(x)$ has degree less than $2n$. Let $N = 2n$, and ω be a primitive Nth root of unity (ω is a primitive Nth root if N is the least integer such that $\omega^N = 1$, for example, $e^{2\pi i/N}$ in the complex number system). The strategy will be:

A. FORWARD TRANSFORM

Evaluate $a(x)$ and $b(x)$ on $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$, where ω is a primitive Nth root of unity, to obtain $\{a(1), a(\omega), \ldots, a(\omega^{N-1})\}$ and $\{b(1), b(\omega), \ldots, b(\omega^{N-1})\}$.

B. SOLVE IN NEW SETTING

Multiply pointwise $c(\omega^i) = a(\omega^i)b(\omega^i)$ to get $\{c(1), c(\omega), \ldots, c(\omega^{N-1})\}$. By the above theorem these points determine uniquely the polynomial $c(x) = a(x)b(x)$. Notice that we have *eliminated the need to deal with the cross terms* which arise when multiplying polynomials in the straightforward way.

C. INVERSE TRANSFORM

Interpolate the polynomial $c(x)$. End by rewriting it as $(c_{N-1} \cdots c_0)_B$.

Evaluation on $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ will be done by the procedure FFT to be described shortly. For interpolation, the Lagrange formula—while useful as a heuristic device—is far too clumsy for computing. Amazingly, interpolation in the inverse transform can be accomplished by basically the *same* procedure FFT used for evaluation in the forward transform.

There is also symmetry at a lower level. The set $W_N = \{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ possesses an important property of symmetry:

THEOREM 2. *Let $N = 2n$ be an even positive integer, ω an Nth root of unity, and $W_N = \{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$. If ω^i is in W_N, then $-\omega^i = \omega^{n+i}$ is in W_N.*

Proof. $(\omega^n)^2 - 1 = 0$ implies that $\omega^n = 1$ or -1; but ω being a primitive $2n$th root of unity means that $\omega^n = -1$. Now $-\omega^i = (-1)\omega^i = \omega^n\omega^i = \omega^{n+i}$ lies in the cyclic group W_N.

Notice that when we square each element of W_N, we obtain the cyclic group $W_{N/2}$ of $N/2$ elements, where ω^2 is a primitive $N/2$th root of unity. Applying theorem 2 shows that $W_{N/2}$ has the above property of symmetry if $N/2$ is even. By choosing N to be a power of 2, then inductively every set $W_N, W_{N/2}, W_{N/4}, \ldots, W_2 = \{-1, 1\}$ would have the symmetry. Moreover, each set $W_{N/2^k}$ clearly contains $N/2^k$ distinct elements.

Forward transform unveiled The procedure FFT will exploit this property of symmetry to the fullest. In the forward transform—evaluating a polynomial $a(x) = \sum_{i=0}^{n-1} a_i x^i$ on the set W_N—we need only do the necessary multiplications for *at most half the points*. Make the substitution $s = x^2$, then

$$a(x) = \left(a_0 + a_2 s + a_4 s^2 + \cdots + a_{n-2} s^{n/2-1}\right) + x\left(a_1 + a_3 s + a_5 s^2 + \cdots + a_{n-1} s^{n/2-1}\right).$$

Once $a(x)$ is computed, $a(-x)$ can be obtained without any more multiplications because

$$a(-x) = \left(a_0 + a_2 s + a_4 s^2 + \cdots + a_{n-2} s^{n/2-1}\right) - x\left(a_1 + a_3 s + a_5 s^2 + \cdots + a_{n-1} s^{n/2-1}\right)$$

as the -1 pops out of the odd terms. So far the number of multiplications is halved.

We can do even better. Each of the two polynomials $(a_0 + a_2 s + a_4 s^2 + \cdots + a_{n-2} s^{n/2-1})$ and $(a_1 + a_3 s + a_5 s^2 + \cdots + a_{n-1} s^{n/2-1})$ are just required to be evaluated on $W_{N/2}$ by the remarks of the last paragraph. The upshot is that having chosen the points $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ cleverly, evaluation on them requires substantially less multiplying than evaluation on N arbitrary distinct points.

For all subsequent evaluation purposes, given $f(x) = \sum_{i=0}^{N-1} f_i x^i$ to be evaluated on W_N, where N is a power of 2, use the following method.

PROCEDURE FFT

Step 1. Do if degree of $f(x)$ is 0.
Set answer as f_0.
End of procedure FFT.

Step 2. Do if degree of $f(x)$ is greater than 0.
a. Separate $f(x)$ into $p(x) = \sum_{i=0}^{N/2-1} f_{2i} x^i$ and $q(x) = \sum_{i=0}^{N/2-1} f_{2i+1} x^i$.
b. Recursively call procedure FFT to evaluate both $p(x)$ and $q(x)$, obtaining $p(\omega^{2k})$ and $q(\omega^{2k})$, for $k = 0, 1, \ldots, N/2 - 1$.
c. Set $f(\omega^k) = p(\omega^{2k}) + \omega^k q(\omega^{2k})$ and $f(\omega^{k+N/2}) = p(\omega^{2k}) - \omega^k q(\omega^{2k})$ for $k = 0, 1, \ldots, N/2 - 1$.

Now we estimate the speed.

THEOREM 3. *Procedure FFT requires* $(N/2)\log_2 N$ *multiplications to evaluate* $f(x) = \sum_{i=0}^{N-1} f_i x^i$ *on* W_N, *where* N *is a power of* 2.

Proof. Prove by induction. Let $M(N)$ be the number of multiplications for evaluation of $f(x)$ on W_N. The case $N = 1$ is trivial since Step 1 has no multiplications and $M(1) = 1/2 \log_2 1 = 0$. Assume the theorem to be true for all values less than N. Then the number of multiplications is the sum of the ones occurring in Step 2. Step 2.b takes $2M(N/2)$ and Step 2.c takes $(N/2)$. Hence

$$M(N) = 2M(N/2) + (N/2)$$

and by the inductive hypothesis,

$$M(N) = 2(N/4)\log_2(N/2) + (N/2)$$
$$M(N) = N/2(\log_2(N/2) + 1)$$
$$M(N) = N/2(\log_2 N).$$

The inverse transform—déjà vu Here, the problem is to interpolate the polynomial $c(x) = \sum c_i x^i$ given $\{c(1), c(\omega), \ldots, c(\omega^{N-1})\}$. Equivalently, we wish to find the c_i's in the following matrix equation:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & & \omega^{N-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{N-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \omega^{N-1} & (\omega^{N-1})^2 & \cdots & (\omega^{N-1})^{N-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} c(1) \\ c(\omega) \\ c(\omega^2) \\ \vdots \\ c(\omega^{N-1}) \end{bmatrix}.$$

The square matrix is called a Vandermonde matrix which is usually invertible. In our context, the inverse has an especially nice form when N^{-1} lives in the field.

$$N^{-1}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & (\omega^{-1})^2 & \cdots & (\omega^{-1})^{N-1} \\ 1 & \omega^{-2} & (\omega^{-2})^2 & \cdots & (\omega^{-2})^{N-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \omega^{-(N-1)} & (\omega^{-(N-1)})^2 & \cdots & (\omega^{-(N-1)})^{N-1} \end{bmatrix}$$

This is the inverse as readily checked. (Hint: think geometric series.) Hence

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix} = N^{-1}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \cdots & (\omega^{-1})^{N-1} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega^{-(N-1)} & \cdots & (\omega^{-(N-1)})^{N-1} \end{bmatrix}\begin{bmatrix} c(1) \\ c(\omega) \\ \vdots \\ c(\omega^{N-1}) \end{bmatrix}$$

Now the right-hand side is just an evaluation problem! The polynomial is $N^{-1}\sum_{i=0}^{N-1} c(\omega^i)x^i$; the points to be evaluated on are $\{1, \omega^{-1}, (\omega^{-1})^2, \ldots, (\omega^{-1})^{N-1}\}$. Since ω is a primitive Nth root of unity, so is ω^{-1}. Therefore we can simply use the procedure FFT to interpolate with, needing hardly any modification.

The multiplication algorithm Finally, putting all the pieces together according to the strategy outlined earlier, we devise an algorithm for multiplying large integers.

MULTIPLICATION ALGORITHM

Input: Integer $N = 2n$, a power of 2
Primitive Nth root of unity ω
Multiplicands $(a_{n-1} \cdots a_0)_B$, and $(b_{n-1} \cdots b_0)_B$
Output: Product $(c_{N-1} \cdots c_0)B$
Step 1. Evaluate $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$ on $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ by calling procedure FFT.
Step 2. Multiply pointwise to get $\{a(1)b(1), a(\omega)b(\omega), \ldots, a(\omega^{N-1})b(\omega^{N-1})\}$.
Step 3. Interpolate $c(x) = \sum_{i=0}^{N-1} c_i x^i$ by evaluating $N^{-1}\sum_{i=0}^{N-1} a(\omega^i)b(\omega^i)x^i$ on $\{1, \omega^{-1}, (\omega^{-1})^2, \ldots, (\omega^{-1})^{N-1}\}$ by calling procedure FFT.
Step 4. Return $(c_{N-1} \cdots c_0)_B$.

THEOREM 4. *Let $N = 2n$ be an even integer, and F a field with N^{-1} and a primitive Nth root of unity. Then multiplying two n-digit integers can be accomplished in $O(n \log_2 n)$ multiplications.*

Proof. It suffices to show this for N a power of 2 because we are studying asymptotic behavior. To determine the speed, we count the number of multiplications occurring in the above algorithm. Step 1 uses $2(N/2)\log_2 N$ multiplications by theorem 3 in calling procedure FFT twice. Step 2 requires N multiplications, and Step 3 uses $(N/2)\log_2 N$. The total is

$$(3/2)N \log_2 N + N = 3n \log_2 2n + 2n = O(n \log_2 n).$$

Finite fields do have a lot to offer At this point, we have developed a fast multiplication algorithm over arbitrary fields in which exist N^{-1} and a primitive Nth root of unity. The question is which field to use. The fast Fourier transform's complex variables heritage together with the familiar primitive Nth root of unity $e^{2\pi i/N}$ make

the field of complex numbers $\mathbf{C}$ the natural choice. Computational difficulties arise, however, since the typical computer lacks the routines to do complex arithmetic with absolute precision. Such routines would be prohibitive to write because the components of $e^{2\pi i/N}$ may be irrational. An elegant alternative was presented by Pollard in [6], who overcame the fixation on $\mathbf{C}$. After all, $\mathbf{C}$ is not the only field that contains N^{-1} and a primitive Nth root of unity. We shall demonstrate that finite fields $\mathbf{Z}_p$, where p is prime and N divides $p-1$, have the desired properties. One pleasant consequence is that field multiplication uses the computer's built-in multiplication, plus a simple mod p reduction. Thus, the base B may be chosen to be a large prime, and multiplication is taken modulo B. (Technically, one also must worry about well-definedness.)

The key result describes the remarkably orderly structure of $\mathbf{Z}_p^x$, the multiplicative group of nonzero elements. It is true for all finite fields in general, and not much more difficult to prove.

THEOREM 5. *Let F be a finite field, F^x its multiplicative group of nonzero elements. Then F^x is cyclic.*

Proof. F is finite, so we may choose an element g having maximal order m, so $|F^x| \geqslant m$. F^x being a finite abelian group implies that every element of F^x has order dividing m, as readily checked. Hence, all the elements of F^x satisfy the equation $y^m - 1 = 0$. On the other hand, F is a field and has at most m zeros. Therefore, $|F^x| = m$, and g generates all of F^x.

From this theorem, we derive the criterion for a usable field $\mathbf{Z}_p$.

COROLLARY. *Let p be prime. If N divides $p-1$, then N^{-1} is in $\mathbf{Z}_p$ and $\mathbf{Z}_p$ has a primitive Nth root of unity.*

Proof. Obviously $N \leqslant p-1$, whence N is in $\mathbf{Z}_p$ and so must N^{-1}. $\mathbf{Z}_p^x$ is cyclic by the last theorem; let g be a generator, then $g^{(p-1)/N}$ is a primitive Nth root of unity.

Feasibility: a little handwaving goes a long way How hard is it to find these primes and their primitive roots? In practice, primes of the form $N = 2^r k + 1$ may be found in lists, e.g. see Riesel [7, pp. 381–83]. Their primitive roots are readily located by successively testing 2, 3, ... Philosophically, the multiplication algorithm over $\mathbf{Z}_p$ is feasible because given r, there exists an abundance of primes in the arithmetic sequence $2^r k + 1$ by appealing to an advanced result in number theory known as the Generalized Prime Number Theorem; and by another result in number theory, primitive roots make up more than 3 out of every π^2 elements in $\mathbf{Z}_p$ on the average ([5, p. 304]). Therefore, the primes and their primitive roots may be found in a reasonable amount of time.

Conclusion In the course of studying the fundamental concept of multiplication, many powerful ideas from different areas of mathematics come into play. The multiplication algorithm over a finite field is obviously not something that can be done effectively by hand, nor worth implementing on a computer for small numbers, so the grade school method still has its place. For large integers, such as those used in cryptography, using the fast Fourier transform is significantly faster and worth implementing.

Acknowledgements. Much thanks to two professors at UC San Diego: Dr. Lance Small for his inspiration and Dr. Ron Evans for his helpful comments. Also, the widely accessible computers at Stanford University were a great help.

REFERENCES

1. J. W. Cooley, P. A. W. Lewis and P. D. Welch, The fast Fourier transform and its applications, *IEEE Trans. on Education* 12 (1969), 34.
2. _____, Historical notes on the fast Fourier transform, *IEEE Trans. Audio and Electroacoustics* 15 (1967), 76–77.
3. J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Math. Comp.* 19 (1965), 297–301.
4. Howard Eves, *An Introduction to the History of Mathematics*. Rinehart, 1953.
5. John D. Lipson, *Elements of Algebra and Algebraic Computing*, Addison-Wesley, 1981.
6. J. M. Pollard, The fast Fourier transform in a finite field, *Math. Comp.* 25 (1971), 365–74.
7. Hans Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, 1985.

A Geometric Proof of Machin's Formula

ROGER B. NELSEN
Lewis and Clark College
Portland, OR 97219

In the history of the computation of π, *Machin's formula*

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}$$

has played an important rôle. Using Machin's formula and the Maclaurin series for the arctangent, William Shanks computed π to 707 places in 1873, a remarkable achievement for the time and an accomplishment which stood until 1946, when it was discovered that he had erred in the 528th place [2].

While the formula can be verified routinely using trigonometry, there is a simple constructive proof using only geometry and interpreting arctangents as angles. If we set $\theta = \arctan(1/5) = \arctan(30/150)$, then by use of similar triangles we find $2\theta = \arctan(60/144)$ [see FIGURE 1]. In a similar fashion, we find $4\theta = \arctan(119/120)$ [see FIGURE 2]. Extending the terminal side of 4θ and employing congruent triangles,

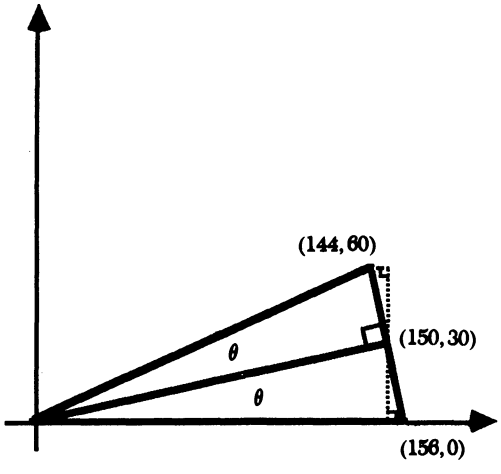

FIGURE 1