MATHEMATICS OF COMPUTATION
VOLUME 62, NUMBER 205
JANUARY 1994, PAGES 305-324

DISCRETE WEIGHTED TRANSFORMS AND LARGE-INTEGER ARITHMETIC

RICHARD CRANDALL AND BARRY FAGIN

ABSTRACT. It is well known that Discrete Fourier Transform (DFT) techniques may be used to multiply large integers. We introduce the concept of Discrete Weighted Transforms (DWTs) which, in certain situations, substantially improve the speed of multiplication by obviating costly zero-padding of digits. In particular, when arithmetic is to be performed modulo Fermat Numbers $2^{2^m}+1$, or Mersenne Numbers 2^q-1, weighted transforms effectively reduce FFT run lengths. We indicate how these ideas can be applied to enhance known algorithms for general multiplication, division, and factorization of large integers.

1. INTRODUCTION

The utility of transform methods for multiplication of large integers is well known [1, 11, 12, 16]. The basic idea is to treat the digits of integers, in an appropriate base representation, as signals upon which we perform transforms. For general multiplication one often "zero-pads," i.e., appends a sufficient number of zero digits to each of two signals, so that multiplication is equivalent to cyclic convolution. This cyclic convolution can be performed via FFTs. Other techniques are known for negacyclic convolution, which is equivalent to multiplication modulo Fermat numbers, as we discuss herein. In the negacyclic case and in certain other cases one may avoid zero-padding, and hence reduce the transform run length. The purpose of this paper is to expand on the set of such cases. Though straightforward "grammar-school" multiplication for integers having N words each requires $O(N^2)$ bit operations, it has been shown that at least some transform methods require only $O(N \log N \log\log N)$ bit operations [1]. The primary feature of the weighted transform approach herein is that the implicit O constant is reduced in many cases, by virtue of reduced run length for the transforms.

Various combinations of the methods of this treatment have been used to achieve new results in primality proving, factorization, and other number-theoretic domains. These empirical results are enumerated in the last section.

2. WEIGHTED TRANSFORMS AND CONVOLUTION

For an integer x having digits $x_0, x_1, \ldots, x_{N-1}$ in some base representation, we define the signal of x as the collection of digits:

Received by the editor July 26, 1991 and, in revised form, April 6, 1992.
1991 *Mathematics Subject Classification.* Primary 11Y11, 11Y05; Secondary 11A07, 11A51, 65T10.

$$\mathbf{x} = \{x_j : 0 \le j < N\}. \tag{2.1}$$

For scalars A we define a scalar-signal product by

$$A\mathbf{x} = \{Ax_j\}. \tag{2.2}$$

For signals $\mathbf{a}$, $\mathbf{x}$ we define a signal-signal product by

$$\mathbf{ax} = \{a_j x_j\}, \tag{2.3}$$

and when all signal elements of $\mathbf{a}$ are nonzero, we denote an inverse signal by

$$\mathbf{a}^{-1} = \{a_j^{-1}\}. \tag{2.4}$$

Our Discrete Weighted Transform (DWT) is defined by analogy with the usual Discrete Fourier Transform (DFT). A weight signal $\mathbf{a}$, comprised of N nonzero constants, is understood. The weighted transform is then taken to be the signal $\mathbf{X}$ whose components are

$$X_k = \sum_{j=0}^{N-1} a_j x_j g^{-jk}, \tag{2.5}$$

where g is a primitive Nth root of unity in the appropriate domain. The inverse DWT is

$$x_j = (Na_j)^{-1} \sum_{k=0}^{N-1} X_k g^{kj}. \tag{2.6}$$

To express the straightforward relationship between the DWT and DFT, we use the following notation to represent (2.5) and (2.6) compactly:

$$\begin{aligned} \mathbf{X} &= \mathrm{DWT}(N, \mathbf{a})\mathbf{x} = \mathrm{DFT}(N)\mathbf{ax}, \\ \mathbf{x} &= \mathrm{DWT}^{-1}(N, \mathbf{a})\mathbf{X} = \mathbf{a}^{-1}\mathrm{DFT}^{-1}(N)\mathbf{X}. \end{aligned} \tag{2.7}$$

Another simple observation is that weighted transforms become precisely the DFTs in the degenerate case $\mathbf{a} = \mathbf{1}$, where the signal $\mathbf{1}$ denotes $\{1, 1, \ldots, 1\}$.

We denote the traditional cyclic convolution of two length-N signals $\mathbf{x}$, $\mathbf{y}$ by the signal $\mathbf{x} * \mathbf{y}$, whose components are

$$(\mathbf{x} * \mathbf{y})_n = \sum_{j+k=n \pmod N} x_j y_k. \tag{2.8}$$

We can isolate key parts of this cyclic convolution by defining, for $b = 0$ or 1, the convolutions

$$(\mathbf{x} * \mathbf{y})_n^{(b)} = \sum_{j+k=bN+n} x_j y_k, \tag{2.9}$$

so that

$$\mathbf{x} * \mathbf{y} = (\mathbf{x} * \mathbf{y})^{(0)} + (\mathbf{x} * \mathbf{y})^{(1)}. \tag{2.10}$$

We denote the negacyclic convolution referenced in the literature [11] by $\mathbf{x} \bullet \mathbf{y}$, defined as

$$\mathbf{x} \bullet \mathbf{y} = (\mathbf{x} * \mathbf{y})^{(0)} - (\mathbf{x} * \mathbf{y})^{(1)}. \tag{2.11}$$

Also of interest is a right-angle convolution [8], for which the coefficient of $(\mathbf{x} * \mathbf{y})^{(1)}$ is a square root of -1. The weighted transform approach can be thought of as a means for introducing general phase factors as coefficients of the part $(\mathbf{x} * \mathbf{y})^{(1)}$.

Note that acyclic convolution, which is the part $(\mathbf{x} * \mathbf{y})^{(0)}$, can be obtained from the sum of (2.10) and (2.11). Similarly, if $\mathbf{x}$, $\mathbf{y}$ are both real signals and complex arithmetic is used, then the acyclic convolution can be obtained simply as the real part of a right-angle convolution $(\mathbf{x} * \mathbf{y})^{(0)} \pm i(\mathbf{x} * \mathbf{y})^{(1)}$.

For a constant, length-N nonzero weight signal $\mathbf{a}$ understood, we define the weighted convolution of two length-N signals $\mathbf{x}$, $\mathbf{y}$ to be

$$\mathbf{x} *^{\mathbf{a}} \mathbf{y} = \mathbf{a}^{-1}((\mathbf{ax})^*(\mathbf{ay})). \tag{2.12}$$

In the important cases where the weight signal is generated from a scalar A,

$$a_j = A^j, \tag{2.13}$$

the weighted convolution takes the simple form

$$\mathbf{x} *^{\mathbf{a}} \mathbf{y} = (\mathbf{x} * \mathbf{y})^{(0)} + A^N(\mathbf{x} * \mathbf{y})^{(1)}. \tag{2.14}$$

Given the forward and inverse DWTs (2.5) and (2.6), the appropriate analog of the classical convolution theorem can be derived in a straightforward manner. From (2.5), (2.6), and the weighted convolution definition (2.12) one finds

$$\begin{aligned}(\mathbf{x} *^{\mathbf{a}} \mathbf{y})_n &= (\mathbf{a}^{-1})_n N^{-1} \sum_{k=0}^{N-1} X_k Y_k g^{kn} \\ &= \mathrm{DWT}^{-1}(N, \mathbf{a})(\mathbf{XY}) \\ &= \mathbf{a}^{-1}\mathrm{DFT}^{-1}(N)[(\mathrm{DFT}(N)\mathbf{ax})(\mathrm{DFT}(N)\mathbf{ay})],\end{aligned} \tag{2.15}$$

the last equality showing explicitly how to compute weighted convolutions via existing FFT algorithms for the DFTs.

3. Multiplication via weighted convolution of digits

We require some notation pertaining to digit representations of integers. The standard representation of a nonnegative integer x, for some fixed base W, involves digits x_j, where

$$x = \sum_{j=0}^{N-1} x_j W^j, \tag{3.1}$$

with all digits constrained by

$$0 \le x_j < W. \tag{3.2}$$

In many cases, notably when floating-point FFTs are employed, it is advantageous in practice to adopt a balanced representation, where we assume that W is even and for which we demand

$$-W/2 \le x_j < W/2. \tag{3.3}$$

A balanced representation thus involves "bipolar" digits which tend to yield reduced errors for the convolutions we intend to perform. In the sense of signal processing theory, the balanced representation involves digit signals $\mathbf{x}$ which are

in some sense "high-pass filtered," with the filter that converts from constraint (3.2) to constraint (3.3) being generally nonlinear. For one thing, the "DC component", or mean value of the $\mathbf{x}$ signal, is usually significantly smaller in balanced representation than in the standard representation.

Conversion between standard and balanced representation is not difficult. To convert from standard to balanced, one may proceed as follows. Starting at the least significant digit x_0, check whether this digit is as large as $W/2$. If $x_0 \geq W/2$, replace x_0 with $x_0 - W$ and increment x_1. Then apply this "if-subtract-increment" test to x_1, and so on, possibly with a final carry into one extra balanced digit $x_N = 1$.

We shall have occasion to contemplate variable-base representations of the form

$$x = \sum_{j=0}^{N-1} x_j \prod_{i=0}^{j} W_i = x_0 + x_1 W_1 + x_2 W_1 W_2 + \cdots , \tag{3.4}$$

where $W_0 = 1$ and all other W_i take on values from a finite set of even integers. Cumbersome as such representations might appear, the variable-base approach has resulted in unprecedented efficiencies for arithmetic modulo large Mersenne numbers. We define a standard variable-base representation by the constraint

$$0 \leq x_j < W_{j+1}, \tag{3.5}$$

and an alternative balanced-variable representation by the constraint

$$-W_{j+1}/2 \leq x_j < W_{j+1}/2. \tag{3.6}$$

When calculations are based on floating-point arithmetic, the functions Floor$(\cdot)$, Ceiling$(\cdot)$, and Round$(\cdot)$ are important, because various steps of the algorithms require integer digits at certain junctures. We define the functions as follows. For integers n,

$$\text{Floor}(n) = \text{Ceiling}(n) = \text{Round}(n) = n. \tag{3.7}$$

Otherwise, for $z = n + e$, where n is an integer and $0 < e < 1$,

$$\begin{aligned} &\text{Floor}(z) = n, \\ &\text{Ceiling}(z) = n + 1, \\ &\text{Round}(z) = \begin{cases} \text{Floor}(z + 1/2), & z \geq 0, \\ \text{Ceiling}(z - 1/2), & z < 0. \end{cases} \end{aligned} \tag{3.8}$$

Note that Floor$(\cdot)$ is not equivalent to the common machine function Trunc$(\cdot)$, the latter obtained merely by zeroing the fractional part of the mantissa. For example $z = -0.6$ has Floor$(z) = -1$, Ceiling$(z) = 0$, Round$(z) = -1$, Trunc$(z) = 0$. Inequalities that prove useful for certain weighted convolutions apply for all nonnegative reals a, b, in the form

$$\begin{aligned} &\text{Floor}(a) + \text{Floor}(b) \leq \text{Floor}(a + b), \\ &\text{Ceiling}(a) + \text{Ceiling}(b) \geq \text{Ceiling}(a + b). \end{aligned} \tag{3.9}$$

The Round$(\cdot)$ function is especially important when floating-point FFTs are used for convolution. In pseudocode descriptions we shall denote the round of a complex signal as

$$\text{Round}(\mathbf{z}) = \text{Round}(\text{Re}(z_j)) + i\,\text{Round}(\text{Im}(z_j)). \tag{3.10}$$

Such operations are used to infer correct integer convolution values from complex floating-point results.

In many applications of large-integer arithmetic one generally follows a multiplication xy by a mod operation. We should mention some known techniques for fast calculation of integers $(\mathrm{mod}\, p)$ when p has special form, for example $p = 2^q \pm 1$. In either respective case, $z \ (\mathrm{mod}\, p)$ can be computed rapidly by representing z in the form $a + b2^q$ and noting $z = a - (\pm b) \ (\mathrm{mod}\, p)$. One simply continues this reduction, which can be effected via shifts and adds alone, until z has a sufficiently small number of bits. Similar tricks apply to negation, multiplication by powers of two, and so on [7, 15]. As we shall see in §10, general mod operations can be efficiently performed via weighted transforms as long as the denominator remains fixed.

Nomenclature thus established, we can state the central algorithm as follows:

Algorithm W: **weighted convolution algorithm for multiplication of** x, y.

(1) Choose digit representations $\mathbf{x}, \mathbf{y}$, together with an appropriate run length N and weight signal $\mathbf{a}$.
(2) Compute $\mathbf{X} = \mathrm{DWT}(N, \mathbf{a})\mathbf{x}$, and $\mathbf{Y} = \mathrm{DWT}(N, \mathbf{a})\mathbf{y}$.
(3) Compute $\mathbf{Z} = \mathbf{XY}$.
(4) Compute $\mathbf{z} = \mathrm{DWT}^{-1}(N, \mathbf{a})\mathbf{Z}$. (This is the weighted convolution $\mathbf{x} *^{\mathbf{a}} \mathbf{y}$.)
(5) Set $\mathbf{z} = \mathrm{Round}(\mathbf{z})$, if noninteger (e.g., floating-point) FFTs were used.
(6) Adjust the digits $\{z_n\}$ to the digit representation of choice.

The various types of multiplication we anticipate (direct, polynomial, Fermat-mod, Mersenne-mod, and so on) will differ only in the choices that occur in step (1), and in the digit adjustment, step (6). The steps (2) and (4) require $O(N \log N)$ arithmetic (word) operations, while the signal multiplication (3), as well as the digit adjustment procedures in (5), (6), require only $O(N)$ arithmetic operations.

4. FFT MULTIPLICATION FOR INTEGERS AND POLYNOMIALS

Herein we review direct FFT methods with which multiplication in a field is achieved by cyclic convolution of zero-padded sequences [12]. The signal in this case is $\mathbf{a} = \mathbf{1}$. For nonnegative integers x, y we adopt representations of the form (3.1) and assume further that the digit sequences $\{x_j\}$, $\{y_j\}$ are zero-padded, in the sense that

$$x_j = y_j = 0 \quad \text{for } j \geq N/2. \tag{4.1}$$

The integer product xy is thus

$$xy = \sum_{n=0}^{N-1} \sum_{j+k=n} x_j y_k W^n. \tag{4.2}$$

Because of the zero-padding, the part $(\mathbf{x} * \mathbf{y})^{(1)}$ vanishes, and we may compute digits of xy according to

$$xy = \sum_{n=0}^{N-1} (\mathbf{x} * \mathbf{y})_n W^n. \tag{4.3}$$

Thus, multiplication can be effected via cyclic convolution of the $\mathbf{x}$, $\mathbf{y}$ signals. For integer multiplication, Algorithm W may thus start with the specific paraphrase:

(1) Represent x, y in base W, with the digits of x, y zero-padded such that $x_j = y_j = 0$ for $j \geq N/2$, with run length N appropriate to the available FFT routines, and choose $\mathbf{a} = \mathbf{1}$.

Note that the new digits $z_n = (\mathbf{x} * \mathbf{y})_n$ may violate the representation's digit constraint, so that some carry operations may be required to adjust (4.3) to a legal representation, as in step (6) of Algorithm W. This adjustment of digits is especially important when repeated multiplication is contemplated, i.e., when one wishes to loop back to step (1) many times.

For multiplication of two polynomials x, y, each with integer coefficients, everything proceeds as in the integer multiplication case, except that *less* work is needed: step (6) is not performed. This is because, after step (5), the digits of z_n are in fact the correct integer coefficients of the polynomial product.

For multiplication of two polynomials with all coefficients interpreted $(\mathrm{mod}\, p)$ one would paraphrase step (6) as:

(6) Set $z_n = z_n \pmod p$ for $n = 0, 1, \ldots, \deg(x) + \deg(y)$.

A critical issue in the implementation of large-integer direct FFT multiplication is the choice of FFT algorithm. Define a real-signal FFT as one that exploits the fact of all original signal elements being real, and a real-result inverse FFT as one that exploits the fact of all final signal elements being real. In practice one has options to which we next devote a few paragraphs.

For N a power of two, use a real-signal FFT in step (2) of Algorithm W, and a real-result inverse FFT in step (4); for example, the very efficient split-radix forward and inverse FFTs of Sorensen et al. [19] achieve in practice more than twice the speed of their conventional complex-signal Cooley-Tukey counterparts. Another common technique of taking the FFT of two real signals at once, using the complex signal $\{x_j + iy_j\}$, gives better overall performance than the employment of two separate complex FFTs, but in our experience this approach is consistently slower than the split-radix real-signal method.

For N again a power of two, use right-angle convolution to avoid zero-padding at the expense of invoking a complex FFT. We observe from (2.14) that for weighting constant $A = e^{\pi i/(2N)}$ the desired product can be obtained as the acyclic (real) part of the weighted convolution, plus W^N times the imaginary part. This approach is of interest when special memory constraints prevent zero-padding, or when an especially efficient complex FFT routine is available.

For N not a power of two, but rather a product of small primes to powers, a Prime Factor Algorithm (PFA) may be suitable. Some striking successes in large-integer arithmetic have been achieved in this way [4]. In the present treatment we concentrate throughout on N a power of two; in fact, it is the special feature of the weighted transform approach that relatively small run lengths $N = 2^m$ can often be used.

Our implementations of direct FFT multiplication, whether in standard or balanced representation, use word size $W = 2^{16}$. Even though our machines possess means for arithmetic on 32- or 48-bit integers, the choice $W = 2^{16}$ is the largest reasonable word size for the simple reason that convolution errors attendant to floating-point methods must be kept under control. The problem

of bounding convolution errors is extremely difficult. Though some interesting general theorems are known on the subject of FFT errors [3], the general results do not always give a fair picture of the errors obtained in practice. One problem is the fact of more than one source of error; for example, one suffers from both roundoff error and errors in the representations of sin and cos. Since convolution errors are not completely understood, some basic empirical observations are of interest. When the weighted convolution is pure real, then in step (5) of Algorithm W it is certainly necessary that an error bound of the form

$$e_n = |\mathrm{Re}(z_n) - (\mathbf{x} *^{\mathbf{a}} \mathbf{y})_n| < 1/2 \tag{4.4}$$

hold, lest the operation of Round$(\cdot)$ possibly give incorrect convolution values. For floating-point arithmetic with Q-bit mantissa resolution, on the assumption that $\sin(\cdot)$ and $\cos(\cdot)$ mantissas be correct to $Q-1$ bits, our numerical work suggests that some bound of the form

$$e_n < c2^{-Q}W^2N^{3/2}\log N \tag{4.5}$$

might hold for a universal constant $c \sim 1$ when standard digit representation is used. Furthermore, when balanced representation is used, it appears that a much better bound, better by a factor of $\sqrt{N}$, may hold in the form

$$e_n < c'2^{-Q}W^2N\log N. \tag{4.6}$$

Again, these bounds are conjectural, and based only on a finite set of experiments. It is reasonable, though, that the error bound for balanced representation be tighter, because the balanced digits should in some average sense behave as distributed bipolar values, giving rise to some error cancellation in the final convolution. In fact, we found that on a Cray YMP, with floating-point FFT routines involving 48-bit floating-point numbers, the standard representation is useless (i.e., e_n exceeds 0.5 in some cases) for $W = 2^{16}$ and multiplicands having more than 2^{18} bits. But balanced representation on this machine appears to allow accurate multiplication with 2000000 bits per multiplicand. For 64-bit floating-point mantissa resolution and fixed word size $W = 2^{16}$, direct FFT multiplication appears sufficiently accurate for general integers x, y having up to 2^{21} bits each in standard representation. When balanced representation is used, the 64-bit floating-point machines can perform this multiplication on integers having up to 2^{24} bits. In spite of the lack of rigorous results concerning the conjectures (4.5), (4.6), it appears that programs should always avoid standard representation when such avoidance is possible.

The fact of floating-point convolution errors is unfortunate, and may be discomforting to the reader. There are at least two sources of relief on this issue. First, in factoring experiments per se, a discovered factor can immediately be tested as a divisor, so that floating-point errors in any intermediate stages are irrelevant. Second, there exist errorless, integer convolutions, examples of which we discuss in §8. One approach is to write very fast floating-point routines, in this way obtaining results, then checking these results at critical junctures with rigorous integer routines.

Finally, one may drastically increase error margins by doing an error-correction side calculation. One may compute, in addition to the main convolution, the same convolution $(\mathrm{mod}\, B)$, where B is some convenient small integer such as 256, and in this way relax the constraint (4.4) to

$$e_n < B/2. \tag{4.7}$$

This method works because the word size B, when significantly less than W, gives rise to much less error, as expected on the basis of a heuristic relation (4.5) or (4.6). In this way one may use the components of a sufficiently reliable (mod B)-convolution to force the values of the less reliable components of the main convolution.

5. Fermat numbers and negacyclic convolution

Here we describe in detail a situation, discovered in essence by Schönhage and Strassen [16], in which negacyclic convolution may be used to avoid zero-padding of digits, resulting in a halving of the run length that was required by the direct FFT method. The Fermat numbers are defined by

$$F_m = 2^{2^m} + 1. \tag{5.1}$$

Multiplication modulo F_m may be effected as follows. Adopt a fixed base W which divides $F_m - 1$, say $W = 2^{2^m/N}$. Represent integers $x, y \pmod{F_m}$, neither of which $= -1 \pmod{F_m}$, as in (4.1). It is assumed that the extraneous cases where some key integer $= -1 \pmod{F_m}$ can be handled by simple means. Note that we shall not be zero-padding the signals in the present case. Indeed, since $W^N = -1 \pmod{F_m}$, it follows that

$$xy = \sum_{n=0}^{N-1} (\mathbf{x} \bullet \mathbf{y})_n W^n \pmod{F_m}. \tag{5.2}$$

In other words, the digits of $xy \pmod{F_m}$ can be taken to be components of the negacyclic convolution of $\mathbf{x}$ and $\mathbf{y}$ as defined by (2.11). As before in the direct FFT method, a reduction of these digits to the current representation of choice it usually required.

The weighted transform concept comes into play as follows. Set $a_j = A^j$, where A is an Nth root of -1. When floating-point FFTs are to be used, $A = e^{\pm \pi i/N}$ will suffice. The first step in Algorithm W appropriate to this negacyclic case is:

(1) Choose a base $W = 2^{2^m/N}$, where N will be the run length, and define the signal $\mathbf{a} = \{A^j\}$, where $A = e^{-\pi i/N}$. Represent x, y in base W, zero-padding only to N digits inclusive.

One might attempt to argue that the weighted transform approach gives no net advantage, by observing that the direct FFT method of §4 involved zero-padding, but allowed real-signal FFTs, whereas a transform such as

$$X_k = \sum_{j=0}^{N-1} x_j e^{-\pi i j/N} e^{-2\pi i j k/N}, \tag{5.3}$$

though enjoying half the run length, is nevertheless a complex-signal transform. The avoidance of zero-padding, this argument goes, would be offset by the need for complex transforms in place of the faster real-signal transforms. This argument fails, for the simple reason that transforms of the type (5.3) can actually be effected in terms of real-signal transforms of length N, with only a few extra operations that do not affect the asymptotic run time. Define a special transform

$$X'_k = 2 \sum_{j=0}^{N-1} x_j \cos(\pi j/N) e^{-2\pi i j k/N}, \tag{5.4}$$

which can, after $O(N)\cos(\cdot)$ multiplications, be computed as a real-signal FFT. It should be mentioned that this special transform is, strictly speaking, not a weighted transform because the weight signal $\{\cos(\pi j/N)\}$ is not invertible. Noting the identity

$$X'_k = X_k + X_{k-1} \tag{5.5}$$

and the fortuitous symmetry

$$X_k = X^*_{N-k-1}, \tag{5.6}$$

we may obtain X_0 directly from (5.3), then use the recursion

$$X_1 = X'_1 - X_0, \quad X_2 = X'_2 - X_1, \ldots, \tag{5.7}$$

so that, indeed, the DWT (5.3) can be obtained from a real-signal FFT and $O(N)$ extra operations. In fact, for actual multiplication we only need compute X_k and Y_k for $0 \le k < N/2$, because the symmetry (5.6) determines the rest of the components. Similarly for the final, inverse transform, we can compute the negacyclic components

$$z_n = (\mathbf{x} * \mathbf{y})_n = e^{\pi i n/N} N^{-1} \sum_{k=0}^{N-1} X_k Y_k e^{+2\pi i k n/N} \tag{5.8}$$

via a real-result inverse FFT:

$$(2\cos(\pi n/N))z_n = N^{-1} \sum_{k=0}^{N-1} (X_k Y_k + X_{k-1} Y_{k-1}) e^{+2\pi i k n/N} \tag{5.9}$$

when $n \neq N/2$, and obtain the single missing component from the side calculation

$$z_{N/2} = -2N^{-1} \sum_{k=0}^{N/2-1} \mathrm{Im}(X_k Y_k)(-1)^k. \tag{5.10}$$

In summary, multiplication modulo Fermat numbers may be effected without zero-padding, with a genuine gain, in the form of halved run length, through the use of real-signal FFTs and real-result inverse FFTs.

It is even possible to cut down the run length to 1/4 of that required for the direct FFT approach. In this case complex-signal FFTs must be used, but the method is called for in situations where, for example, a very fast complex FFT is available (more precisely, when the add-carry operations are relatively expensive in comparison to the FFTs). As before, let the fixed word length be $W = 2^{2^m/N}$, but now represent an integer x by a complex equivalent,

$$x' = \sum_{j=0}^{N/2-1} (x_j + i x_{j+N/2}) W^j, \tag{5.11}$$

and employ an analogous representation for an integer y. Define a new run length $N' = N/2$, and posit a constant length-N' signal $\mathbf{a} = \{A^j\}$, where $A^{N'} = i$. It is straightforward to show that

$$x'y' = \sum_{n=0}^{N'-1} (\mathbf{x}' *^{\mathbf{a}} \mathbf{y}')_n W^n \pmod{F_m}, \tag{5.12}$$

where the weighted convolution is, in view of the present definition of A, a right-angle convolution:

$$\mathbf{x}' *^{\mathbf{a}} y' = (\mathbf{x}' * y')^{(0)} + i(\mathbf{x}' * y')^{(1)}. \tag{5.13}$$

Algorithm W for this case requires step (1) to read:

(1) Choose a base $W = 2^{2m/N}$, where $N' = N/2$ will be the run length, and define the signal $\mathbf{a} = \{A^j\}$, where $A = e^{-\pi i/(2N')}$. Represent x, y as having N' complex digits, each in base W, as in (5.11), zero-padding only to N' complex digits inclusive.

It should be kept in mind that in steps (5) and (6) the digits $\{z_n\}$ are generally complex.

In summary, this algorithm has 1/4 the run length of the direct FFT method, but necessarily involves complex FFTs. We have found this approach to be effective on Cray machines, for which vectorized, optimized complex FFTs are available, and for which the carry adjustments on step (6) are difficult to vectorize.

6. Mersenne numbers and irrational bases

For $p = 2^q - 1$, we consider multiplication $(\mathrm{mod}\, p)$. An interesting observation is that binary multiplication of two integers x, y $(\mathrm{mod}\, p)$ is equivalent to cyclic convolution of bits. If we adopt a binary representation

$$x = \sum_{j=0}^{q-1} x_j 2^j, \tag{6.1}$$

and an analogous form for y, then, because of the fact that $2^q = 1 \pmod p$,

$$xy = \sum_{n=0}^{q-1} (\mathbf{x} * \mathbf{y})_n 2^n \pmod p. \tag{6.2}$$

Elegant though this equivalence of bitwise cyclic convolution and multiplication $(\mathrm{mod}\, p)$ may be, there are two reasons why the scheme is impractical. First, most machines are relatively inefficient in performing one-bit multiplication; and second, any FFT methods for the convolution must involve length-q signals. In many interesting cases, q is prime, and although prime-length FFTs can be performed via established methods, such FFTs for large q are not generally competitive with fast transforms for comparable but highly composite run lengths.

We have been able to circumvent these drawbacks of bitwise convolution by employing weighted transforms to perform arithmetic with respect to irrational bases. The new method is based on the observation that, if we could generalize the representation (6.1) to

$$x = \sum_{j=0}^{N-1} x_j 2^{qj/N}, \tag{6.3}$$

with an analogous representation for y, then in spite of the general irrationality of the digits x_j, y_j we can write xy $(\mathrm{mod}\, p)$ again in terms of a cyclic

convolution, this time of length N:

$$xy = \sum_{n=0}^{N-1} (\mathbf{x} * \mathbf{y})_n 2^{qn/N} \pmod{p}. \tag{6.4}$$

Of course, we cannot normally handle irrational representations exactly by computer, but weighted transforms and floating-point arithmetic can be brought to bear in order to compute convolutions such as (6.4) with sufficient accuracy. We adopt an integer representation, but with variable base as in (3.4),

$$x = \sum_{j=0}^{N-1} x_j 2^{\text{Ceiling}(qj/N)}. \tag{6.5}$$

The word sizes are $W_j = 2^{b_j}$, with the number of bits allocated for digit x_{j-1} being

$$b_j = \text{Ceiling}(qj/N) - \text{Ceiling}(q(j-1)/N). \tag{6.6}$$

From (3.9) it follows that for given q and run length N the b_j take on at most two possible values, namely Ceiling(q/N) or Floor(q/N). An appropriate weighted convolution proceeds on the basis of the constant signal $\mathbf{a}$ defined by

$$a_j = 2^{\text{Ceiling}(qj/N) - qj/N}, \tag{6.7}$$

which in practice will be approximated by a floating-point number always lying in the interval $[1, 2)$. Assuming the representation (6.5) for an integer x, and the analogous representation for an integer y, we have

$$\begin{aligned} xy &= \sum_{n=0}^{N-1} ((\mathbf{ax}) * (\mathbf{ay}))_n 2^{qn/N} \pmod{p} \\ &= \sum_{n=0}^{N-1} (\mathbf{x} *^{\mathbf{a}} \mathbf{y})_n 2^{\text{Ceiling}(qn/N)} \pmod{p}. \end{aligned} \tag{6.8}$$

Thus, the digits of the weighted convolution $\mathbf{x} *^{\mathbf{a}} \mathbf{y}$ will serve in the variable-base representation of $xy \pmod{p}$. The reason for using the Ceiling$(\cdot)$ function and the particular definition (6.7) for the components of $\mathbf{a}$ now becomes evident, as we observe that each convolution component $(\mathbf{x} *^{\mathbf{a}} \mathbf{y})_n$ in (6.8) must be an integer. Indeed, this component always takes the form

$$x_j y_k 2^{\text{Ceiling}(qj/N) + \text{Ceiling}(qk/N) - \text{Ceiling}(qn/N)} \tag{6.9}$$

with $j + k = n \pmod{N}$. By (3.9) it follows that the exponent of 2 in (6.9) is always 0 or 1. This means that our usual rounding techniques can be used to ascertain exact integer values for the weighted convolution, assuming that the floating-point arithmetic is sufficiently precise.

Algorithm W for multiplication of integers $x, y \pmod{p}$, where $p = 2^q - 1$, via weighted convolution of this type may now proceed with a specific step (1):

(1) Choose run length $N \geq q$ and establish bit-sizes b_j for digits according to (6.6). Represent $\mathbf{x} = \{x_j\}$, $\mathbf{y} = \{y_j\}$ according to (6.5), zero-padding only to N digits inclusive. Compute the components of the weight signal $\mathbf{a}$ according to (6.7).

The digit representations may be either standard-variable or balanced-variable type. One simply sticks to a consistent constraint (3.5) or (3.6) in steps (1) and (6). Our most powerful Mersenne-mod routines use balanced-variable representation to cut down the overall convolution errors.

Since the variable-base representations are not common ones for programmers, a brief worked example of the multiplication algorithm is in order. Let

$$q = 37, \quad p = 2^{37} - 1, \quad N = 4. \tag{6.10}$$

The bit-sizes from (6.6) are

$$\{b_j\} = \{10, 9, 9, 9\}. \tag{6.11}$$

Let us use the algorithm to square the number $x = 78314567209 \pmod{p}$. In the first step of Algorithm W we determine

$$\mathbf{x} = \{553, 93, 381, 291\}. \tag{6.12}$$

Note, for example, that 553 is indeed a 10-bit number and that the other three digits are 9-bit numbers. The constant signal is computed as

$$\mathbf{a} = \{1, 2^{3/4}, 2^{1/2}, 2^{1/4}\}. \tag{6.13}$$

After step (4) we find a typical floating-point representation of the $\mathbf{z}$ signal:

$$\mathbf{z} = \{704383., 324600., 523365.0000000001, 463577.9999999999\}. \tag{6.14}$$

The rounding in step (5) gives us integer digits for the convolution components in (6.8). Adjustment for proper constraints on digit sizes are to follow, but we can see that the weighted convolution has given the correct product $\pmod{p}$. Indeed, it is easy to check that

(6.15)

$$78314567209^2 = 704383 + 324600 * 2^{10} + 523365 * 2^{19} + 463578 * 2^{28} \pmod{2^{37} - 1}.$$

7. Chinese remainder methods

The weighted convolutions of the last two sections, suitable for specific Fermat- or Mersenne-mod cases, can sometimes be used together to enhance general multiplication routines. Assume, for example, that each of x, y has $2^m + s$ digits in a fixed, even base W, with $0 < s < 2^{m-1}$. The required zero-padding of the direct FFT method of §4, for power-of-two run length N, implies a minimum run length $N = 2^{m+2}$. That is, each of x, y must be zero-padded to length 2^{m+1}, and further zero-padded to run length 2^{m+2}. Because of the assumed constraint on s, we have the option of using the methods of §§5, 6 respectively, to compute two integers

$$u = xy \pmod{W^{2^{m+1}} + 1}, \qquad v = xy \pmod{W^{2^m} - 1}, \tag{7.1}$$

and, because the product xy has at most $3 * 2^m$ nontrivial digits, to reconstruct the exact product using (7.1) and the Chinese Remainder Theorem (CRT). Since W is even, the two modulus bases in (7.1) are relatively prime, whence

$$2xy = u + v + W^{2^{m+1}}(v - u) \pmod{(W^{2^{m+1}} + 1)(W^{2^m} - 1)}. \tag{7.2}$$

The mod operation in (7.2) may be effected using the same principles that allow fast Fermat- and Mersenne-mod arithmetic; namely, a few shift and add operations generally suffice to perform the mod. Thus, for example, two 1500-digit base-W integers may be multiplied using a length-2048 negacyclic convolution (for u) and a length-1024 cyclic convolution (for v). This amounts to a noticeable speed improvement over the direct FFT method, which would require a length-4096 cyclic convolution.

A different, "microscopic" CRT approach is to attempt parallelism by computing weighted convolution components modulo distinct primes p_i. One might use a separate processor for each p_i. One might further demand that each p_i possesses a primitive root g_i of order N, and also that each p_i admits of a suitable weight signal $\mathbf{a}_i$; then calculate a weighted transform (2.5) modulo each p_i. Weighted convolution elements from (2.15) will then be known modulo each respective p_i, and can be reconstructed efficiently using known "pre-conditioning" algorithms for the CRT calculation [1]. To achieve unambiguous reconstruction, one must use enough primes so that πp_i exceeds the largest possible convolution component. Component bounds are discussed in the next section.

8. Number-theoretic weighted transforms

Traditional number-theoretic transforms, which by avoiding floating-point arithmetic provide exact integer convolutions, can be given weighted counterparts. Let integers x, y be expressed in balanced representations of the type (3.1), so that constraint (3.2) applies. Both cyclic and negacyclic convolutions then satisfy the inequality

$$|(\mathbf{x} *^{\mathbf{a}} \mathbf{y})_n| \leq NW^2/4 \tag{8.1}$$

for each $n = 0, 1, \ldots, N-1$. Consider number-theoretic weighted transforms (2.5) based on arithmetic over a finite commutative ring R with unity. A practical special case is

$$X_k = \sum_{j=0}^{N-1} x_j A^j g^{-jk}, \tag{8.2}$$

where A is invertible in R and g is a primitive Nth root of unity in R. We may take $A = 1$ for the cyclic case, or A is a primitive Nth root of (-1) for the negacyclic case. In this negacyclic case it suffices to use $g = A^2$. When we use a finite field $R = GF(p)$ for a prime p sufficiently large that

$$NW^2/4 < p/2, \tag{8.3}$$

then weighted transforms of the type (8.2), with all arithmetic performed $(\mathrm{mod}\, p)$, can be used to determine unambiguously the (possibly bipolar) convolution elements. Bounds sharper than (8.3) may be derived, especially if zero-padding of digits or special symmetries are taken into account.

A popular choice [8, 9] is $p = F_m = 2^{2^m} + 1$. The resulting Fermat Number Transform (FNT) has attractive features but one major drawback. The advantages of the FNT lie in one's ability to perform (8.2) (with the generating scalar A equal to a power of 2) on the basis of shift and add operations alone, while the drawback is that maximum allowable FNT run lengths are quite limited.

An FNT example runs as follows. Start with $W = 2^{16}$, $N = 64$, so that cyclic or negacyclic convolution are desired for x, y each having at most 1024 bits. Since $NW^2 = 2^{38}$, it suffices by (8.3) to choose $p = F_6 = 2^{64} + 1$. Choose $A = 2$, which is a 64th root of $(-1) \pmod{F_6}$, and $g = 4$. Then weighted transforms

$$X_k = \sum_{j=0}^{63} x_j 2^j 4^{-jk} \pmod{F_6} \tag{8.4}$$

may be used to compute negacyclic convolutions appropriate to original integers x, y having at most 1024 bits each. Clearly, (8.4) or its even simpler cyclic analog can be effected via shift, add, and Fermat-mod operations without any explicit multiplication required.

Because of the restrictions on FNT run length, it is somewhat difficult to perform exact multiplication of integers having, say, hundreds of thousands of bits in this way. However, there are multidimensional techniques that circumvent this problem somewhat [6, 9]. Luckily, there are other options. Consider what we shall call Galois Transforms, for which the expressions such as (8.2) are to be evaluated in $GF(p^2)$, where $p = 2^q - 1$ is a Mersenne prime. These transforms [13, 15] take due advantage of two facts. First, for arithmetic in $GF(p^2)$ we can assume that every field element is $a + bi$, with all real and imaginary components reduced $(\mathrm{mod}\, p)$ at every stage. Second, the order of the multiplicative group is $p^2 - 1 = (p + 1)(p - 1)$, which is divisible by 2^{q+1}, thus allowing in practice enormous power-of-two run lengths.

Let h be a primitive multiplicative 2^{q+1}th root of 1 in $GF(p^2)$. For $r \leq q$ let

$$N = 2^r, \quad A = h^{(p-1)2^{q-r-1}}, \quad g = A^2. \tag{8.5}$$

Since $A^N = -1$, length-N cyclic (where A is simply omitted from (8.2)) or negacylic digit convolution may proceed, under the constraint (8.3). If r is strictly less than q, then the choices

$$N = 2^r, \quad A = h^{(p-1)2^{q-r-2}}, \quad g = A^4 \tag{8.6}$$

allow right-angle convolution, because A^N is now a square root of (-1). In this case the acyclic convolution is just the real part of the weighted convolution.

By a theorem of Creutzburg and Tasche [5] one can obtain closed-form expressions for primitive roots in $GF(p^2)$. For example,

$$h = 2^{2^{q-2}} + (-3)^{2^{q-2}} i \tag{8.7}$$

is always a primitive multiplicative 2^{q+1}th root of 1. From (8.5) we can attempt run lengths as powers of two up through 2^q for cyclic or negacyclic convolution. An interesting further observation is that $gg^* = 1 \pmod{p}$, so that the aforementioned techniques for real-signal and real-result transforms may be applied to the weighted cases (8.2), in view of appropriate symmetries of $\mathbf{X}$.

A useful example of Galois Transform applications arises when p is the Mersenne prime $2^{61} - 1$. For fixed base $W = 2^{16}$, one may, by virtue of (8.3), contemplate run lengths up to the impressive size of $N = 2^{27}$ digits. Programs for weighted convolution can be realized by starting with a single primitive root such as

$$h = 2147483648 + 1033321771269002680\, i, \tag{8.8}$$

then using relations (8.5) or (8.6) to handle specific run lengths 2^r. Say that we wish to multiply two integers, each having ≤ 1 million bits. We may proceed with right-angle convolutions as follows. Make, according to (8.6), the choices

$$\begin{aligned} N &= 2^{16}, \\ A &= h^{(p-1)2^{43}} = 1973234539278172120 + 1244201103777839971\, i, \\ g &= A^4 = 1510466207055935382 + 120042544849731353\, i, \end{aligned} \tag{8.9}$$

for which one may verify that $A^N = i$ and thus $g^N = 1$ in $GF(p^2)$. Then (8.2) and the weighted convolution formula (2.15) can be used to compute the nth digit of xy as $\mathrm{Re}((\mathbf{x} *^{\mathbf{a}} \mathbf{y})_n)$ for $n < N$, and $\mathrm{Im}((\mathbf{x} *^{\mathbf{a}} \mathbf{y})_{n-N})$ for $n \geq N$. Though complex FFTs with 64-by-64 bit high-precision multiplies are called for in this example, it is guaranteed by (8.3) that the multiplication is exact, devoid of the errors attendant to floating-point methods.

9. Applications to factorization methods

The weighted transform approach can sometimes be used to enhance modern factorization algorithms when the number N to be factored (not to be confused with run length in this section) is sufficiently large. The idea is to use transform methods to resolve certain algebraic forms. By storing precomputed transforms, one may effectively reduce the total number of required multiplications for these forms.

A first example is a calculation that occurs in most implementations of the "second stage" of the Pollard $(p-1)$ Method or the Elliptic Curve Method (ECM) [10]. One accumulates products of the form

$$r = \prod_{i<j} (x_i - x_j) \pmod N, \tag{9.1}$$

where integers $x_1, x_2, \ldots, x_n$ have been stored, and computes $\mathrm{GCD}(r, N)$, hoping for a factor of N. If one computes r by successively calling a DWT-based general multiply routine, then $(n-2)(n+1)/2$ multiplies, hence $3(n-2)(n+1)/2$ transforms, will be required. But by storing transforms appropriately, we can cut the asymptotic $3n^2/2$ transform count down to n^2, as follows.

Algorithm for computing products (9.1) by storing transforms.

(1) Let $\mathbf{x}_i$ denote the signal corresponding to the digits of integer x_i, and compute and store signals $\mathbf{X}_i = \mathrm{DWT}(N, \mathbf{a})\mathbf{x}_i$, where $\mathbf{a}$ is appropriate to the problem of multiplication $(\mathrm{mod}\, N)$.

(2) Set $r = 1$.

(3) To introduce a new term $(x_i - x_j)$ into the product for r, set $\mathbf{r} = \mathrm{DWT}^{-1}(N, \mathbf{a})[(\mathbf{X}_i - \mathbf{X}_j)(\mathrm{DWT}(N, \mathbf{a})\mathbf{r})]$, and recover r from $\mathbf{r}$ by adjusting to the representation of choice $(\mathrm{mod}\, N)$. Repeat this step (3) until all terms are introduced into the product.

It is not hard to see that (9.1) can be computed in this way, with a total of $n^2 - 3$ DWTs (inverse transforms included), for an overall gain of 3/2 in the asymptotic run time.

As a second example, we analyze some ECM arithmetic. The inversionless parametrization of ECM due to Montgomery [10] uses the elliptic curve

$$By^2 = x^3 + Ax^2 + x \pmod N. \tag{9.2}$$

One assumes an initial rational point $(x, y) = (u_1/v_1, y_1)$, where y_1 is not needed in the calculation, and uses such formulae as

$$\begin{aligned} u_{m+n} &= v_{m-n}(u_m u_n - v_m v_n)^2 \pmod N, \\ v_{m+n} &= u_{m-n}(u_m v_n - v_m u_n)^2 \pmod N \end{aligned} \tag{9.3}$$

to obtain the x-coordinate of $(m+n)(u_1/v_1, y_1)$ when $m \neq n$, with a different (doubling) formula that applies when $m = n$. In this approach one computes certain multiples $k(u_1/v_1, y_1)$ and continually checks $\mathrm{GCD}(v_k, N)$, hoping for a factor of N. As Montgomery points out, the computation (9.3) requires 8 multiplications and some additions. The multiplications would involve 24 DWTs if a general weighted convolution multiplication routine were systematically called. If we exploit the fact that squares require only 2 DWTs, then (9.3) will require 22 transforms. This cost of 22 transforms can be reduced further, in the following way. First, compute four transforms $\mathrm{DWT}(N, \mathbf{a})\mathbf{u}_m$, $\mathrm{DWT}(N, \mathbf{a})\mathbf{u}_n$, $\mathrm{DWT}(N, \mathbf{a})\mathbf{v}_m$, $\mathrm{DWT}(N, \mathbf{a})\mathbf{v}_n$. Then each square in (9.3) can be computed with three transforms, and, because the transforms of $\mathbf{u}_{m-n}$, $\mathbf{v}_{m-n}$ can have been stored, four more transforms will yield the desired final values in (9.3). Thus, a total of 14 DWTs can be employed to resolve (9.3). In this way, ECM arithmetic can be sped up by factors $> 3/2$.

As intimated in §1, these methods for improving factorization times will provide genuine improvement when N has some thousands of binary bits, the precise magnitude of N at which weighted transform methods prevail being machine-dependent.

10. Generalized fast mod

In this section we show that transform methods can be used to compute mod operations, for sufficiently large fixed denominator N, in at most twice the time required for the general multiplication of two $(\mathrm{mod}\, N)$ integers. Such a situation obtains, for example, when N is a number to be factored, and each mod operation $x \pmod N$ proceeds for the fixed N, with $0 \leq x < N^2$. The mod problem can be reduced to successive evaluations of $\mathrm{Floor}(x/N)$. Given a $\mathrm{Floor}(\cdot)$ evaluation, one may perform mod operations simply according to

$$x \pmod N = x - N\,\mathrm{Floor}(x/N). \tag{10.1}$$

The idea is to compute and store a kind of reciprocal of the fixed denominator N. Let $2^b > N^2$ and define $c = \mathrm{Ceiling}(2^b/N) = 2^b/N + e$, where $0 \leq e < 1$. Then

$$xc/2^b = x/N + xe/2^b, \tag{10.2}$$

from which it follows that

$$\mathrm{Floor}(x/N) = \mathrm{Floor}(xc/2^b) - d, \tag{10.3}$$

where $d = 0$ or 1. Thus, a mod operation may be performed using

$$x \pmod N = x - N\,\mathrm{Floor}(xc/2^b) + dN. \tag{10.4}$$

In this way the generalized mod operation is brought down to multiplications xc, simple bit-shifting, and a multiplication by N, with a trivial adjustment required to determine whether $d = 0$ or 1. The appeal of this approach is that transforms of the digits of N and of c can be determined just once.

Algorithm for computing successive values $x \pmod N$ **for fixed** N, $0 \le x < N^2$.

(1) Choose b such that $2^b > N^2$, and compute $c = \text{Ceiling}(2^b/N)$, for example by a Newton method. Compute and store a transform **C** of the digits of c, and a transform **N** of the digits of N.

(2) For each successive x, compute a transform **X** of the digits of x; then use **C** and an inverse transform to determine xc and thus $z = \text{Floor}(xc/2^b)$.

(3) Obtain the transform **Z** of the digits of z, and use this with the **N**-transform to determine $x = x - Nz$.

(4) (Determination of d) If $x < 0$, set $x = x + N$.

Let us assume, as in §4, that general FFT multiplication of two $(\text{mod } N)$ integers takes asymptotic time $3T$, where T is the time for one relevant transform. The two transforms in step (2) will consume a total time of $4T$, since x contains about twice as many digits as does N. But in step (3), there will be two transforms each taking time T. Thus the algorithm should require approximate time $6T$, amounting to twice the time for a general multiplication. In some situations the CRT technique of §7 can be used to further reduce the time in step (2), since c will usually have about half as many digits as does x.

11. Results to date

In this final section we describe numerical results obtained by the authors and collaborators. Some of these results are merely empirical data, intended to aid future researchers in checking methods and programs.

(1) Our implementation of direct FFT multiplication for general integers, on a 68040-based NeXTstation, surpasses the efficiency of straightforward "grammar-school" multiplication when the magnitude of (roughly equal) integers to be multiplied crosses the region of $\sim$ 5000 bits. When arithmetic is to proceed modulo Fermat or Mersenne numbers, the crossover occurs at approximately 2000 bits per multiplicand, because of the advantages of DWT methods.

(2) The reader may have interest in an empirical assessment of the improvement to be realized with balanced digit representations. We computed the floating-point convolution errors for random squares $(\text{mod } 2^{524287} - 1)$, as one might do to decide whether M_{524287} is prime. On a 64-bit mantissa machine, using the direct method with zero-padding, we found that the error e_n of (4.4) has a typical value of 0.01 for standard representation, and 0.0001 for balanced representation.

(3) Methods described herein, together with the Elliptic Curve Method (ECM), enabled us to find two 19-digit factors of $F_{13} = 2^{2^{13}} + 1$, both verified by Wagstaff [20]. It is now known that

$$F_{13} = 2710954639361 * 2663848877152141313 * 3603109844542291969 * c,$$

where c is a 2417-digit composite.

(4) Implementation of transform multiplication and Mersenne-mod arithmetic has enabled us, using variants of the Pollard $(p-1)$ method, to determine factors

$$3085953375452873 \text{ of } M_{320213} = 2^{320213} - 1,$$
$$34013668352159 \text{ of } M_{500113} = 2^{500113} - 1,$$
$$13364077516908463 \text{ of } M_{500249} = 2^{500249} - 1.$$

(5) Using FFT multiplication and balanced digit representation for polynomial multiplication $(\mathrm{mod}\, p)$, Buhler et al. [2] found all irregular prime pairs $(p, 2k)$ for all primes $< 10^6$. The data establish the truth of Fermat's "Last Theorem" for all exponents less than one million. The method involves the polynomial multiplication described in §3, together with a technique for resolving Bernoulli numbers $(\mathrm{mod}\, p)$ by reciprocating huge polynomials $(\mathrm{mod}\, p)$. Error-correction techniques, used to augment floating-point transforms with $B = 65536$ in (4.7), proved sufficient to settle regularity criteria for even larger primes $p \sim 10^7$; for example, it is now known that 8388013 and 8388019 are regular primes, with each case requiring about 2 CPU hours on a 68040-based NeXTstation. The irregularity calculations are now being pressed in a systematic way for primes up to four million.

(6) In attempts to discover new Mersenne primes, a network group [18] has employed various DFT and DWT methods. In particular, using the DWT approach of §6, one may compute random squares $x^2 \ (\mathrm{mod}\, p)$, where p is the Slowinski prime $2^{216091} - 1$, in CPU times:

1 second, 68040 NeXTstation,
0.07 second, Connection Machine $8K$,
0.02 second, Cray YMP,

whereas the less efficient direct FFT method takes for example ~ 3 seconds on the NeXTstation and ~ 0.3 seconds on the Connection Machine. The factor of 3 or 4 improvement for the weighted transform approach is due primarily to the automatic gain of 2 arising from the halving of the requisite run length of the internal FFTs. No new Mersenne primes have been found as of this writing, although the entire region $430000 < q < 524288$ has been systematically resolved (no new Mersenne exponents were found).

(7) The current largest explicit prime, $2^{756839} - 1$, recently found by Slowinski and Gage [17] was verified by D. Smitley, J. Doenias, and one of the authors (REC) as indeed prime, in March 1992, at the request of the original discoverers.

At Slowinski's request, a new, even larger prime was verified, using DWT methods, by Doenias and Crandall; said prime to be announced.

(8) Whether the twenty-second Fermat Number F_{22} is prime is a question that is definitely accessible on machines of today, on the basis of the Fermat-mod techniques herein. We have programmed the appropriate Pepin test, in which about 2^{22} squares must be performed $(\mathrm{mod}\, F_{22})$. We find that Cray machines perform a typical such square in less than 1 second, even though we found the error-correction technique of §3 necessary. The entire primality test for F_{22} will consume roughly 600 CPU hours—not an unreasonable amount of time. We have not performed a full run on F_{22}, but at this point the issue

is plainly that of availability of machine cycles. We note that the composite character of F_{20} has been settled by direct FFT methods [21].

ACKNOWLEDGMENTS

We are indebted to Joseph Buhler, Department of Mathematics, Reed College, for his work on recursive large-GCD and other algorithms, and to Joshua Doenias of NeXT Computer, Inc., for his various program implementations of the weighted transform method. We wish to acknowledge the aid of Robert Silverman, Samuel Wagstaff, and Peter Montgomery on various factorization and verification issues. We are likewise indebted to our informal prime-search network group called "Gang-of-Nine" [18], out of which have emerged various ideas fundamental to this paper. We are grateful to a reviewer who indicated substantial improvements to the manuscript. Many of the empirical results herein were made possible by a grant from the San Diego Supercomputer Center.

BIBLIOGRAPHY

1. A. Aho, J. Hopcroft, and J. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA, 1974.
2. J. P. Buhler, R. E. Crandall, and R. W. Sompolski, *Irregular primes to one million*, Math. Comp. **59** (1992), 717–722.
3. D. Calvetti, *A stochastic roundoff error analysis for the Fast Fourier Transform*, Math. Comp. **56** (1991), 755–774.
4. K. Chen, *A New Record: The largest known prime number*, IEEE Spectrum **27** (1990), 47.
5. R. Creutzburg and M. Tasche, *Parameter determination for complex number-theoretic transforms using cyclotomic polynomials*, Math. Comp. **52** (1989), 189–200.
6. B. Fagin, *Large integer multiplication on hypercubes*, J. Parallel Distrib. Comput. **14** (1992), 426–430.
7. L. Leibowitz, *A simplified arithmetic for the Fermat number transform*, IEEE Trans. Acoust. Speech Signal Process. **24** (1976), 356–359.
8. W. Li and A. Peterson, *FIR filtering by the modified Fermat number transform*, IEEE Trans. Acoust. Speech Signal Process. **38** (1990), 1641–1645.
9. J. McClellan and C. Rader, *Number theory in digital signal processing*, Prentice-Hall, Englewood Cliffs, NJ, 1979.
10. P. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp. **48** (1987), 243–264.
11. H. Nussbaumer, *Fast Fourier and convolution algorithms*, Springer-Verlag, Heidelberg, 1982.
12. J. M. Pollard, *The fast Fourier transform in a finite field*, Math. Comp. **25** (1971), 365–374.
13. F. P. Preparata and D. V. Sarwate, *Computational complexity of Fourier transforms over finite fields*, Math. Comp. **31** (1977), 740–751.
14. G. U. Ramos, *Roundoff error analysis of the fast Fourier transform*, Math. Comp. **25** (1971), 757–786.
15. I. Reed and T. Truong, *The use of finite fields to compute convolutions*, IEEE Trans. Inform. Theory **21** (1975), 208–213.
16. A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281–292.
17. D. Slowinski and P. Gage, private communication (1992).
18. D. Smitley, R. Crandall, B. Fagin, W. Colquitt, R. Frye, J. Buhler, J. Doenias, D. Slowinski, and R. Silverman, *"Gang of Nine" network group for Mersenne prime search*, 1990–1992.

19. H. V. Sorenson et al., *Real-valued fast Fourier transform algorithms*, IEEE Trans. Acoust. Speech Signal Process. **35** (1987), 849–863.
20. S. Wagstaff, private communications, 1991.
21. J. Young and D. Buell, *The twentieth Fermat number is composite*, Math. Comp. **50** (1988), 261–263.

Scientific Computation Group, NeXT Computer, Inc., Redwood City, CA 94063

Thayer School of Engineering, Dartmouth College, Hanover, New Hampshire 03755