MATHEMATICS OF COMPUTATION
Volume 00, Number 0, Xxxx XXXX, Pages 000–000
S 0025-5718(XX)0000-0

RAPID MULTIPLICATION MODULO THE SUM AND DIFFERENCE OF HIGHLY COMPOSITE NUMBERS

COLIN PERCIVAL

ABSTRACT. We extend the work of Richard Crandall et al. [2] to demonstrate how the Discrete Weighted Transform (DWT) can be applied to speed up multiplication modulo any number of the form $a \pm b$ where $\prod_{p|ab} p$ is small. In particular this allows rapid computation modulo numbers of the form $k \cdot 2^n \pm 1$.

In addition, we prove tight bounds on the rounding errors which naturally occur in floating-point implementations of FFT and DWT multiplications. This makes it possible for FFT multiplications to be used in situations where correctness is essential, for example in computer algebra packages.

1. INTRODUCTION

In their seminal paper of 1994, Richard Crandall and Barry Fagin introduced the Discrete Weighted Transform (DWT) as a means of eliminating zero-padding when performing integer multiplication modulo Mersenne numbers [2]. While this does not give any improvement in the order of the multiplication, it nevertheless cuts the transform length (and thus time and memory) in half. For these reasons the DWT has become a fixture of the search for Mersenne primes [10].

By using the same form of irrational-base representation as is used for Mersenne numbers, we can in fact eliminate the zero-padding when working modulo $a \pm b$ provided that $\prod_{p|ab} p$ is sufficiently small that we have enough precision. Essentially, as with Mersenne numbers, we choose the base so that the reduction modulo $x^n - 1$ implicit in the FFT multiplication turns into a reduction modulo $a \pm b$.

As a result of decisions made in the design of modern processors, it is generally much faster to perform FFTs over the complex numbers using floating-point arithmetic than it is to perform them over finite fields. Consequently, it is very important to consider the size of errors which will result from using inexact floating-point arithmetic. Little work has been done here as far as error bounds go; in practice implementors have measured the size of errors for a set of random inputs, and have then chosen parameters which they believe to be 'safe'.

Although proven errors bounds are quite conservative in practice when dealing with random inputs, it can be demonstrated that there are some values for which the error comes quite close to the bound. Because of these 'unlucky' input values, proven error bounds should be quite useful in applications such as computer algebra packages where the correctness of multiplication is implicitly assumed.

1991 *Mathematics Subject Classification.* Primary 65G50, 65T50; Secondary 11A51.
Key words and phrases. Rapid multiplication, FFT, rounding errors.
This work was supported by MITACS and NSERC of Canada.

2. Multiplication modulo $x^N - 1$

The FFT has an interesting hybrid nature, in that it arises naturally in two distinct domains. Although it is best known from signal processing as a means of mapping between signal space and frequency space, the FFT also arises as the 'obvious' way of evaluating or interpolating a polynomial at the $N = 2^n$th roots of unity.

From these two viewpoints, the FFT multiplication can be seen as either a cyclic convolution of two signals (which is equivalent to the product of the corresponding polynomials modulo $x^N - 1$), or as a process of evaluating polynomials, multiplying the results pointwise, and interpolating the product polynomial, where the reduction modulo $x^N - 1$ results from the fact that $x^N - 1 = \prod_{w^N=1}(x - w)$. (For more details see [5].)

Since this paper is not intended to serve as a reference on how to perform FFTs rapidly, we will merely note that it is very easy to perform FFTs 'badly', and that a good implementation should use a radix appropriate to the processor and be very much aware of caches, page translation look-aside buffers (TLBs), and in the case of parallel or disk-based transforms, the physical location of the data. Much has been written about such implementation issues, and the interested reader is encouraged to consult [4] and [1].

For our purposes in this paper, we consider the multiplication $c(x) = a(x)b(x) \mod x^N - 1$ ($\sum c_i x^i = \sum a_i x^i \sum b_i x^i \mod x^N - 1$) to represent the process of computing $(A_i) = FFT((a_i))$, $(B_i) = FFT((b_i))$, $C_i = A_i B_i$, $(c_i) = FFT^{-1}((C_i))/N$, where FFT and FFT^{-1} represent the unnormalized FFT and the unnormalized inverse FFT respectively. We will give a (somewhat) more detailed description of the FFT process when we compute error bounds later.

Naturally, if we do not wish our answer to be reduced modulo $x^N - 1$, we need merely ensure that $N > \deg_x a(x) + \deg_x b(x)$, by 'padding' the polynomials with zeroes, so that the reduction has no effect. This gives a rapid polynomial multiplication in any field where the 2^nth roots of unity exist; in particular this includes the complex numbers and some finite fields.

Once we have a fast polynomial multiplication, it is simple to construct a fast integer multiplication; indeed, multiprecision integers are usually represented internally as polynomials in some convenient base (often a power of two or of ten), and the only additional work is to release the carries after the inputs have been multiplied as polynomials. This provides a simple yet effective fast integer multiplication.

A more sophisticated integer multiplication notes that the 'polynomials' being multiplied have coefficients over the real numbers, instead of the complex numbers. This allows for significant improvement, by either using a real-value FFT, or by utilizing the DWT to perform a right-angle convolution [2].

3. Multiplication modulo $a \pm b$

Richard Crandall and Barry Fagin define the Discrete Weighted Transform as the FFT of a weighted signal

$$DWT_{\mathbf{a}}(\mathbf{x}) = FFT(\mathbf{ax}) \tag{3.1}$$

where the juxtaposition of the signal $\mathbf{x}$ and the weighting vector $\mathbf{a}$ represents the componentwise product of the two. They then define the weighted convolution of two signals $\mathbf{x}$ and $\mathbf{y}$ in analogy with the (unweighted) cyclic convolution as

$$\begin{aligned} \mathbf{x} *^{\mathbf{a}} \mathbf{y} &= DWT_{\mathbf{a}}^{-1}(DWT_{\mathbf{a}}(\mathbf{x})DWT_{\mathbf{a}}(\mathbf{y})) \\ &= \mathbf{a}^{-1}FFT^{-1}(FFT(\mathbf{ax})FFT(\mathbf{ay})) \end{aligned} \tag{3.2}$$

Given these definitions, they observe that the cyclic, right-angle, and negacyclic convolutions of two signals can be computed as $x *^a y$ with $a_j = 1$, $a_j = e^{\pi i j/(2N)}$, and $a_j = e^{\pi i j/N}$ respectively.

Of particular interest is their observation that this can speed up multiplication modulo Mersenne numbers $M_q = 2^q - 1$. If we write numbers in the mixed-radix form

$$x = \sum_{j=0}^{N-1} \mathbf{x}_j 2^{\lceil qj/N \rceil} \tag{3.3}$$

then a weighted convolution with

$$\mathbf{a}_j = 2^{\lceil qj/N \rceil - qj/N} \tag{3.4}$$

will correctly multiply two numbers modulo M_q, leaving the product in the same form. This works because the input weighting transforms the inputs so that they have a fixed (irrational) radix, and the output weighting transforms them back. This 'irrational-base' DWT makes it possible to effect the modular squaring required by the Lucas-Lehmer primality test without requiring zero-padding, while still allowing use of convenient (i.e. highly composite) FFT lengths.

To multiply modulo $a - b$ in general, we can apply the same method. If the prime factorization of a and b are

$$a = p_1^{t_1} p_2^{t_2} \dots$$

$$b = q_1^{s_1} q_2^{s_2} \dots$$

then given numbers written in the mixed-radix form

$$x = \sum_{j=0}^{N-1} \mathbf{x}_j \prod p_i^{\lceil t_i j/N \rceil} \cdot \prod q_i^{\lceil -s_i j/N \rceil} \tag{3.5}$$

a weighted convolution with

$$\mathbf{a}_j = \prod p_i^{\lceil t_i j/N \rceil - t_i j/N} \cdot \prod q_i^{\lceil -s_i j/N \rceil + s_i j/N} \tag{3.6}$$

will correctly multiply two numbers in the given mixed-radix form, for the same reasons. A multiplication modulo $a + b$ follows the same method but adds another term in order to effect a negacyclic convolution.

This method, however, has its difficulties. Foremost among them is the problem of converting into and out of the given mixed-radix form. We know of no efficient solution to this problem in general. In special cases, however, the conversion is straightforward: If $b = 1$ and $a = k \cdot 2^n$ for some small k, the conversion, as with the Mersenne multiplication, is a matter of splitting the base 2 representation of the input into variable-sized pieces.

Further, for any number it is reasonably easy to convert in small numbers (0,1,2, etc.) and detect zero. As most primality testing algorithms currently in use require nothing more than the ability to input small numbers, multiply, and detect zero, we expect that this will prove useful for conducting tests on numbers of this special form.

Unfortunately, for precision reasons the product $\prod_{p|ab} p$ of prime numbers dividing a and b must be quite small; this will be discussed in detail after error bounds have been given later in this paper.

4. Previous FFT error estimates

While little has been done so far on proven error bounds, there have been a large number of error estimates. Indeed, most times that results derived by making use of FFT multiplications are reported, the authors will find it necessary to devote a few paragraphs to justifying the correctness of their multiplication, usually on the basis of a small number of lengths for which errors are computed.

In the paper where they introduce the DWT, Richard Crandall and Barry Fagin suggest, based on numerical work, that errors should be bounded by $O(N^{3/2} \log N)$ for standard (positive) digit representations, and by $O(N \log N)$ for balanced representations [2]. We note that these estimates are for random inputs; for worst-case inputs, the errors must be of the same order, as any input resulting from an integer expressed in positive representation can also result from an integer expressed in balanced representation with a slightly larger base. As we will see later, these estimates are pessimistic by a factor of $\sqrt{N}$.

It is suggested in a later paper that the maximum errors should be roughly $O(\sqrt{N \log N \log\log N})$ [3]. This is argued based on the hypothesis that the errors in the FFT resemble a random walk of length $N \log N$. This, in our opinion, is quite wrong: The FFT errors more closely approximate a random walk of length $\log N$ in an N-dimensional space. It is consequently somewhat surprising that the error estimate achieved is within a factor of $\sqrt{\log\log N}$ of correct.

Some rigorous work has nevertheless been done. In a paper which considers only the forward FFT, George Ramos derives upper bounds for the root-mean-square and maximum errors resulting from the FFT [9]. However, as he is considering the FFT as a signal-processing operation, where errors appear as added 'noise', he does not consider the relevance of these bounds to the correctness of FFT multiplication.

A more recent treatment is given in [7], but surprisingly the author reaches a bound which is suboptimal by a factor of $\sqrt{N}$. A slight modification to the proof can remove the extraneous factor.

Finally, Donald Knuth, in his *Seminumerical Algorithms* [8] gives an error bound for FFT multiplications implemented with fixed-point arithmetic. This bound is however extremely pessimistic, weighing in at roughly $O(N^6)$. As a result it is interesting only as a curiosity.

5. FFT error bounds

In contrast to the heuristic and statistical treatments discussed in the preceeding section, it is quite possible to prove a rigorous bound on the maximum error, as shown by the following theorem:

Theorem 5.1. *The FFT allows computation of the cyclic convolution $z = x * y$ of two vectors of length $N = 2^n$ of complex values such that*

$$|z' - z|_\infty < |x| \cdot |y| \cdot ((1+\epsilon)^{3n}(1+\epsilon\sqrt{5})^{3n+1}(1+\beta)^{3n} - 1)$$

where $|\cdot|$ and $|\cdot|_\infty$ denote the Euclidean and infinity norms respectively, ϵ is such that $|(a \pm b)' - (a \pm b)| < \epsilon|a \pm b|$, $|(ab)' - (ab)| < \epsilon|ab|$ for all machine floats a, b,

$\beta > |(w^k)' - (w^k)|$, $0 \le k < N$, $w = e^{\frac{2\pi i}{N}}$, and $(\cdot)'$ refers to the computed (stored) value of $\cdot$ for each expression.

Proof. We begin by giving error bounds for complex addition and multiplication. For addition we can trivially see that $|(z_0 \pm z_1)' - (z_0 \pm z_1)| < \epsilon \cdot |z_0 \pm z_1|$. For complex multiplication, we note that we can restrict ourselves to considering $z_j = a_j + b_j i$ with $0 < a_0, b_0, a_1, b_1$ and $a_0 a_1 \ge b_0 b_1$. Now considering the error in computing $a_0 a_1 - b_0 b_1$, we observe that if $2b_0 b_1 \ge a_0 a_1$ there is no error introduced by the subtraction [6]; further, if $2b_0 b_1 < a_0 a_1$ then the total error introduced in computing $b_0 b_1$ and performing the subtraction is bounded by $\epsilon(a_0 a_1 - b_0 b_1)$.

Combining these results with the straightforward bounds on other errors gives us that $|(a_0 a_1 - b_0 b_1)' - (a_0 a_1 - b_0 b_1)| < \epsilon \max(a_0 a_1 + b_0 b_1, 2a_0 a_1 - b_0 b_1)$ and $|(a_0 b_1 + b_0 a_1)' - (a_0 b_1 + b_0 a_1)| < \epsilon(2a_0 b_1 + 2b_0 a_1)$. From this we can readily obtain the bound $|(z_0 z_1)' - (z_0 z_1)| < \epsilon\sqrt{5} \cdot |z_0 z_1|$.

Now we observe that the FFT and inverse FFT each consist of n steps, each of which pairs off points and replaces each pair (z_0, z_1) with $(z_0 + wz_1, z_0 - wz_1)$ for various Nth roots of unity w.

Given the above, we see that

$$\begin{aligned} |(wz_1)' - (wz_1)| &\le \epsilon\sqrt{5} \cdot |w' z_1| + |w' - w| \cdot |z_1| \\ &\le |z_1| \cdot ((1 + \epsilon\sqrt{5})(1 + \beta) - 1) \end{aligned}$$

and thus

$$\begin{aligned} |(z_0 + wz_1)' - &(z_0 + wz_1)|^2 + |(z_0 - wz_1)' - (z_0 - wz_1)|^2 \\ &\le (|z_0 + wz_1|^2 + |z_0 - wz_1|^2)((1 + \epsilon)(1 + \epsilon\sqrt{5})(1 + \beta) - 1)^2 \end{aligned}$$

Since $(A_i) = FFT((a_i))$ and $(B_i) = FFT((b_i))$, we now have

$$|A' - A| \le |A| \cdot ((1 + \epsilon)^n (1 + \epsilon\sqrt{5})^n (1 + \beta)^n - 1)$$

and the corresponding relation for B.

Since C is the componentwise product of A and B, we can apply the Cauchy-Schwarz inequality to obtain

$$\sum |C_i' - C_i| \le |A| \cdot |B| \cdot ((1 + \epsilon)^{2n} (1 + \epsilon\sqrt{5})^{2n+1} (1 + \beta)^{2n} - 1)$$

and, observing that in the worst case these errors accumulate additively, while the errors introduced during the inverse FFT are of the same order as the errors introduced during the (forward) FFT, we see that

$$|z' - z|_\infty < |x| \cdot |y| \cdot ((1 + \epsilon)^{3n} (1 + \epsilon\sqrt{5})^{3n+1} (1 + \beta)^{3n} - 1)$$

as required. □

Note again that this is a worst-case bound, and in general the error will be lower. As a rough estimate of how much lower, we note that for random inputs the errors $|C_i' - C_i|$ will be of roughly the same size; while in the worst case these are all added together, in an average case they will cancel as in a random walk of length N to give an average error a factor of $\sqrt{N}$ lower.

We observe that on any IEEE 754/854 compliant processor, using normal double precision arithmetic (53 bit mantissa) we have $\epsilon = 2^{-53}$, and if the Nth roots of unity are 'perfect', that is, rounded from the correct (infinite precision) values, we can take $\beta = \epsilon/\sqrt{2}$, since the errors in sin and cos are each bounded by $\epsilon \cdot 2^{-1}$.

This gives us the useful benchmark that a straightforward multiplication of two degree 524288 polynomials with 'digits' between -5000 and 5000, given perfect trigonometric data, will always produce the correct answer using double precision arithmetic. This polynomial multiplication is equivalent to multiplying two 2 million (base ten) digit numbers.

We further observe that, for worst-case inputs, the largest element in $z = x * y$ can be as large as $|x| \cdot |y|$, so any implementation will have a worst case error of at least $\Omega(|x| \cdot |y| \cdot \epsilon)$. In this light our proven bound of $O(|x| \cdot |y| \cdot \epsilon \cdot n)$ is very reasonable.

6. DWT error bounds

Theorem 6.1. *The discrete weighted transform allows computation of the weighted convolution $\boldsymbol{z} = \boldsymbol{x} *^{\boldsymbol{a}} \boldsymbol{y}$ of two vectors with maximum error*

$$|z' - z|_\infty < |\boldsymbol{ax}| \cdot |\boldsymbol{ay}| \cdot ((1+\epsilon)^{3n}(1+\epsilon\sqrt{5})^{3n+4}(1+\beta)^{3n}(1+\delta)^3 - 1)$$

where ϵ and β are as defined earlier, $\delta \geq \max(|a'_k - a_k|/|a_k|)$, $\delta \geq \max((a_k^{-1})' - a_k^{-1})$ and $|a_k| \geq 1$ for all k.

Proof. We observe that $\mathbf{x} *^{\mathbf{a}} \mathbf{y} = \mathbf{a}^{-1}((\mathbf{ax}) * (\mathbf{ay}))$. Now we see that

$$|(\mathbf{ax})'_i - (\mathbf{ax})_i| \leq |(ax)| \cdot ((1+\epsilon\sqrt{5})(1+\delta) - 1)$$

and similarly for $\mathbf{z}$, and the cyclic convolution is computed via the FFT with the errors as given earlier.

The result follows. □

We observe that in effecting the right-angle convolution of two signals we have $\delta = \beta$ and $|(ax)| = |\mathbf{x}|$, so the maximum error is $|x| \cdot |y| \cdot ((1+\epsilon)^{3n}(1+\epsilon\sqrt{5})^{3n+4}(1+\beta)^{3n+3} - 1)$; consequently with double precision multiplication, accurate trigonometric data and a balanced representation the right-angle convolution can safely multiply 2 million digit integers in base 10^4, 96 million digit integers in base 10^3, and 8 billion digit integers in base 10^2. It is interesting to note that in the case of multiplying 2 million digit integers in base 10^4, the maximum error is lower when using a right-angle convolution than when using a straightforward multiplication, as the extra error incurred due to the DWT is dwarfed by the errors which would have been encountered in the extra pass of FFT which is eliminated by the length-halving inherent in the right-angle convolution.

We can also compute explicit limits for safe multiplication modulo Mersenne numbers using the DWT. For working modulo $2^m - 1$ with an FFT length of $N = 2^n$ complex values it is easy to show that $|\mathbf{ax}| < \sqrt{2^{n+m/N-1} 3/\log 4}$. This means that, given perfect weights and trigonometric data, FFTs of length 2^{18} complex values will be safe for m under 7 million; FFTs of length 2^{20} will be safe for m under 26 million. For comparison, the Great Internet Mersenne Prime Search [10] uses limits of about 10 million and 40 million for these same lengths. While this demonstrates that the bounds are conservative, it also provides a concrete example of a problem where using entirely 'safe' multiplications would only increase the run time by about 50%.

For the general case of multiplication modulo $a \pm b$ the bound is slightly more difficult, but in the worst case we still have $|\mathbf{ax}| < \sqrt{(ab)^{1/N} \cdot (P^2 - 1)/2 \log P^2}$, where $P = \prod_{p|ab} p$. In the case of numbers of the form $k2^m \pm 1$ the bound tightens slightly to $|\mathbf{ax}| < \sqrt{(k2^m)^{1/N} \cdot 3(k^2 - 1)/(2 \log k^2 \log 4)}$; this gives us that for $k = 3$,

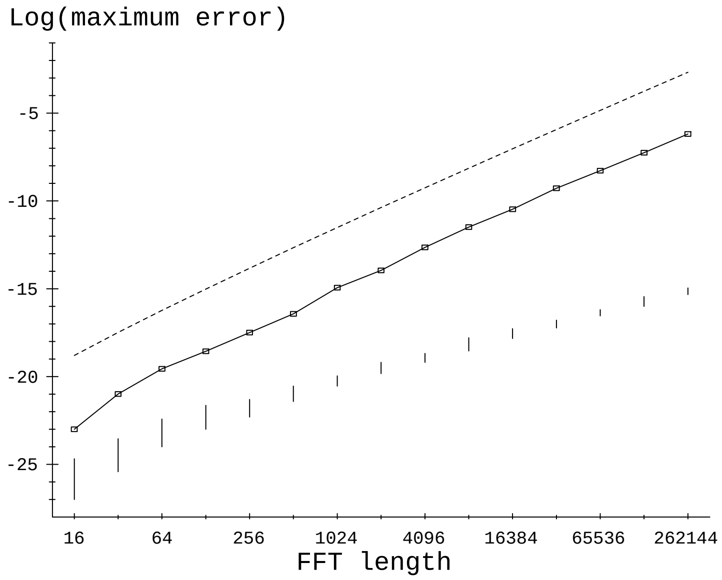

FIGURE 1. Maximum errors for various FFT sizes. The dotted line represents the proved error bound; the boxes represent the error resulting from 'unlucky' inputs; the vertical lines represent the range of maximum errors for various random inputs.

an FFT of length 2^{18} will suffice for m up to 6 million, and for $k = 557$, the same FFT suffices for m up to 3 million.

7. Experimental results

Error bounds are by their very nature pessimistic, and it is well worth investigating precisely how pessimistic our bounds are. This is especially true in light of the heuristic error estimates given in [3] which, while giving an expected error far below our proved bound (by a factor of roughly $\sqrt{N}$), appear to agree with numerical experiment.

We note however that these numerical experiments used essentially random inputs; they say nothing about the errors which might result if inputs were deliberately chosen to result in large errors.

One such input is $x_j \doteq A\cos(2\pi j/N) + iA\sin(2\pi j/N)$. This input results in large errors as a result of the distribution of the floating-point values after the FFT; whereas for random inputs the post-transform values are of roughly equal size, this input results in a single value being much larger than the rest, magnifying the error.

In Figure 1 we compare experimental errors for 100 multiplications of random polynomials with coefficients in $[-5000, 4999] + [-5000, 4999]i$, experimental errors for squaring a polynomial with the above 'unlucky' coefficients (with $A = 5000$), and our proved error bound for FFT lengths of 2^4 up to 2^{18}. The range of coefficients was chosen to match what would be used in a long integer arithmetic package which used a base 10^4 representation, but naturally the results would be equivalent for other bases.

Several points here are worth noticing. First, the range of maximum errors for multiplications of random inputs is quite small; consequently the maximum error is quite predictable, and might even be of use in verifying that no errors occurred during the multiplication.

Second, the errors for the atypical inputs are much larger than the random errors, and parallel the error bound. While there is a significant gap between these errors and the error bound, it remains almost constant as the FFT size ranges from 2^4 up to 2^{18}. Meanwhile, the gap between these and the errors for random inputs ranges from 1.6 bits up to 8.7 bits as the FFT length increases.

This clearly demonstrates that, while for some purposes choosing parameters based on the errors resulting from random inputs will suffice, any attempt to ensure that a multiplication will work for *all* inputs must pay attention to these atypical inputs.

8. Conclusions

On most problems, using FFT multiplications with a level of precision which is provably immune to fatal rounding errors will increase the run time by under a factor of two. Because of this, and because of the existence of 'unlucky' inputs values which result in errors far exceeding the norm, we strongly recommend that general purpose floating-point FFT multiplications are implemented in a manner immune to fatal rounding errors, by using FFT lengths based on the error bounds proved here.

Nevertheless there are a few cases where this is not advised; in large computations where machine error becomes a significant possibility, all computations must be duplicated to ensure correctness. If care is taken in this duplication to ensure that the multiplications are different (for example, in computing Lucas-Lehmer tests, the Great Internet Mersenne Prime Search performs double-checks by multiplying all values by a random power of two [10]), then there is no need to utilize multiplications which always produce the correct result. Another case where a 'safe' multiplication might be unnecessary would be in a probabilistic primality test; the unlikely possibility of a factor being missed would be far outweighed by the increased speed of using an unsafe multiplication.

However, these examples are few, and only the end-user of an arithmetic package could know how the multiplications are to be used; consequently any floating-point FFT multiplications in computer algebra packages or general long integer arithmetic packages should use provably correct multiplications.

9. Acknowledgements

The author wishes to thank Peter Borwein and Richard Crandall for their helpful comments and suggestions, and Paul Percival for his considerable assistance with the intricacies of LaTeX.

References

1. D.H. Bailey, *FFTs in External or Hierarchical Memory*, NAS report RNR-89-004, December 1989.
2. R. Crandall and B. Fagin, *Discrete weighted transforms and large integer arithmetic*, Math. Comp. **62** (1994), 305-324.
3. R.E. Crandall, E.W. Mayer, and J.S. Papadopoulos, *The twenty-fourth Fermat number is composite*, submitted to Math. Comp.

4. M. Frigo and S.G. Johnson, *FFTW*, http://www.fftw.org/ (2000).
5. K.O. Geddes, S.R. Czapor, and G. Labahn, *Algorithms for computer algebra*, Kluwer Academic Publishers, 1992.
6. David Goldberg, *What every compute scientist should know about floating-point arithmetic*, ACM Computing Surveys, **23**(1) (March 1991), 5-48.
7. N.J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, 1996.
8. D.E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 2nd edition, Addison-Wesley, 1997
9. G.U. Ramos, *Roundoff error analysis of the fast Fourier transform*, Math. Comp. **25** (1971), 757-786.
10. G. Woltman, *The Great Internet Mersenne Prime Search*, http://www.mersenne.org/ (2000).

Department of Mathematics and Statistics, Simon Fraser University, Burnaby, BC, Canada

E-mail address: cperciva@sfu.ca