Large Integer Multiplication on Hypercubes

Barry S. Fagin
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755
barry.fagin@dartmouth.edu

ABSTRACT

Previous work has reported on the use of polynomial transforms to compute exact convolution and to perform multiplication of large integers on a massively parallel processor. We now present results of an improved technique, using the Fermat Number Transform.

When the Fermat Number Transform was first proposed, word length constraints limited its effectiveness. Despite the development of multidimensional techniques to extend the length of the FNT, the relatively small word length of existing machines made the transform of little more than academic interest. The emergence of new computer architectures, however, has made the Fermat Number Transform more attractive. On machines like the Connection Machine, for example, we may implement the Fermat Number Transform without regard to word length considerations.

We present a convolution algorithm on a massively parallel processor, based on the Fermat Number Transform. We present some examples of the tradeoffs between modulus, interprocessor communication steps, and input size. We then discuss the application of this algorithm in the multiplication of large integers, and report performance results on a Connection Machine. Our results show multiplication times ranging from about 50 milliseconds for 2K-bit integers to 2 seconds for 8M-bit integers, an improvement of about 5 times over previous work.

1.0 Introduction

Several algorithms exist for the multiplication of large integers [10]. The straightforward technique requires $O(n^2)$ processors to execute in O(log n) time. Other more efficient algorithms, such as the Karatsuba and Toom-Cook methods, run much faster on a sequential machine: Karatsuba takes $O(n^{\lg 3})$ time, while Toom-Cook takes $O_{n2}(2 \lg n)^{(1/2)} \log n)$ [10]. These algorithms, while suitable for sequential machines, are not good candidates for parallel implementation. Karatsuba and Toom-Cook procedures use a divide and conquer approach, applying the same technique to smaller and smaller pieces of the input and then reassembling the results. This reassembly requires significant interprocessor communication time on a multiprocessor, and suggests an examination of other algorithms.

FFT-based algorithms are known to be good candidates for parallel implementation. These algorithms use transforms to compute the convolution products of two integer sequences, and then "unfold" the resulting sequence into a single multiplication product. The implementation of the FFT on multiprocessors has been studied in [7] and [8], reporting O(log n) running times if sufficient processing power is available. Two FFT-based parallel algorithms for multiplication were analyzed in [15]. Computing the FFT in the ring of complex numbers yields a running time of O(log n log log n) on an EREW PRAM, while computing in the ring of integers modulo $2^B + 1$ yields an O(log n) running time. This latter procedure is due to Schönhage and Strassen [17].

We present an algorithm related to Schönhage and Strassen's procedure, based on the Fermat Number Transform. Our procedure has the advantage of a straightforward, efficient implementation on a massively parallel hypercube. We also take advantage of the unique features of massively parallel processors to overcome the word-length restrictions associated with number theoretic transforms. We present performance results of our algorithm, and discuss directions for future work.

We assume the reader is familiar with parallel implementations of convolution transforms, in particular the FFT. We will use the terms "butterfly" and "twiddle factor", for example, without definition.

2.0 The Fermat Number Transform

The Fermat Number Transform, or FNT, of a sequence x of length N is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n]\, \alpha^{nk} \bmod M \qquad k = 0, 1, \ldots N-1$$

where $M = 2^B+1$, $B = 2^b$ for some positive integer b, α a primitive Nth root of unity mod M. M is the b^{th} *Fermat Number;* the transform is carried out in the b^{th} *Fermat Ring*. The inverse FNT is defined as

$$x[n] = 1/N \sum_{k=0}^{N-1} X[k]\, \alpha^{-nk} \bmod M \qquad k = 0, 1, \ldots N-1$$

These transforms have the convolution property [1], and can be used to multiply large integers under certain conditions.

There are well known difficulties in using a Fermat ring for computing digital convolution. For computing transforms in a ring modulo 2^B+1, the maximum possible transform length is 4B. Since B is the number of bits necessary to represent integers in the input sequences, it is usually the word length of the host computer. Most computers have words no larger than 32 bits, making the use of one dimensional Fermat Number Transforms impractical for large input sequences.

Agarwal and Burrus [2] present an algorithm for extending the transform length of the FNT through multidimensional techniques. Using two-dimensional transforms, a convolution of length N can be replaced with multiple convolutions of length $<= 2N^{1/2}$. The algorithm is as follows.

Let x and h be two sequences of length N = LM. Define the 2L x M arrays

$h'[i,j] = 0 \qquad i = 0$

$h'[i,j] = h[(i-L+Lj) \bmod N] \qquad i > 0$

$x'[i,j] = x[i+Lj]$ $\quad 0 <= i < L$

$x'[i,j] = 0$ $\quad L <= i <= 2L$

for $0 <= i < 2L$, $0 <= j < M$.

Let y' be the two-dimensional convolution of x' and h'; let y be the one-dimensional convolution of x and h. Aggarwal and Burrus show that y' and y are related by

$y[i+Lj] = y'[i+L,j]$ $\quad 0 <= i < L, 0 <= j < M$

Since y' is a 2L x M two dimensional convolution, y' may be computed using one dimensional convolutions of length <= 2L. For N a power of two, this is at most $2N^{1/2}$, a considerable reduction in the maximum transform length required.

Aggarwal and Burrus' algorithm generalizes to multiple dimensions in a straightforward manner; for a detailed discussion the reader is referred to [2]. We report performance results only for the two dimensional case, although we present some analysis for transforms of higher dimensionality.

In addition to transform length considerations, data representation concerns must be addressed. Arithmetic in a Fermat ring mod 2^B+1 on a machine with B-bit words is complicated by the necessity to represent 2^B+1 numbers 0, 1, ... 2^B with B bits. We adopt the technique of *1's reduced notation,* first proposed by Leibowitz [12]. A binary number I such that $1 <= I < 2^B$ is represented by I-1 in 1's reduced notation. 0 is represented by 2^B. This means that 0 can be identified by a 1 in the B+1st bit, and special procedures invoked to handle this as a special case. If neither operand is zero, multiplication by a power of two of a 1's reduced number is equivalent to a rotation and inversion of bits. Addition of non-zero operands is similar to binary addition, with the complement of the carry out added to the result. Since addition and multiplication when one or both of the operands are zero are trivial operations, the added complexity of this notation is minimal.

Finally, we note that the technological framework of computer design has evolved considerably since these ideas were first proposed. For new computer architectures like the Connection Machine, word length considerations are of reduced importance. Thus we may implement the Fermat Number Transform on current multiprocessors with much less emphasis on word length considerations. Other concerns, such as reducing the interprocessor communication time and the number of processing elements, become important.

3.0 Digital Convolution Using Multidimensional Fermat Number Transforms

We first present an analysis of the tradeoffs involved in using multidimensional FNT's to compute digital convolution. Let us assume two input sequences of total bit length N, which we may partition into digits of some size P >= 1 and convolve using the FNT. For ease of presentation, we assume N, P, and B are powers of two. Lower case letters will denote the base 2 logarithm of their upper case equivalents.

We begin with two sequences of N bits each. We wish to partition each sequence into P-bit digits, and then compute their convolution modulo $M = 2^B+1$ using a d-dimensional FNT. For a given N, we wish to understand the relationships between d, B, and P. First, in order for the convolution to be uniquely representable in the Fermat ring mod $M = 2^B+1$, the maximum value of a convolution product must be less than or equal to the largest integer in the ring. Since each digit contains P bits, the largest any single digit can be is 2^P-1. Each term in a convolution product can be no greater than $(2^P-1)^2$, and there are at most N/P terms contributing to a single convolution product. This implies that

$$(2^P-1)^2(N/P) <= 2^B \qquad (1)$$

Taking the log of both sides, we have

$$2 \log (2^P-1) + n - p <= B \qquad (2)$$

It would be convenient to approximate $\log(2^P-1)$ by P. Letting $\varepsilon = P - \log(2^P-1)$, we have

$$2(P-\varepsilon) + n - p <= B \qquad (3)$$

or equivalently

$$2P + n - p <= B + 2\varepsilon \qquad (4)$$

For P >= 1 ε is positive, and we may thus neglect it in (4). Replacing $\log(2^P-1)$ with P, then, gives

$$2P + n - p <= B \qquad (5)$$

Suppose we now employ a d-dimensional FNT to compute the convolution of the input sequences, d > 1. We continue the analysis presented in [2]. Although only 2-dimensional transforms are discussed, the process of extension to higher dimensions is straightforward. Employing a d-dimensional transform extends a 1-dimensional array into a d-dimensional hypercube, doubling the length along d-1 axes. Thus a sequence of length L digits is transformed into a d-dimensional cube of 2^{d-1}*L elements. For convenience, we assume L is a power of 2. The lengths of each axis are distributed as evenly as possible, again as powers of 2. To perform a 4-dimensional FNT on a sequence of 2^{10} elements, for example, would require a total of $2^3*2^{10} = 2^{13}$ processors, arranged in a 2^4 x 2^3 x 2^3 x 2^3 4-cube.

Let n-p, the logarithm of the sequence length in digits, be congruent to r mod d. We distinguish 3 cases: r = 0, r = 1, and r >= 2. If r = 0, there will be d-1 long axes with log length equal to ∈ (n-p)/d _+1, and one short axis of length (n-p)/d. If r = 1, all axes are of log length ∈ (n-p)/d _ + 1; in this case we say the transform is *balanced.* If r >= 2, d-1 axes must have their lengths doubled, but d-r of them can be doubled without increasing the longest axis length. Any axes remaining must have their lengths doubled. Thus d-1-(d-r) = r-1 long axes will have log length ∈(n-p)/d_+2, and d-r+1 axes will have log length ∈(n-p)/d_+1.

For example, suppose that n-p = 10, corresponding to a sequence of 1K digits, and that d=4. Then r = n-p mod d = 2. We can split 2^{10} into 2^3 x 2^3 x 2^2 x 2^2. Three axis lengths will be doubled by the transformation, giving 2^4 x 2^3 x 2^3 x 2^3. We see that r-1 = 1 axis has log length = ∈(n-p)/d_+2 = 4, and that d-r+1 = 3 axes have log length = ∈(n-p)/d_+1 = 3. If instead n-p=8, we split 2^8 into 2^2 x 2^2 x 2^2 x 2^2, and double three axis lengths to obtain 2^3 x 2^3 x 2^3 x

2^2. r=0, and we see that d-1 = 3 axes have log length 8/4 + 1 = 3, and that one axis has log length 2.

A multidimensional FNT is similar to other multidimensional transforms; it is computed by computing 1-dimensional transforms along each dimension. Thus B must be large enough for an FNT of length 4B to cover any side of the cube. Depending on the congruence class of n-p mod d, we have either

$$4B = 2^{(\in (n-p)/d\_+2)} \quad (n-p \bmod d >= 2) \qquad (6)$$

or

$$4B = 2^{(\in (n-p)/d\_+1)} \quad (n-p \bmod d \_ 0, 1) \qquad (7)$$

Using logarithmic notation, these become

$$b = \in (n-p)/d\_ \quad (n-p \bmod d >= 2) \qquad (8)$$

and

$$b = \in (n-p)/d\_ -1 \quad (n-p \bmod d \_ 0, 1) \qquad (9)$$

Equations (5), (8), and (9) give us the defining relationships between n,p,d, and b. For a given N and d, we may examine the various tradeoffs involved in digit size, interprocessor communication, and the number of processors required. For a detailed description of these tradeoffs, the reader is referred to [5].

4.0 Performance on a Massively Parallel Processor

The FNT and other number-theoretic transforms can be used for any application in which convolution is required. They are particularly well suited for the computation of exact convolution, and hence are useful for the multiplication of large integers. We have implemented a program to multiply large integers on the Connection Machine, using the parallel FNT.

The Connection Machine is a massively parallel SIMD processor, with processing elements interconnected in a boolean n-cube. CM software provides support for "virtual processors", permitting programs to be written for a number of PEs greater than those physically available.

When the number of virtual processors exceeds the number of physical processors, the system is said to be saturated. Further increases in the number of virtual processors result in a corresponding decrease in execution time, although this can be affected by interprocessor communication patterns. For more information on the Connection Machine, the reader is referred to [18] and [19].

Figure 1 shows a plot of multiplication time as a function of input size, for a 2D FNT-based algorithm using balanced transforms with $\alpha=2$. The values of b mandated by a 1-D transform would make the point-by-point multiplication performance unacceptable, while transforms with d > 2 would saturate the system with processing elements. The results shown are for a 32K Connection Machine. Values of n corresponding to n-p+1 > 15, in this case n >= 17, would execute in approximately half the time on a 64K CM. Execution times range from approximately 50 milliseconds for 4Kbit numbers to 2 seconds for 8Mbit numbers.

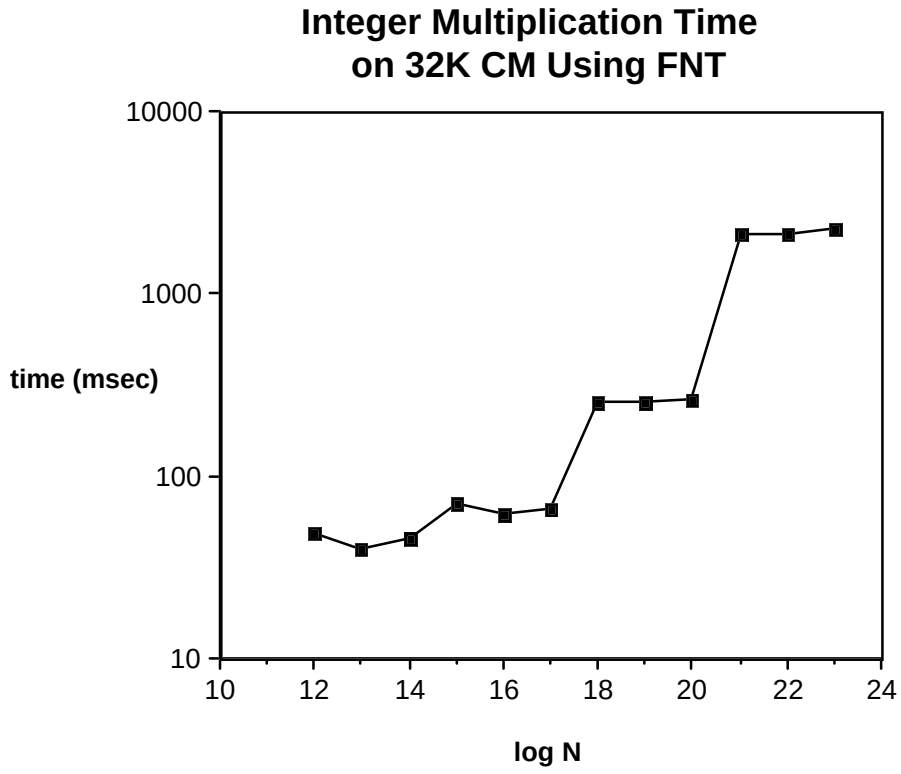
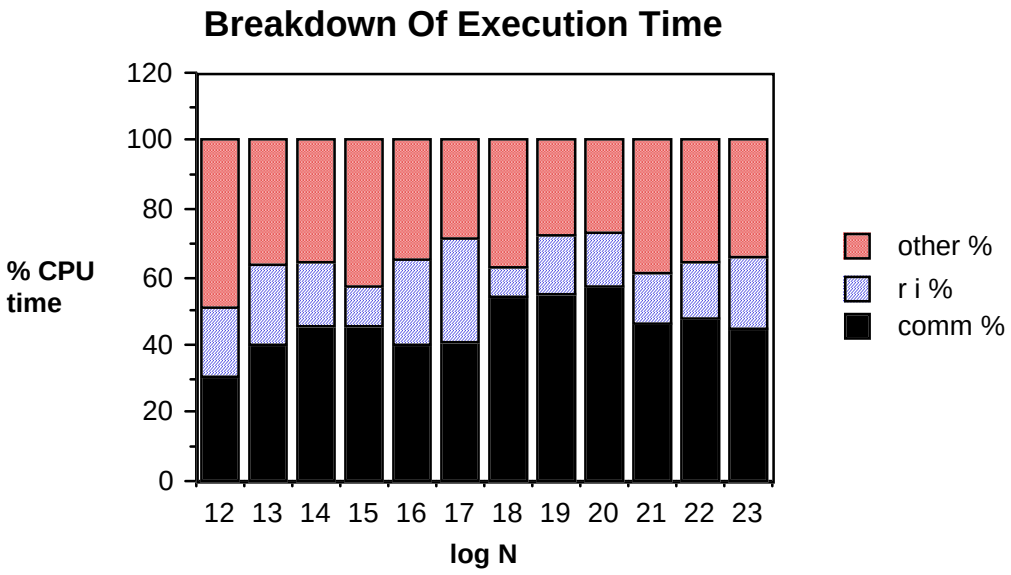


Figure 1: Integer Multiplication Time

We see that the execution time is a jagged function of n. This is because the dominant factor in execution time is not the input size in bits but the number of digits. In many cases a doubling of the input size does not result in any increase in the number of digits, resulting in similar execution times. The log of number of PE's required to multiply two 2^n-bit numbers is given by the equations of the preceding sections as $2 \in n/3\_ + 4$. Since the number of communication steps also grows linearly, the execution time scales linearly in the input size once system saturation is reached.

Figure 2 shows a proportional breakdown of execution time, listing communication time, rotation and inversion time, and other operations. Approximately 45% of execution time is spent in communication, 20% in rotation and inversion of digits, and 35% in all other operations.



The largest numbers our program can multiply are 8Mbits wide. Larger numbers exceed the memory limitations of a 32K CM, due to the number of processing elements required.

We have reported in previous work [4] our implementation of another integer convolution algorithm on the Connection Machine. The results reported here represent an improvement of 6 times for 2K numbers, decreasing to about 4 times for 8M numbers. The decrease in performance improvement is due to system saturation, which dampens the performance effects of otherwise more efficient algorithms.

Kubina and Wunderlich [11] have implemented a variant of the Schönhage-Strassen algorithm on the Connection Machine to test Waring's Conjecture. Their code can handle larger inputs than that describe here, but appears to be considerably slower. Performance comparisons are difficult, as [11] reports only a single performance point: about 520 seconds on a 64K CM to multiply 2^{28}-bit integers. Additionally, our program requires the entire input to reside in physical memory, whereas that of [11] employs disk storage to increase the maximum input size. We propose to incorporate disk storage into future versions of our code.

7.0 Conclusions

The Fermat Number Transform was proposed several years ago as an alternative to the FFT, but not pursued extensively due to transform length restrictions. For transforms in a Fermat ring modulo 2^B+1, the longest transform length is 4B. Since b = log B is limited by the word length of the machine computing the transform, existing computers were limited to short sequences. Multidimensional techniques were proposed to overcome this restriction, and were partially successful.

The existence of computers like the Connection Machine, however, make such restrictions obsolete. The CM does not make use of the concept of "word length"; operations can be performed on bit quantities of virtually arbitrary size. The CM is particularly attractive for the FNT due to the parallel nature of the algorithm.

Our results show that while word length restrictions no longer limit the length of sequences to be used as inputs to the FNT, the parameter b = log B remains important. The FNT convolution algorithm requires the point-by-point multiplication of B-bit numbers; if b becomes too large, point-by-point multiplication time will dominate. For small b, the number of processors and the number of interprocessor communication steps become the principal factors affecting performance.

To test these and other ideas, we implemented an algorithm for the multiplication of large integers using the parallel FNT on the Connection Machine. Our times range from about 50 msec for 2K-bit numbers to 2 seconds for 8Mbit numbers. These results are about 5 times faster than our previous work, an implementation of another integer convolution algorithm on the Connection Machine [4].

Future work should focus on reducing interprocessor communication requirements, perhaps through the use of alternative CM configurations or lower-level CM instructions than those currently employed by our program. Point-by-point multiplication time could also be improved through the application of any of the well-known efficient algorithms for integer multiplication. Extended-precision FFTs for the CM could also be developed and compared with the parallel FNT to determine which algorithm is more suitable for large integer multiplication. Comparisons with other high-performance architectures would also be useful. Presently, we are combining the program discussed here with previous work [3] to develop a massively parallel software library for the arithmetic manipulation of large integers.

8.0 Acknowledgements

The author gratefully acknowledges the use of the Connection Machine Network Server Pilot Facility, supported under terms of DARPA contract DACA76-88-C-0012 and made

available to the author by Thinking Machines Corporation. Thanks are due in particular to TMC employees Roger Frye and David Gingold for their patience and assistance.

9.0 References

[1] Agarwal, R.C. and Burrus, C.S. "Fast Convolution Using Fermat Number Transforms With Applications to Digital Filtering", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol ASSP-22 no 2 pp 87-97 Apr 1974.

[2] Agarwal, R.C. and Burrus, C.S. "Fast One-dimensional Digital Convolution by Multidimensional Techniques", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol ASSP-22 pp 1-10 Feb 1974.

[3] Fagin, Barry, "Fast Addition of Large Integers", submitted to IEEE Transactions on Computers, 1989. (Copies available from the author).

[4] Fagin, Barry, "Negacyclic Convolution Using Polynomial Transforms on Hypercubes", to appear in IEEE Transactions on Acoustics, Speech and Signal Processing.

[5] Fagin, Barry "Large Integer Multiplication Using the Fermat Number Transform", Technical Report, Thayer School of Engineering, Dartmouth College, Hanover NH 03755.

[6] Hillis, W. and Steele G, "Data Parallel Algorithms", *Communications of the ACM* 29,12 (Dec 1986), pp 1170-1183.

[7] Jamieson, Leah et. al., "FFT Algorithms for SIMD Parallel Processing Systems", *Journal of Parallel and Distributed Computing*, Volume 3, No. 1, pp 48-71, March 1986.

[8] Johnsson, S. et al., "Computing Fast Fourier Transforms on Boolean Cubes and Related Networks", Technical Report NA87-3, Thinking Machines Corporation, Cambridge, Mass.

[9] Johnsson, S. and Ho, Ching-Tien, "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Transactions on Computers*, Vol. 38, No. 9, September 1989.

[10] Knuth, Donald, The Art of Computer Programming, Vol. 2, Addison-Wesley, Reading, Mass, 1981.

[11] Kubina, J. and Wunderlich, M., "Extending Waring's Conjecture to 471,600,000",

Mathematics of Computation, Volume 55, Number 192, October 1990, pp 815-820.

[12] Leibowitz, L.M. "A Simplified Binary Arithmetic for the Fermat Number Transform", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-24, pp 356-359, October 1976.

[13] Lin, Xinming et. al, "The Fast Hartley Transform on the Hypercube Multiprocessors", Technical Report CSD-880011, Computer Science Department, University of California, Los Angeles.

[14] Nussbaumer, Henri "Fast Polynomial Transform Algorithms for Digital Convolution", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. ASSP-28, No. 2, April 1980.

[15] Peng, Shie-Tung and Hudson, Thomas, "Parallel Algorithms For Multiplying Very Large Integers", Proceedings of the 1988 International Conference on Parallel Processing, 1988.]

[16] Saad, Youcef and Schultz, Martin, "Data Communication in Hypercubes", *Journal of Parallel and Distributed Computing,* Volume 6, No. 1, pp 115-135, February 1989.

[17] Schönhage, Arnold and Strassen, Volker, "Schnelle Multiplikation großer Zahlen", *Computing 7,* pp 281-292, 1971.

[18]Thinking Machines Corporation, Connection Machine Technical Summary, Cambridge Massachusetts,1989.

[19] Thinking Machines Corporation, C/Paris 5.1 User's Manual, Cambridge Massachusetts, 1988.