FAST ALGORITHMS FOR MATRIX MULTIPLICATION USING PSEUDO-NUMBER-THEORETIC TRANSFORMS

Andrew E. Yagle

Dept. of EECS, The University of Michigan, Ann Arbor, MI 48109-2122 aey@eecs.umich.edu

ABSTRACT

We provide a novel approach to the design of fast algorithms for matrix multiplication. The operation of matrix multiplication is reformulated as a convolution, which is implemented using pseudo-number-theoretic transforms. Writing the convolution as multiplication of polynomials evaluated off the unit circle reduces the number of multiplications without producing any error, since the (integer) elements of the product matrix are known to be bounded. The new algorithms are somewhat analogous to the arbitrary precision approximation (APA) algorithms, but have the following advantages: (1) a simple design procedure is specified for them; (2) they do not suffer from roundoff error; and (3) reasons for their existence is clear. The new algorithms are also non-commutative, so that they may be applied recursively to block matrix multiplication. This work establishes a link between matrix multiplication and fast convolution algorithms and so opens another line of enquiry for the fast matrix multiplication problem. Some numerical examples illustrate the operation of the new proposed algorithms.

1. INTRODUCTION

1.1. Overview

Ever since Strassen's [1] observation that the product of two 2×2 matrices may be computed using only seven multiplications (less than the obvious eight), the problem of determining the minimum number of multiplications needed to multiply matrices and the design of fast algorithms for multiplying matrices have both been active research areas. Most workers have approached the problem using the theory of bilinear and trilinear forms. We do not attempt to provide a summary of all of the work done on this problem; the review paper [2] of Pan provides a good introduction.

This paper approaches the problem from a different perspective, viz., the theory of fast algorithms for convolutions. In doing so, we open up another line of enquiry into the problem of designing algorithms for fast matrix multiplication, and provide different perspectives on the problem. Linking these two problems allows result from one area to be applied to problems in the other area.

Our approach is as follows. First, we reformulate matrix multiplication as a linear convolution, which can be implemented as the multiplication of two polynomials using the z-transform. Second, we scale the variable z, producing a scaled convolution, and make it cyclic. This aliases some quantities, but they are separated by a power of the scaling factor. Third, we compute this scaled convolution using pseudo-number-theoretic transforms. Finally, the various components of the product matrix are read off of the convolution, using the fact that the elements of the product matrix are bounded. This can be done without error if the scaling factor is sufficiently large.

This work was supported by the National Science Foundation under Presidential Young Investigator Grant #MIP-8858082

1.2. Relevant Previous Work: APA Algorithms

Previous work closest in spirit to ours is the arbitrary precision approximation (APA) algorithms first proposed by Bini in [3]. For comparison to our work, we review an APA algorithm from [3] for multiplying two 2×2 matrices A, B whose $(i,j)^{th}$ elements are $a_{i,j}$ and $b_{i,j}$, respectively. Define

$$p_1 = (a_{2,1}+\epsilon a_{1,2})(b_{2,1}+\epsilon b_{1,2}); p_2 = (-a_{2,1}+\epsilon a_{1,1})(b_{1,1}+\epsilon b_{1,2});$$

$$p_3 = (a_{2,2} - \epsilon a_{1,2})(b_{2,1} + \epsilon b_{2,2}); \quad p_4 = a_{2,1}(b_{1,1} - b_{2,1});$$

$$p_5 = (a_{2,1} + a_{2,2})b_{2,1} \qquad (1a)$$

and compute the elements $c_{i,j}$ of $C = AB$ by

$$c_{1,1} = (p_1 + p_2 + p_4)/\epsilon - \epsilon(a_{1,1} + a_{1,2})b_{1,2}; \quad c_{2,1} = p_4 + p_5;$$

$$c_{2,2} = (p_1 + p_3 - p_5)/\epsilon - \epsilon a_{1,2}(b_{1,2} - b_{2,2}). \qquad (1b)$$

If we now let $\epsilon \to 0$, the second terms in (1b) become negligible next to the first terms, and so they need not be computed. Hence three of the four elements of $C = AB$ may be computed using only five multiplications. $c_{1,2}$ may be computed using a sixth multiplication, so that in fact two 2×2 matrices may be multiplied to arbitrary accuracy using only six multiplications. This is even better than the result of Strassen [1]. A Strassen-like algorithm for 3×3 matrix multiplication requires 23 multiplications, while an APA algorithm only requires 21. Greater improvements occur for larger matrices.

We make the following observations about (1):

1. multiplicands are sums of scaled matrix elements;
2. error can be made arbitrarily small, at the price of increasing the bit length of the multiplicands;
3. roundoff error makes the computations unstable;
4. (1) and the reason why it exists are not obvious.

A design methodology for fast matrix multiplication algorithms by grouping terms has been proposed in a series of papers by Pan (see references of [2]). While this has proven quite fruitful, it has two drawbacks: (1) at some point the

0-7803-2431-5/95 $4.00 © 1995 IEEE

methodology of grouping terms necessarily becomes somewhat ad hoc, i.e., it requires some user judgment during the design; and (2) it is not clear from this why these fast algorithms exist at all.

1.3. Advantages of New Approach

Our approach yields algorithms that require the same number of multiplications or fewer as APA for 2×2 and 3×3 matrices. The multiplicands are again sums of scaled matrix elements, as in APA.

However, our approach has the following advantages:

1. no errors, since the matrix products are bounded;
2. no unstable computations like (1b);
3. the design methodology is quite simple;
4. the reason why the fast algorithm exists is now clear.

Like the Strassen, Pan, and APA algorithms, our algorithm is *non-commutative*: it does not require that multiplication be commutative. This is a very significant feature, since non-commutative algorithms may be applied to block matrices. For example, multiplication of two $2^n \times 2^n$ matrices has the same form as multiplication of two 2×2 matrices, except that elements are now $2^{n-1} \times 2^{n-1}$ blocks. The Strassen algorithm can be applied directly, even though matrix multiplication does not commute, since the Strassen algorithm does not rely on commutativity of multiplication. Each multiplication in the Strassen algorithm is now a multiplication of two $2^{n-1} \times 2^{n-1}$ matrices, to which the Strassen algorithm can again be applied. $2^n \times 2^n$ matrix multiplication thus requires only 7^n multiplications, and $N \times N$ matrix multiplication requires $O(N^{\log_2 7}) = O(N^{2.807})$ multiplications.

Use of APA or our algorithms reduces this to $O(N^{\log_2 6}) = O(N^{2.585})$. However, since our approach allows relatively simple design for any size matrix, we can multiply two $N \times N$ matrices, where $N = 2^a 3^b 5^c \ldots$, by first using the 2×2 algorithm a times, then using the 3×3 algorithm b times, and so on. This maximum efficiency is better than the above bounds.

We assume throughout that all matrix entries have been scaled to integers. We focus computation counts on multiplications, rather than additions, for the following reasons:

1. although recent chips can perform multiplications in the same number of clock cycles as additions require, they require extensive chip area to do so;
2. multiplication is inherently more complex than addition;
3. in the above nesting multiplications accumulate faster.

2. DERIVATIONS OF ALGORITHMS

2.1. Matrix Multiplication as Convolution

We reformulate the product of two $N \times N$ matrices as the linear convolution of a sequence of length N^2 and a sparse sequence of length $N^3 - N + 1$. This results in a sequence of length $N^3 + N^2 - N$, from which elements of the product matrix may be obtained. For convenience, we write the linear convolution as the product of two polynomials:

THEOREM. Let A, B, and $C = AB$ all be $N \times N$ matrices with elements $\{a_{i,j}\}$, $\{b_{i,j}\}$, and $\{c_{i,j}\}$, respectively. Define sequences

$$a_{i,j} = a_{i+jN}; \quad b_{i,j} = b_{N-1-i+jN}; \quad 0 \le i,j \le N-1. \tag{2}$$

The matrix multiplication $C = AB$ is computed by the polynomial multiplication $\sum_{i=0}^{N^3+N^2-N-1} c_i x^i =$

$$\left(\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_{i+jN} x^{i+jN}\right)\left(\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} b_{N-1-i+jN} x^{N(N-1-i+jN)}\right), \tag{3}$$

where the elements of C are read off of the $\{c_i\}$ computed in (3) using

$$c_{i,j} = c_{N^2-N+i+jN^2}; \quad 0 \le i,j \le N-1. \tag{4}$$

PROOF. The coefficient of $x^{N^2-N+i_3+j_3N^2}$ in (3) is the sum of the products of the coefficients of $x^{i_1+j_1N}$ in the first polynomial and the coefficients of $x^{N(N-1-i_2+j_2N)}$ in the second polynomial such that $0 \le i_k, j_k \le N-1; 1 \le k \le 3$

$$(i_1+j_1N)+N(N-1-i_2+j_2N) = N^2-N+i_3+j_3N^2. \tag{5}$$

The solution to (5) is readily seen to be

$$i_1 = i_3; \quad j_2 = j_3; \quad 0 \le j_1 = i_2 \le N-1. \tag{6}$$

Comparing (2), (4), (6) shows (3) implements $C = AB$.

The following comments are appropriate here:

1. The degrees of the polynomials multiplied in (3) are N^2-1 and $N(N^2-1)$, and the degree of the resulting polynomial is $N^3 + N^2 - N - 1$;
2. The coefficients of all three polynomials are read off of the matrices A, B, and C column-by-column. Each column of B is reversed in order, i.e., read off bottom-to-top, rather than top-to-bottom. Alternatively, rows rather than columns may be used, by implementing $C^T = B^T A^T$;
3. The multiplication is non-commutative since a function of the elements of A is multiplied by a function of the elements of B; there is no mixing. Hence we can divide-and-conquer as in Section 1.3.

EXAMPLE #1. The 2×2 matrix multiplication

$$\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} = \begin{bmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{bmatrix}$$

is implemented by the polynomial multiplication (equivalent to a convolution)

$$\left(a_{0,0} + a_{1,0}x + a_{0,1}x^2 + a_{1,1}x^3\right)\left(b_{1,0} + b_{0,0}x^2 + b_{1,1}x^4 + b_{0,1}x^6\right)$$
$$= *+*x+c_{0,0}x^2+c_{1,0}x^3+*x^4+*x^5+c_{0,1}x^6+c_{1,1}x^7+*x^8+*x^9, \tag{7}$$

where $*$ denotes an irrelevant quantity.

EXAMPLE #2. The 3×3 matrix multiplication

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix}\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} \\ c_{1,0} & c_{1,1} & c_{1,2} \\ c_{2,0} & c_{2,1} & c_{2,2} \end{bmatrix} \tag{8}$$

is implemented by the polynomial multiplication (equivalent to a convolution)

$$[a_{0,0}+a_{1,0}x+a_{2,0}x^2+a_{0,1}x^3+a_{1,1}x^4+a_{2,1}x^5+a_{0,2}x^6$$
$$+a_{1,2}x^7+a_{2,2}x^8]\times[b_{2,0}+b_{1,0}x^3+b_{0,0}x^6+b_{2,1}x^9+b_{1,1}x^{12}$$
$$+b_{0,1}x^{15}+b_{2,2}x^{18}+b_{1,2}x^{21}+b_{0,2}x^{24}]=\ldots c_{0,0}x^6+c_{1,0}x^7$$
$$+c_{2,0}x^8\ldots c_{0,1}x^{15}+c_{1,1}x^{16}+c_{2,1}x^{17}\ldots c_{0,2}x^{24}+c_{1,2}x^{25}\ldots$$

2.2. Derivation of Algorithm Analogous to APA

In the sequel we focus on 2×2 matrices for clarity; extensions to larger matrices will be clear. In the polynomial multiplication (7), substitute $x=sz$ where s is a scaling factor to be chosen. Render the resulting convolution cyclic by taking the result $mod(z^6-1)$. The result is

$$[a_{0,0}+a_{1,0}sz+a_{0,1}s^2z^2+a_{1,1}s^3z^3][(b_{1,0}+b_{0,1}s^6)+b_{0,0}s^2z^2$$
$$+b_{1,1}s^4z^4]=(*+c_{0,1}s^6)+(*s+c_{1,1}s^7)z+(c_{0,0}s^2+*s^8)z^2$$
$$+(c_{1,0}s^3+*s^9)z^3+*z^4+*z^5;\quad mod(z^6-1)\qquad(10)$$

where $*$ again denotes an irrelevant quantity.

Now choose the scaling factor s such that $|c_{i,j}|<s^6$ and $|*|<s^6$ for each $*$ in (10). Then the $*$ and $c_{i,j}$ may be separated from each other without error, since both are known to be integers. Indeed, if s is a power of two, $c_{0,1}$ may be obtained by simply discarding the $6\log_2 s$ least significant bits in the binary representation of $*+c_{0,1}s^6$.

The significance of (10) is that the polynomial multiplication $mod(z^6-1)$ is equivalent to a 6-point cyclic convolution, which can be computed using pseudo-number-theoretic transforms using only six multiplications. Hence the 2×2 matrix multiplication may be implemented using only six multiplications. Doing the same thing to the polynomial multiplication (9) shows that 3×3 matrices may be multiplied using only 21 multiplications.

This first algorithm is reminiscent of the APA algorithms in the following ways:

1. the number of multiplications is the same as for APA;
2. sums of scaled matrix elements are multiplied in (10);
3. results are sums in which one quantity is neglected or partitioned from another which is of no interest.

The advantages of the new algorithm are as noted in Section 1.3. However, this is somewhat unfair to the APA algorithms–it is clear that this first algorithm is analogous to the APA algorithms. The main difference is that it uses integer quantities that may be separated from each other, rather than floating-point quantities for which one must be neglected compared to the other.

2.3. Algorithms with Fewer Multiplications

Because of this difference, it is clear that a cyclic convolution of order less than six may be used. Imposing $mod(z^5-1)$ on (7) with $x=sz$ results in

$$[a_{0,0}+a_{1,0}sz+a_{0,1}s^2z^2+a_{1,1}s^3z^3][b_{1,0}+b_{0,1}s^6z+b_{0,0}s^2z^2$$
$$+b_{1,1}s^4z^4]=(*+*s^5)+(*s+c_{0,1}s^6)z+(c_{0,0}s^2+c_{1,1}s^7)z^2$$
$$+(c_{1,0}s^3+*s^8)z^3+(*s^4+*s^9)z^4;\quad mod(z^5-1)\qquad(11)$$

which can be computed using pseudo-number-theoretic transforms using only five multiplications. The scaling factor s must be slightly larger, since now we need $|c_{i,j}|<s^5$ and $|*|<s^5$ to ensure separability of $c_{i,j}$ and $*$.

It might seem that cyclic convolutions of even smaller order might well be used. However, consider the effect of imposing $mod(z^4-1)$ on (7) with $x=sz$. The constant term is now $(*+*s^4+*s^8)$, which requires many more bits to store than $(*+*s^5)$. Note that this term is not excessively large in magnitude, as compared to $(c_{1,0}s^3+*s^8)$ for example, but the storage requirement, in terms of bits required, is half again as large. While these issues do not seem to have been addressed previously in the fast matrix multiplication literature, it is clear that they do become significant. Similar comments apply to the APA algorithms; consider bitlengths of the multiplicands in (1) as $\epsilon\to 0$.

2.4. Algorithm with One Multiplication

As an extreme case, consider a one-point (!) cyclic convolution, implemented by imposing $mod(z^1-1)$ on (7) with $x=sz$. This amounts to setting $z=1$, giving

$$\left(a_{0,0}+a_{1,0}s+a_{0,1}s^2+a_{1,1}s^3\right)\left(b_{1,0}+b_{0,0}s^2+b_{1,1}s^4+b_{0,1}s^6\right)$$
$$=*+*s+c_{0,0}s^2+c_{1,0}s^3+*s^4+*s^5+c_{0,1}s^6+c_{1,1}s^7+*s^8+*s^9.\qquad(12)$$

Thus a 2×2 matrix multiplication can be implemented using a single multiplication! The only requirement is that $|c_{i,j}|<s$ and $|*|<s$ to ensure separability of everything in the product. Again, this can be done by simply segmenting a base-s representation of the result of (12).

However, the numbers being multiplied here are massive (note how much bigger s must be). Perhaps (12) is impractical, although there are many FFT-based algorithms for multiplying large numbers and the complexity of multiplying two n-bit numbers is known to be $O(n^{1+\delta})$ for an arbitrarily small δ. Parallel and optical processors designed for multiplying large precision numbers can be applied to matrix multiplication using (12).

2.5. Implementation Issues

Since the order of the cyclic convolution will almost never be prime or a power of two, pseudo-number-theoretic transforms must be used. Tables of factors of $2^n\pm 1$ for various n, and suggested moduli that are primes or products of large primes, are available. There is considerable flexibility, depending on the desired order of the cyclic convolution, and the hardware for implementing the modulo reductions. In particular, reduction $mod(2^n\pm 1)/q$, where q is a small prime, can be implemented by first reducing $mod(2^n\pm 1)$ (done easily by bit shifting), and only then reducing the remainder $mod(2^n\pm 1)/q$.

The scale factor s should be chosen to be a power of two, so that the scalings of the multiplicands can be performed by bit shifting, and the separation can be performed by segmenting binary representations of numbers, as discussed above. If an n-bit cyclic convolution is used, the scaling factor s may be chosen (crudely) as the smallest power of two exceeding $N\,max[a_{i,j},b_{i,j}]^{2/n}$ since each $c_{i,j}$ or $*$ is the

sum of N products in $N \times N$ matrix multiplication and s^n is the separation factor (see (10) and (11)).

Note that the result y_k of the pseudo-number-theoretic transform is not $c_{i,j}+s^n c_{i',j'}$, but $s^k c_{i,j}+s^{k+n}c_{i',j'} \quad mod(p)$, for some $c_{i,j}$ and $c_{i',j'}$ or perhaps $*$ (see (10) and (11)). Due to reduction $mod(p)$, $c_{i,j}$ and $c_{i',j'}$ cannot be separated by binary segmentation. To obtain $c_{i,j}+s^n c_{i',j'}$, it is necessary to solve the equation

$$s^k(c_{i,j}+s^n c_{i',j'}) = y_k \quad mod(p) \quad \rightarrow$$

$$c_{i,j}+s^n c_{i',j'} = s^{m-k} y_k \quad mod(p), \qquad (13)$$

which fortunately can be done by simple bit-shifting if s is a power of two. In (13) m is the number such that $p|(s^m-1)$, i.e., $s = 2^{n/m}$.

In order to minimize the number of multiplications without incurring excessive bitlengths, the order of the cyclic convolution should be a factor of the length N^3+N^2-N of the linear convolution (consider what happened for $N = 2$ in going from a 5-point to a 4-point cyclic convolution). This suggests use of the order $n = N^2+N-1$, since the increased bitlength is by the known factor N. Modulus p must be chosen large enough to represent $c_{i,j}+s^n c_{i',j'}$, so p must be a prime or product of primes where: (1) $p > s^{2n}$; and (2) $p|(2^{nk}-1)$. Note that the multiplicands and products of pseudo-number-theoretic transforms are bounded by p, so none of these will be very large (unless (12) is used).

3. NUMERICAL EXAMPLES

We present some simple numerical examples, which are designed to be illustrative instead of demonstrating the effectiveness of the new algorithms. Larger values of N lead to more efficient algorithms. It should be remembered that these algorithms are non-commutative and can be nested.

3.1. Example #1

We implement the 2×2 matrix multiplication

$$\begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ 7 & 6 \end{bmatrix} = \begin{bmatrix} 46 & 40 \\ 62 & 54 \end{bmatrix}. \qquad (14)$$

using six multiplications. We use 64 as a bound for $c_{i,j}$ and $*$, since this allows the use of $s = 2$. The prime modulus p must have the properties: (1) $p > 64^2 = 4096$; and (2) $p|2^{6k}-1$ for some k. We choose $p = 5419 = \frac{2^{21}+1}{(9)(43)}$; clearly 2 is a 42^{nd} root of unity $mod(p)$, so $2^7 = 128$ is a 6^{th} root of unity $mod(p)$.

The polynomial (7) for this problem is

$$(2+3x+4x^2+5x^3)(7+9x^2+6x^4+8x^6)$$

$$= 14+21x+46x^2+62x^3+48x^4$$

$$+63x^5+40x^6+54x^7+32x^8+40x^9, \qquad (15)$$

Scaling by $s = 2$ and taking the result $mod(z^6-1)$ and $mod(5419)$, the polynomial (10) for this problem is

$$(2+6z+16z^2+40z^3)(519+36z^2+96z^4)$$

$$= 2574+1535z+2957z^2+4719z^3$$

$$+768z^4+2016z^5; mod(z^6-1); mod(5419). \qquad (16)$$

The products of the 6-point pseudo-number-theoretic transforms are:

$$\bullet k = 0 : [2+6+16+40][519+36+96] \equiv 3731.$$

$$\bullet k = 1 : [2+6(128)+16(128)^2-40]\times$$
$$[519+36(128)^2-96(128)] \equiv 4197.$$

$$\bullet k = 2 : [2+6(128)^2-16(128)+40]\times$$
$$[519-36(128)+96(128)^2] \equiv 4627.$$

$$\bullet k = 3 : [2-6+16-40][519+36+96]] \equiv 3448.$$

$$\bullet k = 4 : [2-6(128)+16(128)^2+40]\times$$
$$[519+36(128)^2-96(128)] \equiv 2683.$$

$$\bullet k = 5 : [2-6(128)^2-16(128)-40]\times$$
$$[519-36(128)+96(128)^2] \equiv 2177.$$

Here all computations are $mod(5419)$ (recall that multiplications by powers of 128 are just bit shifts). Note that only half of the NTTs of the elements of B (the second polynomial) need to be computed, since the second polynomial is a function of z^2. Computation of the inverse NTTs of (17) results in the coefficients of the product polynomial in (16).

Separation of the elements $c_{i,j}$ of the product matrix:

$$14+2^6(40) \equiv 2574; \quad 2^1(21)+2^7(54) \equiv 1535; mod(5419)$$

$$2^2(46)+2^8(32) \equiv 2957; \quad 2^3(62)+2^9(40) \equiv 4719; mod(5419)$$

Note that (13) may be used to recover the $c_{i,j}$, although for such small values simply adding 5419 a few times to the y_k to create multiples of 2, 4, and 8 is easier.

3.2. Example #2

We implement the 2×2 matrix multiplication (14) using (12). We choose $s = 100$ to make the operation of (12) clear, and use commas to identify the $c_{i,j}$. The result is

$$(5040302)(8000600090007) = 4032, 54, 40, 6348, 62, 46, 2114.$$

Using $s = 64$ instead of $s = 100$ would produce a similar result in binary notation, and would produce a result smaller than that of (19) by a factor of $(100/64)^{10} = 86.7$.

4. REFERENCES

[1] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, Vol. 13, 1969, pp. 354-356.

[2] V. Pan, "How can we speed up matrix multiplication?" *SIAM Review*, Vol. 26, No. 3, July 1984, pp. 393-415.

[3] D. Bini, M. Capovani, G. Lotti, and F. Romani, "$O(n^{2.7799})$ complexity for matrix multiplication," *Inform. Proc. Lett.*, Vol. 8, 1979, pp. 234-235.

[4] A.E. Yagle, "Fast Algorithms for Matrix Multiplication Using Pseudo-Number-Theoretic Transforms," to appear in *IEEE Trans. Sig. Proc.*, Jan. 1995.