Department of Applied Analysis
and Computer Science
Technical Report CS-73-02
January, 1973
ON THE COMPLEXITY OF
SYMMETRIC COMPUTATIONS
by
Robert L. Probert
University of Waterloo

Faculty of Mathematics

University of Waterloo

Waterloo, Ontario

Canada

Department of Applied Analysis
and Computer Science
Technical Report CS-73-02
January, 1973

ON THE COMPLEXITY OF SYMMETRIC COMPUTATIONS

by
Robert L. Probert
University of Waterloo

ABSTRACT

The paper demonstrates the equivalence of bilinear chains and matrix multiplication algorithms which do not assume commutativity of multiplication. This equivalence is exploited to obtain the main symmetry result, that the same lower bound on the number of multiplications required to compute each of the six matrix products of the form (mxn)(nxp), (nxm)(mxp), (pxm)(mxn), (mxp)(pxn), (nxp)(pxm) holds. An example is given to illustrate the technique of constructing algorithms for computing (nxm)(mxp) and (pxm)(mxn) matrix products from an algorithm for computing a product of the form (mxn)(nxp). Finally, the main theorem is used to obtain a new lower bound on the complexity of general matrix multiplication.

Introduction

Very few lower bounds are known on the complexity of general matrix multiplication other than those resulting from independence arguments such as found in Winograd (1969), Hopcroft and Kerr (1969), and Fiduccia (1970). Moreover, these lower bounds are more appropriate to inner and matrix-vector products than to general matrix by matrix products. Hopcroft and Kerr (1969) have shown that seven multiplications are required to compute the product of two 2 x 2 matrices if commutativity is not assumed. Winograd (1971) has proven that Strassen's (1969) 7-multiplication algorithm for (2x2)(2x2) is optimal even if commutativity is allowed. Finally, by ingenious though tedious combinatorial arguments, Hopcroft and Kerr (1969) proved that $\lceil 7n/2 \rceil$ multiplications are necessary and sufficient to compute products of the form (2x2)(2xn), $n \geq 1$ naturally, and that 15 multiplications suffice and are required to compute a product of the form (3x2)(2x3). Together with Kirkpatrick's (1972) proof that (mxn)(nxp) matrix multiplication requires m(n+p-1) multiplications, this comprises the extent of our knowledge of the essential complexity of the class of matrix multiplication algorithms.

Basically, there are two possible approaches for proving lower bounds depending on the definition of the class of algorithms under discussion. Typical of the first approach is that of Winograd (1969) in which an algorithm α is defined as follows:

Let the number of steps in α be denoted by N and label the steps 1,2,..., N. Let Q represent the field of rational numbers.

Denote by $e_\alpha(j)$ the <u>evaluation</u> of α at step j, ie. the expression computed by α in its jth step.

α is restricted as follows:

Either $e_\alpha(j) \in Q \cup \{a_{11},\ldots,a_{mn},b_{11},\ldots,b_{np}\}$ where the a_{ij}'s and b_{ij}'s are indeterminates,
or $e_\alpha(j) = e_\alpha(j_1)$ 'op' $e_\alpha(j_2)$ where $j_1, j_2 < j$ and 'op' is either '+', '-', or 'x'; division is not allowed.

α is said to compute the product Y of $A_{mxn}B_{nxp}$ if $\exists$ $j_1, j_2,\ldots, j_{mp}$ such that $e_\alpha(j_k) = y_{rs}$ where $k=(r-1)p+s$.

In other words, each evaluation consists of a rational number, an indeterminate, or the sum, difference, or product of earlier evaluations. The algorithm computes the right answer if each element in the product matrix appears at some step as an evaluation. Lower bounds are obtained by using independence arguments. Finally, this definition of a computation permits intermediate evaluations of the form $P_1(a_{ij},b_{ij}) \cdot P_2(a_{ij},b_{ij})$ where P_1,P_2 are polynomials of arbitrary order in the indeterminates with rational coefficients. Obviously, such evaluations are designed to exploit the commutativity of multiplications of matrix elements, ie. $a_{ij}b_{kl} = b_{kl}a_{ij}$ for all $1 \le i \le m$, $1 \le j, k \le n$, $1 \le l \le p$. Conceptually, we may think of the indeterminates as taking on irrational values; thus, since multiplication occurs over the real field, it is a commutative operation.

However, it is an interesting fact that Strassen's (1969) algorithm for multiplying two nxn matrices does not assume commutativity. Moreover, it seems natural that the additional restriction that non-commutativity imposes on the class of algorithms under discussion should allow us to discover more of the nature of the matrix multiplication problem. In order to be more precise, we define the class of algorithms, NC, by:

An algorithm, α, is in NC if and only if any multiplication p_i in α is of the form u.u' where $u \in Q[a_{11},\ldots,a_{mn}]$ and $u' \in Q[b_{11},\ldots,b_{np}]$, i.e. u,u' are polynomials in the a_{ij}'s and b_{ij}'s respectively.

Unless otherwise stated, we will use 'the complexity of a computation' to refer to the minimum number of multiplications required to perform the computation by an algorithm in NC.

STRUCTURE OF NC

In fact, we can be even more precise about the form of multiplications p_i. The following two lemmas parallel Winograd's (1971a).

Lemma 1: If α is a given algorithm to compute a (mxn)(nxp) matrix product and $p_1,\ldots,p_k$ are the results of the multiplications in α, then the partial result computed at the jth step, $e_\alpha(j)$, is of the following form:

$$r + \sum_{i=1}^{k} r_i p_i + \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij} a_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{p} r'_{ij} b_{ij}$$

where the r_i's, r_{ij}'s, and r'_{ij}'s are all in Q.

Proof: Obviously, $e_\alpha(1) \in Q \cup \{a_{11},\ldots,b_{np}\}$ and therefore is of the required form.
Suppose for all steps $j \neq 1$, $e_\alpha(j)$ is of the required form. If $e_\alpha(1)$ is a multiplication, then $e_\alpha(1)=p_k$ for some k. Otherwise, $e_\alpha(1)= e_\alpha(j_1) \pm e_\alpha(j_2)$ for $j_1, j_2 < 1$, i.e.

$$e_\alpha(1) = (r^{(1)} \pm r^{(2)}) + \sum_{i=1}^{k}(r_i^{(1)} \pm r_i^{(2)})p_i + \sum_{i=1}^{m}\sum_{j=1}^{n}(r_{ij}^{(1)} \pm r_{ij}^{(2)})a_{ij}$$
$$+ \sum_{i=1}^{n}\sum_{j=1}^{p}(r_{ij}^{(1)'} \pm r_{ij}^{(2)'})b_{ij}$$

which is of the required form.

Lemma 2: If α is an algorithm which computes the matrix product (mxn)(nxp), has k multiplication steps, and $\alpha \in NC$, then there exists an algorithm α' with no more than k multiplication steps, such that each multiplication is of the form:

$$\left(\sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}a_{ij}\right)\left(\sum_{i=1}^{n}\sum_{j=1}^{p} r'_{ij}b_{ij}\right)$$

and such that α' computes the same matrix product.
(Note: the number of multiplications in an algorithm does not count multiplication by a rational number, e.g. $r_{ij}a_{ij}$ is not counted as a multiplication if r_{ij} is in Q.)

Proof: Let the functions L_i, i=0,1,2,3, of polynomials in the a_{ij}'s and b_{ij}'s be defined as follows:

$L_i : Q[a_{11},\ldots,a_{mn},b_{11},\ldots,b_{np}]$ onto $Q[a_{11},\ldots,b_{np}]$ such that if $u \in Q[a_{11},\ldots,b_{np}]$, then

$L_0(u)$=the constant term of u,

$L_1(u)$=the linear term of u, i.e. $\sum_{i=1}^{m}\sum_{j=1}^{n} v_{ij}a_{ij}+\sum_{i=1}^{n}\sum_{j=1}^{p} w_{ij}b_{ij}$,

$L_2(u)$=the quadratic term of u, and $L_3(u)$, what remains, i.e.

$L_3(u)= u - L_0(u) - L_1(u) - L_2(u)$.

Suppose $e_\alpha(j)= e_\alpha(j_1)e_\alpha(j_2) = u.u'$.

$e_\alpha(j)=L_0(u)L_0(u') + L_0(u)(u'-L_0(u')) + (u-L_0(u))\,L_0(u')$

$+ (u-L_0(u))(u'-L_0(u'))$. The only multiplication which is counted is the final one. Thus, we can construct a new algorithm α_0 such that α_0 has the same number of multiplications as α, computes the same product, and if $e_{\alpha_0}(j)=u.u'$, then $L_0(u)=L_0(u')=0$. Therefore, if $e_{\alpha_0}(j)=u.u'$ then $L_2(u.u')= L_1(u).L_1(u')$.

By Lemma 1, if $e_{\alpha_0}(j_0)=\sum_{l=1}^{n} a_{il}b_{lj}$, 1

then, $e_{\alpha}(j_0) = r_0 + \sum_{i=1}^{k} r_i p_i + \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij} a_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{p} r'_{ij} b_{ij}$ 2

Applying L_2 to 1 and 2, we have $\sum_{l=1}^{n} a_{il} b_{lj} = \sum_{i=1}^{k} r_i L_2(p_i)$.

Then, set $p_i = u_i . u'_i$. Thus,

$\sum_{l=1}^{n} a_{il} b_{lj} = \sum_{i=1}^{k} r_i L_2(u_i . u'_i) = \sum_{i=1}^{k} r_i L_1(u_i) . L_1(u'_i)$. Note that $r = (r_1, \ldots, r_k)$ is a function of i and j.

Hence, to compute y_{ij} for $1 \leq i \leq m$, $1 \leq j \leq p$, an algorithm α' suffices which computes the k multiplications $L_1(u_i) . L_1(u'_i)$. Note that no multiplications are required to compute $L_1(u_i)$, and since $\alpha \in NC$ and α contains $u_i . u'_i$, $L_1(u_i) = \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij} a_{ij}$ and $L_1(u'_i) = \sum_{i=1}^{n}\sum_{j=1}^{p} r'_{ij} b_{ij}$. Thus, the algorithm α' has only multiplications of the required form and computes the same matrix product as α.

The above lemma is a formal proof of the intuitively obvious observation made in (Hopcroft, 1969), namely that since constant terms and terms higher than quadratic do not appear in the product to be computed, these terms need not appear in intermediate multiplications. Thus, an equivalent algorithm exists with the same number of multiplications whose multiplication steps are the product of a linear sum of a_{ij}'s with a linear sum of b_{ij}'s.

The following algorithms both compute the matrix product of two (3x3) matrices "faster" than the usual brute-force method which uses $3^3 = 27$ multiplications. The first method belongs to NC, the second, a typical inner-product algorithm, presumes commutativity of multiplication of matrix elements and hence is not in NC.

Method 1: To compute $A_{3x3} B_{3x3}$, proceed as follows.

Let $A_1 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$, $B_1 = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix}$, $A_2 = (a_{13}\ a_{23}\ a_{33})^T$

$B_2 = (b_{31}\ b_{32}\ b_{33})$.

Then, use the algorithm given in Hopcroft and Kerr to compute A_1B_1 in 15 multiplications (which is optimal). Compute A_2B_2 in 9 multiplications (also optimal) and set $AB = A_1B_1 + A_2B_2$ at a total cost of 24 multiplications.

Method 2: The well-known "fast" inner product algorithm (see Winograd, 1968, 1969) uses only three multiplications fewer than brute force. For example, if $C_{3x3}=AB$, compute c_{ij}, $1 \leq i,j \leq 3$, as $(a_{i1}+b_{2j})(a_{i2}+b_{1j}) - a_{i1}a_{i2} - b_{1j}b_{2j} + a_{i3}b_{3j}$.

Each c_{ij} requires 2 unique multiplications, namely $(a_{i1}+b_{2j})(a_{i2}+b_{1j})$ and $a_{i3}b_{3j}$. There are three multiplications each of the form $a_{i1}a_{i2}$, and $b_{1j}b_{2j}$. Therefore, the total number of multiplications in Method 2 is (9 x 2) + 3 + 3 = 24 multiplications. Thus, exploiting commutativity in this way affords us no saving over the non-commutative Method 1.

In fact, no algorithm for multiplying 3 x 3 matrices has been found, including algorithms which exploit commutativity, which employs fewer than 24 multiplication steps!

BILINEAR CHAINS

This section defines Fiduccia's (1972) notion of a bilinear chain and proves that this class of algorithms is precisely NC.

We will often refer to the matrix-vector product which corresponds to a given matrix multiplication problem. In order to make precise this correspondence, we define:

the <u>tensor product</u> $A_{mxn} \otimes B_{rxs}$ as $C_{\underline{mr}x\underline{ns}}$ where $c_{i+k,j+p} = a_{ij}b_{k+1,p+1}$,

and the <u>direct sum</u> $A_{mxn} \oplus B_{rxs}$ as $C_{(m+r)x(n+s)} = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$;

Then, if I_s stands for the identity matrix with s 1's, we have $I_s \otimes A = A \oplus \ldots \oplus A$, s times. Finally, if B_{rxs} has the s columns $b_1, \ldots, b_s$, define $\chi(B)$ as

$$\chi(B) = \left[b_1^T, \ldots, b_s^T\right]^T, \text{ i.e. an rs-column vector.}$$

<u>Lemma 3</u>: (Fiduccia) If B has s columns, let $d = \chi(B)$, $C = I_s \otimes A$. Then, the set of entries of AB is identical to the set of entries of Cd.

Essentially, this lemma has reduced the matrix multiplication problem to a matrix-vector multiplication problem. Since this idea of entry equivalence is central, denote the set of elements of the matrix result AB by E(AB). Define $AB \overset{E}{=} Cd$ iff $E(AB) = E(Cd)$.

Let R be a ring and let K be a subring of the center of R such that $ar = ra$ for all (a,r) in $K \times R$. Let $X = (x_{ij})$ be a matrix variable which ranges over a

non-empty subset S of R^{mxn} and $y = (y_1,\ldots,y_n)^T$ be a vector variable over $R^{nx1} = R^n$.
α is an <u>algorithm</u> for the matrix-vector product Xy if α computes E(Xy) from E(X) $\cup$ E(y) for any pair (X,y) in SXR^n . Then define $L_K(E(X))$ as the set of all linear combinations $\sum_{i=1}^{s} w_i x_i$ of E(X) (Here, s=mn, X is (mxn)) with fixed w_i's in K, which, for our purposes will be the set $\{0,1,-1\}$.) .

A <u>K-chain</u> ϕ for E(Xy) is a finite sequqnce $\phi_1,\ldots$ such that for each $z \in E(Xy)$ there is a p such that $\phi_p = z$ where each ϕ_k is either in E(X) $\cup$ E(y) or

$$\phi_k = r\phi_i \text{ where } r \in K, \text{ or}$$

$$\phi_k = \phi_i \text{'op'} \phi_j \text{ for } i,j<k \text{ and 'op'} \in \{'+','-','x'\}.$$

The chain ϕ is <u>K-bilinear</u> iff whenever $\phi_k = \phi_i x \phi_j$, $\phi_i \in L_K(E(X))$ and $\phi_j \in L_K(E(y))$.

By comparing the definitions of NC and bilinearity, and applying Lemmas 2 and 3, we have the following straightforward equivalence between the two models:

<u>Lemma 4</u>: For any algorithm $\alpha \in$ NC which computes $A_{mxn}B_{nxp}$ using only k multiplication steps, there exists a bilinear chain which computes E(Xy) where $X = I_p \otimes A$ and $y = (b_1^T,\ldots,b_p^T)^T$ using no more than k multiplications, and conversely.

Hence, using Lemma 4 as a bridge between matrix by matrix multiplication algorithms (in NC) and matrix-vector

bilinear chains, we can now employ a theorem by Fiduccia:

Theorem 5: There is a K-bilinear chain for $E(Xy)$ with t multiplication steps iff there is a K-bilinear chain for $E(X^T z)$, where z ranges over R^m, with t multiplication steps.

This result is illustrated in great detail in a later section; for now, we need only the statement of the theorem to prove a result about symmetry in NC.

Lemma 6: Matrix products of the form $A_{mxn}B_{nxp}$ require t multiplications by algorithms in NC iff products of the form $C_{nxm}D_{mxp}$ require t multiplications to be computed by an algorithm in NC.

Proof: By Lemma 4, there is an algorithm $\alpha \in$ NC which computes $A_{mxn}B_{nxp}$ in t multiplications iff there is a bilinear chain using only t multiplications which computes $E(Xy)$, where Xy is the corresponding matrix-vector product.

By Theorem 5, this chain exists iff there is a bilinear chain for $E(X^T z)$ with t multiplication steps where z ranges over R^m . Note that X^T is of the form $I_p \otimes C_{nxm}$ and z is of the form $(d_1^T,\ldots,d_p^T)^T$.

Again by Lemma 4, this chain exists iff there is an algorithm α' NC which computes $E(C_{nxm}D_{mxp})$ where C,D are matrix variables (as are A,B,X) with exactly t multiplication steps.

Hence, matrix products of the form (mxn)(nxp) require the same number of multiplications as products of the form (nxm)(mxp) .

Lemma 7: For any algorithm α with t multiplication steps which computes matrix products of the form $A_{mxn}B_{nxp}$, there exists an algorithm α' which computes matrix products of the form $C_{pxn}D_{nxm}$ using the same number of multiplication steps.

Proof: Note that simply computing $C_{pxn}D_{nxm}$ as $(D^TC^T)^T$ via algorithm α will not work since we are not allowed to assume element multiplication is commutative; element products $d_{ji}c_{lk}$ do not necessarily equal $c_{lk}d_{ji}$. What we can do, however, is to collect the multiplications m_i of algorithm α where α is applied to a computation of the form $A_{mxn}B_{nxp}$. The matrices A and B are D^T and C^T respectively. For each m_i in α, construct the "reverse" m_i^R multiplication as follows:
If $m_i = (\sum_{i=1}^{m}\sum_{j=1}^{n} v_{ij}a_{ij})(\sum_{i=1}^{n}\sum_{j=1}^{p} w_{ij}b_{ij})$, then set
$m_i^R = (\sum_{i=1}^{n}\sum_{j=1}^{p} w_{ij}b_{ij})(\sum_{i=1}^{m}\sum_{j=1}^{n} v_{ij}a_{ij})$. Now, if we combine these multiplications exactly as α does, we get an element of the product matrix $C_{pxn}D_{nxm}$.

More formally, an element z_{ij} in CD is found as follows:
If α computes z_{ji} as $\sum_{i=1}^{t} v_i m_i$ where $v_i \in K = \{0, 1, -1\}$, then let α' compute z_{ij} (in CD) as $\sum_{i=1}^{t} v_i m_i^R$. The example in the next section will illustrate this procedure.

Now, the main result on the symmetry of the general matrix multiplication problem follows immediately by alternating applications of Lemmas 6 and 7.

Theorem 8 (SYMMETRY THEOREM): Matrix products of each of the following forms have the same computational complexity, i.e. require the same number of multiplications to compute:

(nxm)(mxp), (pxm)(mxn), (mxp)(pxn),
(nxp)(pxm), (pxn)(nxm), (mxn)(nxp) .

Obviously, Theorem 8 is a very important tool for extending known results, for example we can extend all the results in Hopcroft and Kerr's (1969) paper as follows:

Corollary 8a: Algorithms which compute matrix products of the following forms must contain at least 15 multiplication steps:

(3x2)(2x3), (2x3)(3x3), (3x3)(3x2) .

Corollary 8b: Algorithms which compute matrix products of the form (2x2)(2xn), (nx2)(2x2), or (2xn)(nx2) require $\lceil 7n/2 \rceil$ multiplications.

We can also use Theorem 8 to show that these bounds are best bounds; that fact follows immediately from Corollary 8c in the next section on finding algorithms for problems symmetric to solved problems.

TRANSFORMING ALGORITHMS

This section presents a detailed example to illustrate how to obtain new "fast" algorithms from old. We can proceed in a well-defined manner from an algorithm α for (mxn)(nxp) to one for (nxm)(mxp) and then to one for (pxm)(mxn). If we examine the proofs of Theorems 5 and 8, we obtain the following:

Corollary 8c: Given any algorithm to compute a product of the form (nxm)(mxp) which uses t multiplications, we can construct ones using exactly t multiplication steps and computing products of the forms (pxn)(nxm), (nxp)(pxm), (mxp)(pxn), (pxm)(mxn), (nxm)(mxp).

We can also note that transformations from an algorithm for products of the form (mxn)(nxp) to one for products of the form (pxn)(nxm) by the technique of Lemma 7 preserve additions and subtractions as well as multiplications. Transformations via Lemma 6 from algorithms for (mxn)(nxp) to those for (nxm)(mxp), however, may drastically change the number of additions and subtractions.

To illustrate these techniques, we will compute the algorithms for products of the forms (nxm)(mxp) and (pxm)(mxn) from an algorithm for products of the form (mxn)(nxp) where m=4, n=2, and p=4 (here, m happens to be the same as p; the techniques are, however, independent of the values of m,n,p) .

The first step is to construct a "fast" algorithm which computes matrix procucts of the form $A_{4x2}B_{2x4}$. Following Hopcroft and Kerr's (1969) technique, we arrive at the following set of 26 multiplications. The letters

A_i, B_j, C_k, etc. refer to the structure of the multiplication in relation to the seven multiplications in some variant of Strassen's algorithm. In their paper, Hopcroft and Kerr defined a group of product-preserving transformations of algorithms for matrix multiplication. Thus, the seven letters designate the seven equivalence classes of multiplications in Strassen's scheme.

The multiplications which suffice to compute matrix products of the form (4x2)(2x4) are listed below.

A_1 $a_{12}(b_{11}+b_{21})$

B_1 $(a_{11}-a_{12})b_{11}$

C_1 $(a_{21}-a_{22})b_{22}$

D_1 $a_{21}(b_{12}+b_{22})$

E_1 $a_{32}b_{23}$

F_1 $a_{31}b_{13}$

G_1 $(a_{21}+a_{12})(b_{11}-b_{22})$

A_2 $a_{42}(b_{14}+b_{24})$

B_2 $(a_{41}-a_{42})b_{14}$

C_2 $(a_{31}+a_{12}-a_{11}-a_{32})(b_{23}-b_{21})$

D_2 $(a_{31}-a_{11})(b_{13}+b_{23}-b_{11}-b_{21})$

E_2 $(a_{12}+a_{22})(b_{21}+b_{22})$

F_2 $(a_{11}+a_{21})(b_{11}+b_{12})$

G_2 $(a_{12}+a_{31}-a_{11})(b_{11}+b_{21}-b_{23})$

A_3 $(a_{32}-a_{22})(b_{13}+b_{23}-b_{12}-b_{22})$

B_3 $(a_{31}+a_{22}-a_{32}-a_{21})(b_{13}-b_{12})$

C_3 $(a_{31}+a_{42}-a_{41}-a_{32})(b_{23}-b_{24})$

D_3 $(a_{31}-a_{41})(b_{13}+b_{23}-b_{14}-b_{24})$

E_3 $(a_{42}+a_{22})(b_{24}+b_{22})$

F_3 $(a_{41}+a_{21})(b_{14}+b_{12})$

G_3 $(a_{21}+a_{32}-a_{22})(b_{13}-b_{12}-b_{22})$

C_4 $(a_{11}+a_{21}+a_{42}-a_{41}-a_{12}-a_{22})(b_{21}+b_{22}-b_{24})$

D_4 $(a_{11}+a_{21}-a_{41})(b_{11}+b_{12}+b_{21}+b_{22}-b_{14}-b_{24})$

G_4 $(a_{21}+a_{42})(b_{14}-b_{22})$

G_5 $(a_{31}-a_{41}+a_{42})(b_{14}+b_{24}-b_{23})$

G_6 $(a_{11}+a_{21}-a_{41}+a_{42})(b_{14}-b_{21}-b_{22}+b_{24})$

MULTIPLICATIONS IN α

Each diagonal element is computed from two multiplications; symmetric off-diagonal elements are computed in pairs from three additional multiplications. For example, if we let C_{4x4} denote the product matrix $A_{4x2}B_{2x4}$, then the elements c_{11}, c_{22}, c_{12}, and c_{21} of C are computed from the multiplications A_1, B_1, C_1, D_1, E_2, F_2, and G_1 as follows:

$$c_{11} = a_{11}b_{11}+a_{12}b_{21} = A_1 + B_1$$

$$c_{22} = a_{21}b_{12}+a_{22}b_{22} = -C_1 + D_1$$

$$c_{12} = -B_1 - D_1 + F_2 - G_1$$

$$= -a_{11}b_{11}+a_{12}b_{11}-a_{21}b_{12}-a_{21}b_{22}+a_{11}b_{11}+a_{11}b_{12}$$
$$+a_{21}b_{11}+a_{21}b_{12}-a_{21}b_{11}+a_{21}b_{22}-a_{12}b_{11}+a_{12}b_{22}$$

$$= a_{11}b_{12} + a_{12}b_{22}$$

$$c_{21} = -A_1 + C_1 + E_2 + G_1$$

$$= a_{21}b_{11} + a_{22}b_{21}$$

The algorithm uses 8 multiplications to compute the four diagonal elements, as well as 18 multiplications to compute the six pairs of off-diagonal elements. The total number of multiplications used is therefore 8+18= 26. Call this 26-multiplication algorithm for computing products of the form (4x2)(2x4) α.

The next step is to transform α into α' via Lemma 6 such that α' computes products of the symmetric form (2x4)(4x4) using exactly 26 multiplications. To do this, we first must characterize α in terms of the following decomposition theorem:

Theorem 9: (Fiduccia) There exists a K-bilinear chain ϕ for E(Xy) with t multiplication steps iff there exist fixed matrices F,G with elements in K and a (txt)

diagonal matrix U with elements in $L_K(E(X))$ such that

$$X = GUF \ .$$

Actually, Fiduccia's result is slightly more general, characterizing X as GUF + H . Our class of algorithms allows operations on zero matrices as well as on non-degenerate ones, however, and so H=0, yielding Theorem 9 .

To make use of this theorem we first must find the matrix-vector product which corresponds to a matrix by matrix product of the form (4x2)(2x4) . By Lemma 3, this is Xy where X and y are defined as follows.

Let $X = I_4 \otimes A_{4x2} = \begin{pmatrix} A & . & . & . \\ . & A & . & . \\ . & . & A & . \\ . & . & . & A \end{pmatrix}$, where '.' stands for 0,

and let $y = \chi(B) = (b_{11}, b_{21}, \ldots, b_{14}, b_{24})^T$.

Then, Xy has the product form (16x8)(8x1) .

Now, we construct the (26x26) matrix U by placing along its diagonal the contents of the left bracket of each multiplication of α in order as it appears in the previous list, and zeros everywhere else. In other words,

$U = \begin{pmatrix} A'_1 & & & 0 \\ & B'_1 & & \\ & & \ddots & \\ 0 & & & G'_6 \end{pmatrix}$, where M'_i is obtained from M_i by deleting the entries from $L_K(E(y))$. For example,

$A_1 = a_{12}(b_{11}+b_{21})$. Hence, $A'_1 = a_{12}$.

The matrix G will specify which M'_i 's will be used to calculate an element of Xy and exactly how these intermediate results will be composed. The matrix F will operate

on U and y to provide the elements from $L_K(E(y))$ which make up the righthand component of each original M_i . For $X_{16x8}y_{8x1}$, the matrix F_{26x8} will be:

$$F = \begin{pmatrix}
1 & 1 & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & . \\
. & . & 1 & 1 & . & . & . & . \\
. & . & . & . & 1 & . & . & . \\
. & . & . & . & 1 & . & . & . \\
. & 1 & . & . & -1 & . & . & . \\
. & . & . & . & . & . & 1 & 1 \\
. & . & . & . & . & . & 1 & . \\
. & -1 & . & . & . & 1 & . & . \\
-1 & -1 & . & . & 1 & 1 & . & . \\
. & 1 & . & 1 & . & . & . & . \\
1 & . & 1 & . & . & . & . & . \\
1 & 1 & . & . & . & -1 & . & . \\
. & . & -1 & -1 & 1 & 1 & . & . \\
. & . & -1 & . & 1 & . & . & . \\
. & . & . & . & . & 1 & . & -1 \\
. & . & . & . & 1 & 1 & -1 & -1 \\
. & . & . & 1 & . & . & . & 1 \\
. & . & 1 & . & . & . & 1 & . \\
. & . & -1 & -1 & 1 & . & . & . \\
. & 1 & . & 1 & . & . & . & -1 \\
1 & 1 & 1 & 1 & . & . & -1 & -1 \\
. & . & . & -1 & . & . & 1 & . \\
. & . & . & . & . & -1 & 1 & 1 \\
. & -1 & . & -1 & . & . & 1 & 1
\end{pmatrix}
\begin{matrix}
A_1 \\ \\ C_1 \\ \\ E_1 \\ \\ G_1 \\ \\ B_2 \\ \\ D_2 \\ \\ F_2 \\ \\ A_3 \\ \\ C_3 \\ \\ E_3 \\ \\ G_3 \\ \\ D_4 \\ \\ G_5
\end{matrix}
\qquad
\begin{matrix}
B_1 \\ \\ D_1 \\ \\ F_1 \\ \\ A_2 \\ \\ C_2 \\ \\ E_2 \\ \\ G_2 \\ \\ B_3 \\ \\ D_3 \\ \\ F_3 \\ \\ C_4 \\ \\ G_4 \\ \\ G_6
\end{matrix}$$

where the letters on the right indicate which multiplication of α has its left component equal to that row multiplied by y. For example, multiplying the first row of F by the column vector y yields

$$(1,1,0,\ldots,0)(b_{11},b_{21},\ldots,b_{24})^T$$
$$= b_{11}+b_{21} \text{ , the left side of multiplication } A_1 \text{ , as expected.}$$

In fact, the product F_y is a 26-element column vector which contains the righthand components of each multiplication of α in order. Thus, the product UFy yields all the multiplications of α in the order in which they are listed on page 13 .

The (16x26) matrix G is given below. The $4(j-1)+i$ th row of G encodes the combinations of the multiplications in α which are used to compute z_{ij} in the (16x1) vector $Xy = (z_{11}, z_{21}, z_{31}, \ldots, z_{44})$.

$$G = \begin{pmatrix}
1 & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
-1 & . & 1 & . & . & . & 1 & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & 1 & . & . & 1 & . & . & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & 1 & 1 & . & . & . & 1 & . & . & . & . & . & -1 & . & . & -1 & 1 & . & -1 \\
. & -1 & . & -1 & . & . & -1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & -1 & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & -1 & . & . & 1 & . & . & . & . & . & . & . & . & -1 & . & . & . & -1 & . & . & . & . & . & . & . \\
. & . & . & -1 & . & . & . & -1 & . & . & . & . & . & . & . & . & . & 1 & . & . & . & -1 & . & . & . & . \\
1 & . & . & . & . & 1 & . & . & . & -1 & . & -1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
* & . & . & . & 1 & 1 & . & . & . & . & . & . & . & . & -1 & . & . & . & . & 1 & . & . & . & . & . & . \\
. & . & . & . & . & 1 & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & 1 & . & 1 & . & . & . & . & . & . & -1 & . & . & . & . & . & . & -1 & . & . & . \\
. & . & -1 & . & . & . & . & 1 & 1 & . & . & 1 & . & . & . & . & -1 & . & . & 1 & . & -1 & . & 1 & . & . \\
. & . & 1 & . & . & . & . & -1 & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . \\
. & . & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . \\
. & . & . & . & . & . & . & 1 & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{pmatrix}$$

$$\begin{matrix}
A_1 & C_1 & E_1 & G_1 & B_2 & D_2 & F_2 & A_3 & C_3 & E_3 & G_3 & D_4 & G_5 \\
B_1 & D_1 & F_1 & A_2 & C_2 & E_2 & G_2 & B_3 & D_3 & F_3 & C_4 & G_4 & G_6
\end{matrix}$$

For example, to compute z_{23} in Xy we need to scan the $4(3-1)+2 = 10$th row of G to see which multiplications of α are required and, also, to see how these multiplications are combined. Row 10 suggests

$$z_{23} = D_1 + E_1 - A_3 + G_3$$

$$= a_{21}(b_{12}+b_{22}) + a_{32}b_{23}$$

$$- (a_{32}-a_{22})(b_{13}+b_{23}-b_{12}-b_{22}) + (a_{21}+a_{32}-a_{22})(b_{13}-b_{12}-b_{22})$$

$= a_{21}b_{13} + a_{22}b_{23}$ as required.

Multiplying the 10th row by U results in the following 26-element vector u :

$(0,0,0,D'_1,E'_1,0,\ldots,0,-A'_3,0,\ldots,0,G'_3,0,\ldots,0)$

Multiplying u by Fy yields

$D'_1(b_{12}+b_{22})+E'_1(b_{23})-A'_3(b_{13}+b_{23}-b_{12}-b_{22})+G'_3(b_{13}-b_{12}-b_{22})$

which is exactly the previous computation of z_{23} .

Thus, Xy = GUFy and this leads directly to the required decomposition of X; namely, let y range over the subset $\{0,e_1,\ldots,e_8\}$ of R^8 , then we have

$X = XI = (GUF)I = GUF$.

For example, $x_{12}=a_{12}=x_{54}=x_{96}=x_{13,8}$; hence, the decomposition should assign a_{12} to each of these four x_{ij}'s.

$x_{12}=(g_{11},g_{12},\ldots,g_{1,26})U(f_{12},f_{22},\ldots,f_{26,2})^T$

$=(1.a_{12},1.(a_{11}-a_{12}),0,\ldots,0)(1,0,\ldots,-1)^T$

$=a_{12}$

$x_{54}=(g_{51},g_{52},\ldots,g_{5,26})U(f_{14},f_{24},\ldots,f_{26,4})$

$=-a_{21}+1(a_{21}+a_{12})$

$=a_{12} = x_{12}$ as expected.

Similarly, $x_{96}=x_{13,8}=a_{12}$, and in general,

$x_{i+4k,j+2k}=x_{ij}$ for $k=0,1,2,3$.

Thus, we have decomposed our encoding X of the matrix multiplicand A into three component matrices G,U,F such that U is a diagonal matrix, and G,F contain only elements in $K=\{0,1,-1\}$.

Now, we are able to complete step two of the example, namely to derive an algorithm to compute a matrix product of the form (2x4)(4x4) . For now we have a decomposition of matrices of the form (8x16); $X^T = (GUF)^T = F^TUG^T$ because of the nature of U,F, and G . More precisely,

$I_4 \otimes A^T = X = F^TUG^T$. There is a small problem here, however, in that we do not wish a decomposition of the particular matrix A^T, rather a decomposition of all (2x4) matrices, based on α. This problem is easily solved by replacing each a_{ij} in U by a_{ji} ; therefore, call this modification of U, U' . To understand this, note that (let $f'_{ij}=f_{ji}$, $g'_{ij}=g_{ji}$)

$$(f'_{21},f'_{22},\ldots,f'_{2,26})U(g'_{11},\ldots,g'_{26,1})^T$$
$$= (f_{12},f_{22},\ldots,f_{26,2})U(1,1,0,\ldots,0)^T$$
$$= (1,0,\ldots,-1)(a_{12},(a_{11}-a_{12}),\ldots,0)^T$$
$$= a_{12}$$

where the actual element in the second row, first column of $I_4 \otimes A_{2x4}$ is a_{21} . replacing U by U', however, we have

$$(f'_{21},\ldots,f'_{2,26})U'(g'_{11},\ldots,g'_{26,1})^T$$
$$= (1,0,\ldots,-1)(a_{21},(a_{11}-a_{21}),\ldots,0)^T$$
$$= a_{21}$$

as required.

Thus, to compute a matrix product of the form $A_{2x4}B_{4x4}$, find the equivalent matrix-vector product Xy where $X=I_4 \otimes A_{2x4}$, $y = \chi(B) = (b_{11},b_{21},\ldots,b_{34},b_{44})^T$ as $Xy = F^TU'G^Ty$. Note that each element in the 8-element column vector z=Xy is a unique entry in the product matrix $C_{2x4} = A_{2x4}B_{4x4}$. Thus, to compute c_{ij}, compute

the $2(j-1)+i$ th entry in z. This, in turn, is obtained by multiplying the product of the $2(j-1)+i$ th row of F^T with U' by $G^T y$.

We now compute $c_{23}=a_{21}b_{13}+a_{22}b_{23}+a_{23}b_{33}+a_{24}b_{43}$ to illustrate the procedure. First, note that c_{23} is the $2(3-1)+2 = 6$th entry in z. Thus, to see which multiplications are involved, scan the 6th row of F^T, i.e. the 6th column of F; 1's occur in the columns of F^T which correspond to elements of U': E_1'', C_2'', D_2'', A_3'', C_3'', D_3'', -1's in those columns corresponding to G_2'', G_5'', and 0's elsewhere (M_i'' in U' is obtained from M_i' in U by replacing each a_{jk} by a_{kj}). Therefore, let the vector

$$u' = (f'_{61},f'_{62},\ldots,f'_{6,26})U'$$
$$= (0,\ldots,a_{23},\ldots,a_{13}+a_{21}-a_{11}-a_{23},a_{13}-a_{11},0,0,$$
$$-(a_{21}+a_{13}-a_{11}),a_{23}-a_{22},0,a_{13}+a_{24}-a_{14}-a_{23},a_{13}-a_{14},\ldots,$$
$$-(a_{13}-a_{14}+a_{24}),0)$$

As before, $G^T y$ yields a 26-element vector whose entries are the righthand components of the multiplications to be used by α'; thus, the set of multiplications used by α' to compute $A_{2x4}B_{4x4}$ is exactly $E(U'G^T y)$. In particular,

$$c_{23} = u'G^T y$$
$$= a_{23}(b_{31}+b_{23}+b_{33}+b_{34})+(a_{13}+a_{21}-a_{11}-a_{23})b_{13}+(a_{13}-a_{11})\cdot$$
$$(-b_{13})-(a_{21}+a_{13}-a_{11})(b_{31}-b_{13})+(a_{23}-a_{22})(-b_{23})+$$
$$(a_{13}+a_{24}-a_{14}-a_{23})b_{34}+(a_{13}-a_{14})(-b_{43})-(a_{13}-a_{14}+a_{24})(b_{34}-b_{43})$$
$$= a_{23}b_{33}+a_{21}b_{13}+a_{22}b_{23}+a_{24}b_{43} \quad \text{as required.}$$

Thus, if we let α' be the algorithm which combines the multiplications given by $E(U'G^T y)$ according to the entries in F^T, we have accomplished the second stage of

the example, i.e. we have constructed an algorithm α' which computes matrix products of the form (2x4)(4x4) using exactly the same number of multiplications as α uses to compute products of the form (4x2)(2x4). Moreover, the construction is well-defined once α is known. We should also note here that the number of additions/subtractions employed by algorithms for symmetric problems is not necessarily constant.

The third and final problem in this example, is to find α'' which computes products of the form (4x4)(4x2) from the algorithm α' just constructed for the (2x4)(4x4) case. Clearly, if we assumed all multiplications commute, we could easily apply α' to the transposed multiplication problem and transpose the result as an algorithm with no more multiplications than α' . However, this is taboo in NC. But, by the techniques of Lemma 7 we are able to construct the required α'' from α' .

Suppose we wish α'' to compute $C_{4x4}D_{4x2}$. Let $A_{2x4}D^T$, $B_{4x4}=C^T$. Since $\alpha'' \in NC$, we must not assume each intermediate product $d_{ij}c_{kl}$ in the computation of AB commutes. However, we can sum these products as if they did commute. In fact, if α'' sums the reversals of multiplications in $U'G^Ty$ exactly as α' sums the originals, the result computed by α'' will be the reversal of α' 's computation and, therefore, the desired result.

For example, let C'=AB, E=CD . Then, to compute $e_{32}=c_{31}d_{12}+c_{32}d_{22}+c_{33}d_{32}+c_{34}d_{42}$, collect those multiplications M_i which α' uses to compute c'_{23} and form the corresponding reverse multiplications as in Lemma 7 (the multiplications M_i are given on page 20.).

Summing the reverse multiplications as α' sums the M_i, we obtain

$$(b_{31}+b_{23}+b_{33})a_{23}+b_{13}(a_{13}+a_{21}-a_{11}-a_{23})+(-b_{13})(a_{13}-a_{11})$$
$$-(b_{31}-b_{13})(a_{21}+a_{13}-a_{11})+(-b_{23})(a_{23}-a_{22})+b_{34}(a_{13}+a_{24}-a_{14}-a_{23})$$
$$+(-b_{43})(a_{13}-a_{14})-(b_{34}-b_{43})(a_{13}-a_{14}+a_{24})$$
$$= b_{33}a_{23}+b_{13}a_{21}+b_{23}a_{22}+b_{43}a_{24}$$

Finally, substiting $a_{ij}=d_{ji}$, $b_{ij}=c_{ji}$, we get

$$\Sigma M_i^R = c_{33}d_{32}+c_{31}d_{12}+c_{32}d_{22}+c_{34}d_{42}$$
$$= e_{32} \text{ as required.}$$

Since all we modified in α' is the order of the multiplicands, α'' uses exactly as many multiplications, and additions/subtractions to compute products of the form (4x4)(4x2) as α' does to calculate matrix products of the form (2x4)(4x4).

Thus, we have 26-multiplication algorithms for the symmetric problems: (4x2)(2x4), (2x4)(4x4), and (4x4)(4x2) matrix multiplication. It is not known if these algorithms are optimal; obviously, by the symmetry theorem, if any of α, α', α'' is optimal, they all are.

Since we have utilized no specific features of this example to illustrate the algorithm construction process, the process itself is completely general, i.e. we have demonstrated a simple method for building algorithms for matrix products of the forms (nxm)(mxp) and (pxm)(mxn) from an algorithm for the form (mxn)(nxp) such that the new algorithms are of exactly the same computational complexity as the original. By repeating the above processes, we can obtain algorithms of equal cost for the remaining symmetric problems, namely those of the forms (mxp)(pxn), (nxp)(pxm), and (pxn)(nxm) .

A NEW LOWER BOUND

The symmetry of the complexity of matrix multiplications demonstrated by Theorem 8 can be employed to improve existing lower bounds on the number of multiplications required to compute the general matrix product (mxn)(nxp). Obviously, due to the symmetry theorem, optimal algorithms for the product forms (mxn)(nxp), (nxm)(mxp), (mxp)(pxn) and their transposes must contain the same number of multiplication steps. This implies that complexity in terms of number of multiplications must be a function of all three variables, m, n, and p. That this function is not linear in the product mnp of the dimensions can be shown easily by examples such as the following. The 3 product forms (8x2)(2x2), (16x2)(2x1), (4x4)(4x2) all have 32 as the product of their dimensions. However, only the second form actually requires 32 multiplications (by Winograd (1969)). 28 multiplications are sufficient and necessary to compute products of the first form, whereas 26 suffice to compute (4x4)(4x2) by our example. No linear function of 32 could yield such a variety of lower bounds.

Upon closer inspection, we note that the greater the disparity among the size of dimensions, the greater the number of multiplications which are required. Put another way, the more symmetric the two matrices are in size, the fewer the number of multiplications necessary to compute their product. This appears to indicate that the lower bound function l(m,n,p) allocates weight to each dimension variable corresponding to its size in relation to the other two variables. In fact, we show by the next few results that a product of the maximum dimension with the sum of the other two yields a new lower bound for the complexity of matrix multiplication.

Theorem 9: (Kirkpatrick) Any algorithm over $Q[a_{11},\ldots,a_{mn},b_{11},\ldots,b_{np}]$, where Q, a_{ij}, b_{ij} are defined as before, which computes the matrix product $A_{mxn}B_{nxp}$, must employ at least m(n+p-1) multiplication steps.

Actually, Kirkpatrick's theorem is phrased in terms of independent variables in a field and active $*$-operations, but if we restrict the class of algorithms to those which do not use division, we obtain Theorem 9.

Corollary 9: If we consider only algorithms in NC, a product of the form (mxn)(nxp) requires m(n+p-1) multiplications.

This is an obvious corollary, but is included to demonstrate the difference in models.

Lemma 10: For all $m,n,p \geq 1$, let D be the triple $\{m,n,p\}$. Let m_x be the largest element in D. Let d_0, d_1 be the remaining two elements of D. Then,

$$m_x(d_0 + d_1 - 1) \geq d_0(m_x + d_1 - 1) .$$

Proof: $d_1 - 1 \geq 0$, since $d_1 \geq 1$.
Therefore, since $m_x \geq d_0$, $m_x(d_1-1) \geq d_0(d_1-1)$.
Also, $m_x d_0 \geq 1$. Therefore, $m_x d_0 + m_x(d_1-1) \geq m_x d_0 + d_0(d_1-1)$
Hence, $m_x(d_0+d_1-1) \geq d_0(m_x+d_1-1)$ since $m_x d_0 = d_0 m_x$.
Thus, multiplying by the maximum yields the largest possible result.

Theorem 11: Any algorithm which does not assume commutativity of multiplication and which computes a matrix product of the form (mxn)(nxp) must contain at least $m_x(d_0+d_1-1)$ multiplication steps where $m_x = \max\{m,n,p\}$

and $\{d_0,d_1\} = \{m,n,p\} - m_x$. (Here, D = $\{m,n,p\}$ is not a set but an unordered triple. If m=n=p, for example, we still treat D as having three components.) Thus, the complexity of any matrix multiplication algorithm is at least as great as the product of the larger dimension by one less than the sum of the remaining two dimensions.

Proof: By the Symmetry Theorem (theorem 8), the same number of multiplications is required to compute products of the form (mxn)(nxp) and $(m_x x d_0)(d_0 x d_1)$ where m_x = one of $\{m,n,p\}$, and $\{d_0,d_1\} = \{m,n,p\} - m_x$. In particular, we let $m_x = \max \{m,n,p\}$. By Theorem 9, $m_x(d_0+d_1-1)$ multiplications are required to compute an $(m_x x d_0)(d_0 x d_1)$ matrix product.

Observe that by Lemma 10, this lower bound is always as large as Kirkpatrick's (Theorem 9). In fact, whenever $m_x \neq m$, Theorem 11's lower bound is a strict improvement. To see this, repeat the proof of Lemma 10 with $d_0 = m$; the inequalities from the second line on become strict inequalities.

For example, products of the form (2x4)(4x4) require at least 4(2+4-1) = 20 multiplications, whereas Kirkpatrick's lower bound is 2(4+4-1) = 14 multiplications. This lower bound may not be optimal; the best algorithm to date, α' in the detailed example, uses 26 multiplications. However, Theorem 11 is not intended to yield achievable, and therefore best, lower bounds, but only to give a rough estimate of the essential complexity of general matrix multiplication. In fact, we can show that Theorem 11 does not necessarily produce achieveable lower bounds

by examining computations for the product form (2xn)(nx2) for $n \geq 3$. Kirkpatrick's lower bound is $2(n+2-1) = 2n + 2$ multiplications. The new lower bound is $n(2+2-1) = 3n$ multiplications, a significant improvement. However, by Corollaries 8b and 8c of the Symmetry Theorem, $\lceil 7n/2 \rceil$ multiplications is an achievable lower bound. Thus, further refinements of Theorem 11 are required; however, present techniques appear inadequate for this purpose.

Conclusion

The Symmetry Theorem corrects the invalid intuitive notion that one dimension in the product form (mxn)(nxp) determines the complexity of the computation; for example, n, the number of terms involved in brute-force inner-product multiplication. This theorem and its corollaries extend the few known lower bounds; in addition, the theorem can be implemented as demonstrated to find algorithms of equal complexity for symmetric computations of matrix products. This invariance of the complexity of symmetric problems appears capable of being extended to the usual matrix computations.

Unfortunately, the Symmetry Theorem, though a significant result for the theory of general matrix multiplication, offers no insights into the complexity of computing the product of square matrices. Thus, we have no new information about the exact complexity of multiplying even two 3x3 matrices. Detailed combinatorial analysis does not seem to be fruitful. As in other areas of Computational Complexity, (Munro and Borodin, 1972), new techniques must be developed before we can increase our knowledge of lower bounds.

Acknowledgement

This work was supported by the National Research Council of Canada.

References

Fiduccia, C., "Fast Matrix Multiplication", Proc. Third Annual Symp. on Theory of Computing, 45-49 (1971).

Fiduccia, C., "On Obtaining Upper Bounds on the Complexity of Matrix Multiplication", Proc. of the IBM Symp. on Complexity of Computer Computations (1972).

Hopcroft, J.E., and Kerr, L.B., "On Minimizing the Number of Multiplications Necessary for Matrix Multiplication", Cornell University Technical Report 69-44 (1969).

Kirkpatrick, D., Personal Communication

Munro, I. and Borodin, A., "Efficient Evaluation of Polynomial Forms", University of Waterloo Research Report CSTR 1013 (1972).

Strassen, V., "Gaussian Elimination is not Optimal", Numer. Math. 13, 354-356 (1969).

Winograd, S., "On the Number of Multiplications Required to Compute Certain Functions", Proc. N.A.S., Vol. 58, 1840-1842 (1967).

Winograd, S., "On Multiplication of 2 x 2 Matrices", Linear Algebra and Its Applications 4, 381-388 (1971a).