Matrix algorithms

Mike Paterson

CS341 Topics in Algorithms
Spring Term 2009

1 Introduction

In studying the complexity of algorithms we develop techniques for evaluating the amount of "resource", usually time or storage space, used by new or existing programs; we attempt to prove lower bounds for the resources required by any program which performs a given task; we look for interesting relationships among different algorithms for the same problem or explore possible connections between seemingly unconnected problem areas; and in all we aim for a deeper understanding of the essential difficulties of, and possible solutions to, a variety of computational problems.

In this note we consider matrices. Matrix methods have important applications in many scientific fields, and frequently account for large amounts of computer time. The practical benefit from improvements to algorithms is therefore potentially very great. The basic algorithms, such as matrix multiplication are simple enough to invite total comprehension, yet rich enough in structure to offer challenging mathematical problems and some elegant solutions. The subject matter is well enough known for us to start immediately without an extensive introduction.

2 Definitions of matrix arithmetic

If A is a $p \times q$ matrix and B a $q \times r$ matrix then their product $C = A \cdot B$ is a $p \times r$ matrix with entries given by $c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}$ for $i = 1, \ldots, p$ and $j = 1, \ldots, r$.

Sometimes it is useful to think of A as composed of its p row vectors $A_1, \ldots, A_p$, and B as composed of its r column vectors $B_1, \ldots, B_r$. Then c_{ij} is the inner product of vectors A_i and B_j. The sum of two matrices A, B with the same dimensions is the matrix $C = A + B$ given by $c_{ij} = a_{ij} + b_{ij}$ for all i, j.

3 Arithmetic complexity

A computer program for an arithmetic algorithm will usually execute many instructions other than the explicit arithmetic operations of the algorithm. There will, for example, be fetching, storing, loading and copying operations. The properties of cache memory and external storage devices can have a huge effect on running times. The proportion of the total execution time

which is spent on such "overheads" will be very dependent on the computer and programming language used. For simplicity and independence we shall often take account only of the arithmetic operations involved. This measure will be referred to as the *arithmetic complexity*. The consequences of this simplification in particular practical applications must of course be carefully considered.

In the product of a $p \times q$ matrix by a $q \times r$ matrix (a $p \times q \times r$ *product*) each of the pr entries of the product can be computed using q multiplications and $q-1$ additions. We can write this arithmetic complexity as $q\,\underline{\mathrm{m}} + (q-1)\,\underline{\mathrm{a}}$ and then get a total for the $(p \times q \times r)$-product of $pqr\,\underline{\mathrm{m}} + p(q-1)r\,\underline{\mathrm{a}}$. The sum of two $p \times q$ matrices uses only $pq\,\underline{\mathrm{a}}$.

We can think of $\underline{\mathrm{m}}$ and $\underline{\mathrm{a}}$ as formal symbols to keep track of the numbers of each type of arithmetic operation, or else we can think of these as the time or cost of each operation and thus have an expression for the total cost of the operations. We will never distinguish between the complexity of a basic addition and a subtraction and such an operation will be referred to as an *addition/subtraction* (a/s). Similarly we may sometimes write *multiplication/division* (m/d).

The kinds of question to which we shall seek answers are: "Can product be computed by another algorithm using fewer operations?" "What is the minimum number of arithmetic operations required?"

The first question is answered affirmatively; the second has as yet only very incomplete answers.

4 Winograd's algorithm for matrix product

To compute $a_1 \cdot b_1 + a_2 \cdot b_2$ certainly requires 2 multiplications/divisions (and 1 addition/subtraction), and more generally, $a_1 \cdot b_1 + \cdots + a_n \cdot b_n$ requires n multiplication/divisions. An alternative way to compute $a_1 \cdot b_1 + a_2 \cdot b_2$ is the following:

$$\begin{aligned} \mu_1 &= a_1 \cdot a_2 \\ \mu_2 &= b_1 \cdot b_2 \\ \mu_3 &= (a_1 + b_2) \cdot (a_2 + b_1) \\ \text{result} &= \mu_3 - \mu_1 - \mu_2. \end{aligned}$$

It needs good insight to see the significance for matrix product of this identity which, at first glance, appears merely to take more multiplications and more additions than the obvious algorithm. The important feature is that μ_1 and μ_2 are multiplications which involve only a's and only b's respectively. Why is this so important?

We have already remarked that matrix product can be regarded as finding the inner product of each row of one matrix with each column of the other matrix. In the sub-algorithm used for inner product, if there is a computation involving the elements from only one of the vectors then it can be performed just once for that row (column) instead of every time that vector is used. This idea of "pre-processing" is very important and leads in this instance to Winograd's algorithm [9]. The algorithm is described first for the simple case of $n \times n$ matrices with n even. For $x = (x_1, \ldots, x_n)$, define

$$W(x) = x_1 \cdot x_2 + x_3 \cdot x_4 + \cdots + x_{n-1} \cdot x_n.$$

Then
(i) For each row A_i of A compute $W(A_i)$, and for each column B_j of B compute $W(B_j)$.
(ii) For each pair (i,j), writing a for A_i and b for B_j, compute

$$a \cdot b = (a_1 + b_2) \cdot (a_2 + b_1) + (a_3 + b_4) \cdot (a_4 + b_3) + \cdots + (a_{n-1} + b_n) \cdot (a_n + b_{n-1}) - W(a) - W(b).$$

The arithmetic complexity for (i) is

$$2n(n/2\ \underline{\mathrm{m}} + (n/2 - 1)\,\underline{\mathrm{a}}),$$

and for (ii) is

$$n^2(n/2\ \underline{\mathrm{m}} + (3n/2 + 1)\,\underline{\mathrm{a}}),$$

which gives a total of

$$(\tfrac{1}{2}n^3 + n^2)\,\underline{\mathrm{m}} + (\tfrac{3}{2}n^3 + 2n(n-1))\,\underline{\mathrm{a}} = \tfrac{1}{2}n^3\,\underline{\mathrm{m}} + \tfrac{3}{2}n^3\,\underline{\mathrm{a}} + O(n^2).$$

Neglecting lower order terms, we have exchanged roughly $n^3/2$ multiplications for an extra $n^3/2$ additions/subtractions. The algorithm is easily extended to the general $p \times q \times r$ product. If q is even the algorithm is essentially the same. If q is odd then one elementary multiplication in each inner product is done in the conventional manner and added in separately, which does not significantly affect the arithmetic complexity. The extra storage requirements of Winograd's algorithm are minimal: just one extra location for each row and column is needed to store the value of W.

This algorithm is of possible value whenever $\underline{\mathrm{m}} > \underline{\mathrm{a}}$. Typical applications are when the matrix elements are complex numbers or multiple-precision numbers. A significant restriction of the algorithm is that its correctness depends on the commutativity of multiplication. This is seen in the original identity for $a_1b_1 + a_2b_2$ above.

Let us consider the case of complex matrices in further detail. Assuming that the complex numbers are represented by pairs of reals giving their real and imaginary parts, the obvious algorithm to compute

$$(x + iy) \cdot (u + iv) = (xu - yv) + i(xv + yu)$$

takes $4\,\underline{\mathrm{m}} + 2\,\underline{\mathrm{a}}$, and complex addition costs $2\,\underline{\mathrm{a}}$. This seems a good application for Winograd's algorithm. If we are on the look-out for unusual methods, we may find the following alternative for complex product:

$$\begin{aligned} \lambda_1 &= x \cdot u \\ \lambda_2 &= y \cdot v \\ \lambda_3 &= (x + y) \cdot (u + v). \end{aligned}$$

Then

$$(x + iy) \cdot (u + iv) = (\lambda_1 - \lambda_2) + i(\lambda_3 - \lambda_1 - \lambda_2).$$

Although this identity is reminiscent of the identity underlying Winograd's algorithm, note that commutativity of multiplication need not be assumed here. Since this method uses $3\,\underline{\mathrm{m}} + 5\,\underline{\mathrm{a}}$, instead of $4\,\underline{\mathrm{m}} + 2\,\underline{\mathrm{a}}$, it requires a situation where $\underline{\mathrm{m}}$ is much larger than $\underline{\mathrm{a}}$ to be useful. If the elements involved are themselves large matrices then this condition holds.

This observation yields a new class of algorithms for complex matrix product. Note the relevance of the remark above about commutativity. Given complex matrices, A and B, split them into their real and imaginary parts so that we may write

$$A = X + iY, \quad B = U + iV,$$

where X, Y, U, V are *real* matrices. Then the identity above is used to compute $A \cdot B$ using only 3 real matrix products and 5 real matrix sums.

We now have a plethora of algorithms to consider, of which we identify eight. Given two complex matrices, these may be multiplied directly using either the Classical method (C) or Winograd's algorithm (W), and then the complex entries can be multiplied in the Straightforward way (S) or the Unusual (Underhand?) way (U) given by the above identity. We can denote these methods by

$$\mathrm{CS}, \mathrm{CU}, \mathrm{WS}, \mathrm{WU}.$$

Alternatively the original matrices may be split up into real and imaginary parts and multiplied using methods S or U. The real matrix products required are done by C or W, yielding four more methods,

$$\mathrm{SC}, \mathrm{UC}, \mathrm{SW}, \mathrm{UW}.$$

We shall analyse the arithmetic complexity of these methods for $n \times n \times n$ product as the ratio of $\underline{\mathrm{m}}$ to $\underline{\mathrm{a}}$ varies. This is only a theoretical exercise since in practice the "overheads" may be the crucial criterion in a comparison of similar algorithms. We set out in the table below the leading coefficients of the $\underline{\mathrm{m}}$ and $\underline{\mathrm{a}}$ components of the arithmetic complexity.

Method	Coefficient of n^3	Coefficient of n^3
CS	4	4
CU	3	7
WS	2	4
WU	$1\frac{1}{2}$	$5\frac{1}{2}$
SC	4	4
SW	2	6
UC	3	3
UW	$1\frac{1}{2}$	$4\frac{1}{2}$

As one would expect from the above discussion, if one is going to split up the matrices initially, the first stage should be done with U rather than S, and if the matrices are to be multiplied directly, W is better than C. Looking at the remaining complexities we find that

1. if $\underline{\mathrm{m}} > \underline{\mathrm{a}}$ then UW has the lowest;
2. if $\underline{\mathrm{m}} < \underline{\mathrm{a}}$ then UC has the lowest;
3. if $\underline{\mathrm{m}} = \underline{\mathrm{a}}$ then WS, UW, UC are the joint leaders,

but if lower order terms are taken into account in case 3, then UC has the lowest arithmetic complexity:

$$3n^3\,\underline{\mathrm{m}} + (3n^3 + 2n^2)\,\underline{\mathrm{a}}.$$

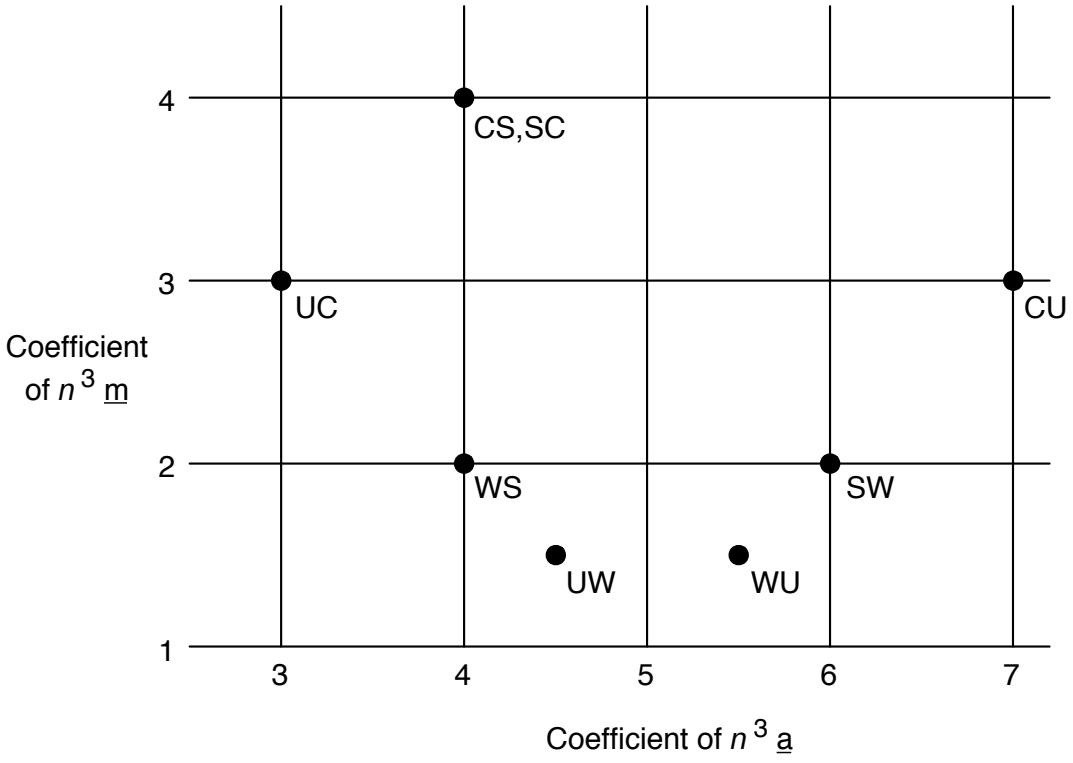

Figure 1: Comparison of eight algorithms

In practice, the methods using an initial "U" splitting have a significant disadvantage since these programs require three matrix multiplications instead of just one, and a large part of the total execution time may be concerned with initialization and calculating the indices and addresses of the arguments for operations. A promising approach which I have not tried out in practice but which may overcome some of the inefficiencies in methods such as UC and UW is the following. We take advantage of the circumstance that there are three real matrix products, all of the same dimensions, to be computed and that they may be performed in parallel. If the corresponding operations of these products are interleaved then some of the "overheads" can be shared.

5 A recursive method and recurrence relations

For a different style of algorithm for matrix product we can use partitioned matrices and "block multiplication". To simplify matters suppose A, B are $n \times n$ matrices with $n > 1$. If we regard A, B as composed of submatrices in the following way

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), \quad \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right),$$

where A_{11}, B_{11} are $r \times r$ matrices, $0 < r < n$, then the product is given by

$$\left(\begin{array}{c|c} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ \hline A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{array}\right).$$

That the result is correct is easily proved, and uses only the associativity property of addition. The product $A \cdot B$ is thus computed by performing 8 products of the sub-matrices, followed by 4 sums of the resulting sub-matrices. The sub-matrix products may be done in a similar manner by further partitioning into smaller matrices, and so on until the resulting matrices are small, maybe 1×1. Thus we have a recursive procedure for matrix product. If we take $r = \lceil n/2 \rceil$, so the partitioning is as nearly as possible into equal parts, and if we write $P(n)$, $S(n)$, for the arithmetic complexity of $n \times n \times n$ product and $n \times n$ sum respectively, we derive the following recurrence relation.

$$P(n) \leq 8P(\lceil n/2 \rceil) + 4S(\lceil n/2 \rceil).$$

But $S(n) = n^2 \, \underline{\mathrm{a}} = O(n^2)$ operations, so we have $P(n) \leq 8P(\lceil n/2 \rceil) + O(n^2)$.
Recurrence relations of the above form occur frequently and we give below a general solution. For the above relation this will imply that

$$P(n) = O(n^{\log_2 8} = O(n^3).$$

This comes as no surprise to the observant reader who has seen that precisely the same multiplications are performed as in the "classical" algorithm and the additions have just been rearranged using associativity.

Theorem 1 *If F is a non-negative function on the positive integers such that for some $a \geq 1, b > 1$ and $\beta \geq 0$, $F(n) \leq a.F(\lceil n/b \rceil) + O(n^\beta)$ then if $\alpha = \log_b a$:*

$$\begin{aligned} F(n) &= O(n^\alpha) && \text{if } \alpha > \beta, \\ &= O(n^\beta) && \text{if } \alpha < \beta, \\ &= O(n^\alpha \log n) && \text{if } \alpha = \beta. \end{aligned}$$

Proof. Left to the reader!

6 Strassen's algorithm

In the light of Winograd's algorithm it would be tempting to conjecture that, while some trade-off between multiplications and additions is possible, the total number of arithmetic operations required is of order n^3 for $n \times n \times n$ product. This is not so! Strassen's simple and astonishing observation [7] is that for multiplying 2×2 matrices only 7 (not 8) multiplications are needed, even if multiplication of elements is non-commutative. Using this fact, the block multiplication algorithm described in the last section may be up-graded to one satisfying:

$$P(n) \leq 7P(\lceil n/2 \rceil) + O(n^2),$$

which, by the theorem given above, yields

$$P(n) = O(n^{\log_2 7}) = O(n^{2.81}).$$

Recall that $P(n)$ is the *total number* of arithmetic operations (multiplications, additions/subtractions). It should be apparent that with a straightforward implementation of this algorithm on a machine with reasonable properties the total execution time is also of the stated order.

6.1 Strassen's identities

We assume for simplicity that A, B are $n \times n$ matrices, and that n is even so the matrices can be partitioned into 4 equal quarter-matrices. For

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \cdot \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right),$$

compute:

$$\begin{aligned} m_1 &= (A_{11} + A_{21})(B_{11} + B_{12}) \\ m_2 &= (A_{12} + A_{22})(B_{21} + B_{22}) \\ m_3 &= (A_{11} - A_{22})(B_{11} + B_{22}) \\ m_4 &= A_{11}(B_{12} - B_{22}) \\ m_5 &= (A_{21} + A_{22})B_{11} \\ m_6 &= (A_{11} + A_{12})B_{22} \\ m_7 &= A_{22}(B_{21} - B_{11}). \end{aligned}$$

Then

$$\begin{aligned} C_{11} &= m_2 + m_3 - m_6 - m_7 \\ C_{12} &= m_4 + m_6 \\ C_{21} &= m_5 + m_7 \\ C_{22} &= m_1 - m_3 - m_4 - m_5. \end{aligned}$$

Thus,

$$P(n) = 7P(n/2) + 18S(n/2)$$

and so

$$P(n) = O(n^{\log_2 7}).$$

These identities may be conveniently expressed in the form of a diagram, where $\bullet(\circ)$ in cell (A_{ij}, B_{kl}) represents the term $+(-)A_{ij}B_{kl}$. The connected groups of circles represent the terms occurring in the respective products. It is now easy to verify the correctness of the identities.

A small improvement may be obtained by applying linear transformations to the above identities to reduce the number of matrix sums required from 18 to 15. Of course this has no effect on the exponent, $\log_2 7$, but reduces the arithmetic complexity by a constant factor. The resulting identities are given below. A curious feature is that the first two of the seven products are $A_{11}B_{11}$ and $A_{12}B_{21}$ which would also be those done by the obvious block multiplication.

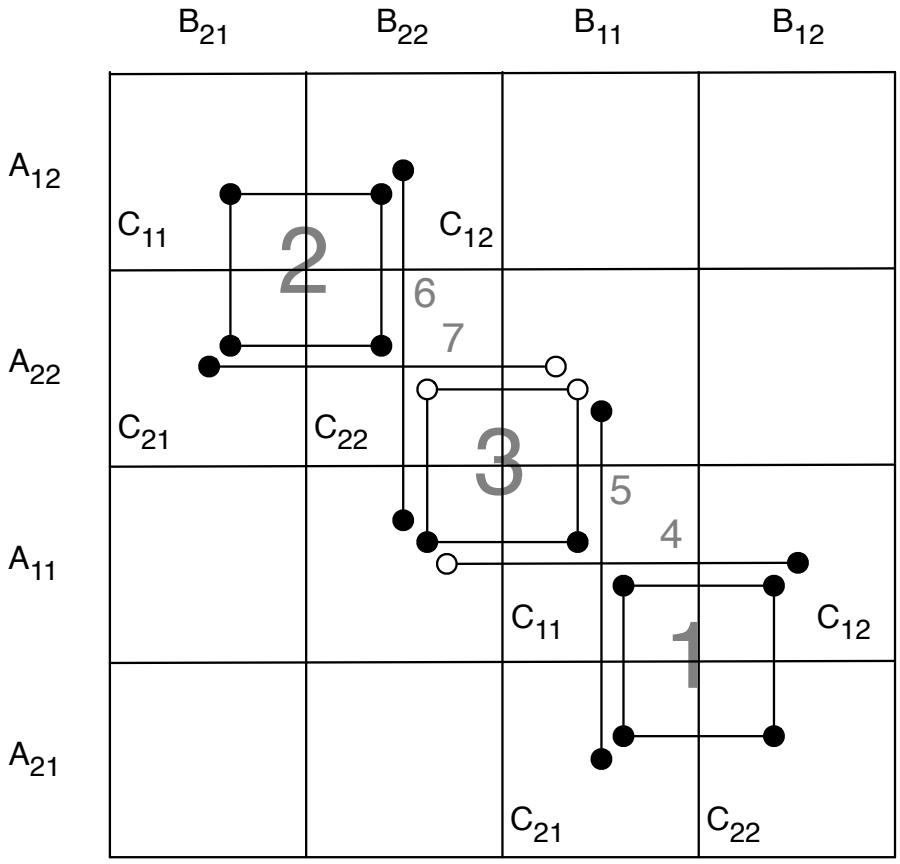

Figure 2: Strassen diagram

$$
\begin{aligned}
m_1 &= A_{11}B_{11} \\
m_2 &= A_{12}B_{21} \\
m_3 &= (-A_{11} + A_{21} + A_{22})(B_{11} - B_{12} + B_{22}) \\
m_4 &= (A_{11} - A_{21})(-B_{12} + B_{22}) \\
m_5 &= (A_{21} + A_{22})(-B_{11} + B_{12}) \\
m_6 &= (A_{11} + A_{12} - A_{21} - A_{22})B_{22} \\
m_7 &= A_{22}(-B_{11} + B_{12} + B_{21} - B_{22}).
\end{aligned}
$$

Then

$$
\begin{aligned}
C_{11} &= m_1 + m_2 \\
C_{12} &= m_1 + m_3 + m_5 + m_6 \\
C_{21} &= m_1 + m_3 + m_4 + m_7 \\
C_{22} &= m_1 + m_3 + m_4 + m_5.
\end{aligned}
$$

Note the claimed 15 additions is only achieved by a careful sharing of common terms. (Check this!) Probert [6] has shown that 15 is optimal.

7 Some related results

Can the $2 \times 2 \times 2$ product be computed using fewer than 7 multiplications? Winograd [10] shows that, even if multiplication is commutative, 7 is the optimal number. Hopcroft and Musinski [3] show that for any non-commutative ring obtained by adjoining indeterminates to a commutative ring, *every* algorithm with 7 multiplications for $2 \times 2 \times 2$ product can be got by applying linear transformations to Strassen's algorithm. An example is provided by the two sets of identities given above.

The *tensor formulation* of the matrix multiplication problem [8, 2] has a symmetry which shows that the minimum number of multiplications required is the same for $p \times q \times r$, $p \times r \times q$, $q \times r \times p$, $q \times p \times r$, $r \times p \times q$ and $r \times q \times p$ products, and thus depends only on the triple $\{p, q, r\}$. Using results from [4] with this result we have that the minimal number for the triple $\{p, q, 2\}$ is $\lceil (3pq + \max(p, q))/2 \rceil$, for $p \leq 2$ or $p = q = 3$, e.g., 7 for $p = q = 2$, and 15 for $p = q = 3$. It is clear that any improvement on Strassen's bound using the same kind of recursion has to be based on a larger basic product than $2 \times 2 \times 2$.

If 3×3 matrices could be multiplied using only 21 multiplications (non-commutative) then a faster algorithm would be obtained since $\log_3 21 < \log_2 7$. Nothing better than 24 has yet been achieved, but neither has any close lower bound been proved. For 4×4 matrices, obviously 48 would need to be achieved. A recursion could be based also on non-square decompositions. The results of [3] show that a result of k multiplications for $p \times q \times r$ product, yields k^3 for $pqr \times pqr \times pqr$ product and hence an exponent for n of $3 \log_{pqr} k$.

In 1980, Victor Pan published an algorithm for $70 \times 70 \times 70$ product using 143640 multiplications [5]. Note that $\log_{70} 143640 < 2.796$. Over the following few years the best exponent known gradually dropped. The current record is still the 2.376 due to Coppersmith and Winograd [1]. However, because of the huge constant factors involved, the only sub-cubic algorithm with any present claim to practicality is Strassen's.

In an algorithm for the product of matrices of arbitrary shapes and sizes it is very inefficient merely to fill out the matrices with 0's to the next power of two. Halving each dimension and adding one row or column of 0's is more efficient, but the best strategy involves partitioning into varying sizes, using some of the non-square matrix recurrences, and transferring to Winograd's or the classical method for small matrices. It is certainly inefficient to use Strassen's recursion right down to 1×1 matrices.

The idea mentioned in Section 2 for sharing some of the non-arithmetic overheads by performing several matrix products in parallel would seem to be useful in an implementation of Strassen's algorithm also. Care must be exercised however to avoid an unacceptable increase in the storage required.

8 Reductions and equivalences to matrix product

In Strassen's original paper [7], he also shows how any fast matrix product algorithm yields a correspondingly fast algorithm for matrix inversion and computing determinants. These reductions are based on the following "block LDU factorization" formula which is easily verified.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & O \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & O \\ O & \Delta \end{pmatrix} \begin{pmatrix} I & A_{11}^{-1}A_{12} \\ O & I \end{pmatrix}$$

if A_{11} is non-singular, where I is the unit matrix, O is the zero matrix and $\Delta = A_{22} - A_{21}A_{11}^{-1}A_{12}$. So,

$$\begin{aligned} A^{-1} &= \begin{pmatrix} I & -A_{11}^{-1}A_{12} \\ O & I \end{pmatrix} \begin{pmatrix} A_{11}^{-1} & O \\ O & \Delta^{-1} \end{pmatrix} \begin{pmatrix} I & O \\ -A_{21}A_{11}^{-1} & I \end{pmatrix} \\ &= \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}\Delta^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}\Delta^{-1} \\ -\Delta^{-1}A_{21}A_{11}^{-1} & \Delta^{-1} \end{pmatrix} \end{aligned}$$

provided Δ is also non-singular. Assuming these non-singularities, we have immediately the recurrence relation for $I(n)$, the arithmetic complexity of inverting an $n \times n$ matrix, given by

$$I(n) \leq 2I(\lceil n/2 \rceil) + O(P(\lceil n/2 \rceil) + O(n^2).$$

If we assume an algorithm for product giving $P(n) = O(n^\alpha)$, for some $\alpha \geq 2$, the general solution given in Section 3 yields

$$I(n) = O(n^\alpha).$$

Similarly, from the LDU factorization, we have

$$\mathrm{Det}\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \mathrm{Det}(A_{11})\mathrm{Det}(\Delta).$$

If $D(n)$ is the arithmetic complexity for determinants we have the recurrence

$$D(n) \leq 2D(\lceil n/2 \rceil) + I(\lceil n/2 \rceil) + O(P(\lceil n/2 \rceil))$$

and so, with the same hypothesis,

$$D(n) = O(n^\alpha).$$

The algorithm for inversion uses block LDU factorization recursively and so will fail, even if A is non-singular, whenever "A_{11}" or "Δ" at *any level* of the recursion happens to be singular. An elegant way around this difficulty is given by the following results.

A non-singular matrix A is *positive-definite* if $x^{\mathrm{T}}Ax > 0$ for all non-zero vectors x.

Theorem 2 *For any non-singular matrix A, the matrix $A^{\mathrm{T}}A$ is positive-definite.*

Proof:

$$x^{\mathrm{T}}(A^{\mathrm{T}}A)x = (Ax)^{\mathrm{T}}(Ax) = ||Ax||^2 \geq 0.$$

If $x \neq 0$ then $Ax \neq 0$, and so $||Ax||^2 > 0$. □

Theorem 3 *For any non-singular matrix A,*

$$A^{-1} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}.$$

Proof:

$$((A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}})A = (A^{\mathrm{T}}A)^{-1}(A^{\mathrm{T}}A) = I.$$

□

In Strassen's algorithm, it can be shown that if A is positive-definite then every matrix which needs to be inverted in the recursive calls is also positive-definite. Hence, matrix inversion can be accomplished via Theorem 3 by applying Strassen's algorithm to $A^{\mathrm{T}}A$.

Is it possible that $I(n)$ is of *lower* order than $P(n)$? We show directly that this is not so. It is easily verified that:

$$\begin{pmatrix} I & A & O \\ O & I & B \\ O & O & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -A & AB \\ O & I & -B \\ O & O & I \end{pmatrix}.$$

Thus, to find the product of two $n \times n$ matrices A, B, it is sufficient to invert an appropriately constructed non-singular $3n \times 3n$ matrix. We therefore have

$$P(n) \leq I(3n).$$

Combining this with a previous result we obtain:

Theorem 4 *For all $\alpha \geq 2$,*

$$P(n) = O(n^{\alpha}) \Longleftrightarrow I(n) = O(n^{\alpha}).$$

A similar result for squaring matrices follows from

$$\begin{pmatrix} O & A \\ B & O \end{pmatrix}^2 = \begin{pmatrix} AB & O \\ O & BA \end{pmatrix}.$$

References

[1] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 1–6, New York, NY, USA, 1987. ACM Press.

[2] Charles M. Fiduccia. On obtaining upper bounds on the complexity of matrix multiplication. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of computer computations*, pages 31–40, New York, 1972. Plenum Press.

[3] J. Hopcroft and J. Musinski. Duality applied to the complexity of matrix multiplications and other bilinear forms. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 73–87, New York, NY, USA, 1973. ACM Press.

[4] L.R. Kerr J.E. Hopcroft. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM J. Applied Math.*, 20(1):30–36, 1971.

[5] Victor Y. Pan. New fast algorithms for matrix operations. *SIAM J. Comput.*, 9(2):321–342, 1980.

[6] Robert L. Probert. On the additive complexity of matrix multiplication. *SIAM J. Comput.*, 5(2):187–203, 1976.

[7] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[8] Volker Strassen. Evaluation of rational functions. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of computer computations*, pages 1–10, New York, 1972. Plenum Press.

[9] Shmuel Winograd. On the number of multiplications necessary to compute certain functions. *Communications on Pure and Applied Mathematics*, 23:165–179, 1970.

[10] Shmuel Winograd. On multiplication of 2×2 matrices. *Linear Algebra and its Applications*, 4:381–388, 1971.