How Can We Speed Up Matrix Multiplication?

Victor Pan

SIAM Review, Vol. 26, No. 3 (Jul., 1984), 393-415.

Stable URL:
http://links.jstor.org/sici?sici=0036-1445%28198407%2926%3A3%3C393%3AHCWSUM%3E2.0.CO%3B2-9

SIAM Review is currently published by Society for Industrial and Applied Mathematics.

SIAM REVIEW
Vol. 26, No. 3, July 1984

005

HOW CAN WE SPEED UP MATRIX MULTIPLICATION?*

VICTOR PAN†

Abstract. Due to the new algebraic methods of algorithm design, recently it became possible to perform multiplication and inversion of $N \times N$ matrices using $O(N^{2.496})$ rather than $O(N^3)$ arithmetical operations. Consequently, algorithms for several other computational problems of linear algebra and combinatorics have been accelerated. The major ideas and techniques that have led to that progress are surveyed.

Introduction. *Matrix multiplication* (hereafter referred to as MM) is a basic operation of linear algebra, which has numerous applications to the theory and practice of computation. In particular, several important applications are due to the fact that MM is a substantial part of several successful algorithms for other computational problems of linear algebra and combinatorics, such as the solution of a system of linear equations, matrix inversion, the evaluation of the determinant of a matrix, Boolean MM, and the transitive closure of a graph. Moreover, the computational time required for MM is the dominant part of the total computational time required for all of those problems, that is, all such problems can be reduced to MM and can be solved fast if MM is solved fast.

How fast can we multiply matrices? The product of a pair of $N \times N$ matrices X and Y can be evaluated in a straightforward way using N^3 multiplications of the entries of X and Y and $N^3 - N^2$ additions of the resulting products. The best upper bound on the number of arithmetical operations for MM, and for all related problems listed above, remained $O(N^3)$ until 1968, while the best lower bounds on that number were of order N^2. The best lower bound is still of the same order, but since 1968 the "upper" exponent 3 has been reduced to the currently record value 2.496.

Such an improvement of "very natural" $O(N^3)$ algorithms is theoretically important, even though any substantial impact of the asymptotically fast MM algorithms on the practice of computing matrix products remains questionable, due to the overhead of those algorithms. On the other hand, historically the design of fast algorithms for MM required the introduction of some new ideas and techniques that are now commonly used for the design of fast algorithms for other theoretically and practically important algebraic and numerical problems. We will survey the recent progress in the acceleration of MM, including all major ideas and techniques of that kind (see [18] for a more extensive survey). We hope that our paper will introduce some readers to the subject of algebraic computing (see more about that topic in the books [1], [4], [9], [26]). Our presentation is elementary and does not require any preliminary knowledge, although some of the results presented, if stated in terms of ranks of tensors and of their direct sums, could be of certain interest for pure mathematics, see, in particular, Remark 8.2 in §8.

We will use the following order of presentation. In §1 we follow the paper [23] and present a nontrivial algorithm for 2×2 MM, used as a basis of a recursive construction that leads to performing $N \times N$ MM for *all* N in $O(N^{2.808})$ arithmetical operations. In §2 we generalize that construction and reduce the problem of the asymptotic acceleration of MM to the design of a basic algorithm of a certain type (that is, formally, of a bilinear algorithm of a smaller rank) for a problem of multiplying some matrices of a particular size, say, 3×3 or 70×70. In §3 we sketch the recent history of the search for the successful basic algorithms. In §4 we present a basic algorithm that immediately leads to the exponent 2.67. (After a simple modification, that basic algorithm leads to even

*Received by the editors July 25, 1983, and in revised form December 13, 1983. This research was supported by the National Science Foundation under grant MCS 820 3232.

†Computer Science Department, State University of New York at Albany, Albany, New York 12222.

smaller exponents and, furthermore, when used within a slightly involved recursive construction, it leads to the record exponent 2.496, see [18].) Since the basic algorithm of §4 is somewhat irregular, we need to justify the exponent 2.67. We do that in §§5–8, by generating two recursive constructions based on such irregular algorithms, where their accelerating power is accumulated. (When such a power becomes large enough, its relatively small part is sacrificed in order to transform the resulting recursive algorithms into regular form.) This leads us to the study (in §§5–8) of the important general class of bilinear λ-algorithms (also known as APA-algorithms) for MM and Disjoint MM. (During that study we disprove the extension of the famous direct sum conjecture [24] to the class of λ-algorithms.) In §§9 and 10 we define the equivalent trilinear representation of bilinear algorithms, and show some applications of that representation to the design of efficient algorithms for MM. In particular, in §10 we introduce the general techniques of trilinear aggregating (TA) that stand behind the design of our basic algorithm of §4 and its modifications. Finally, in §11 we reduce matrix inversion and Boolean MM to MM.

1. The power of recursive algorithms for matrix multiplication. The following important algorithm was discovered by Volker Strassen in 1968 (see [23]).

ALGORITHM FOR 2×2 MM. Evaluate the product $Q = XY$ of two given 2×2 matrices

$$X = \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix}, \qquad Y = \begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix}$$

using the following formulae:

$$p_0 = (x_{00} + x_{11})(y_{00} + y_{11}), \quad p_1 = (x_{10} + x_{11})y_{00}, \quad p_2 = x_{00}(y_{01} - y_{11}),$$
$$p_3 = (-x_{00} + x_{10})(y_{00} + y_{01}), \quad p_4 = (x_{00} + x_{01})y_{11}, \quad p_5 = x_{11}(-y_{00} + y_{10}),$$
$$p_6 = (x_{01} - x_{11})(y_{10} + y_{11}),$$
$$q_{00} = p_0 + p_5 - p_4 + p_6, \quad q_{01} = p_2 + p_4, \quad q_{10} = p_1 + p_5, \quad q_{11} = p_0 - p_1 + p_2 + p_3.$$

For any pair of 2×2 matrices X, Y, this algorithm evaluates the matrix XY in 7 multiplications and 18 additions/subtractions. The straightforward method involves 8 multiplications and 4 additions and seems more efficient. However, this impression changes if we apply Strassen's algorithm recursively in order to evaluate a $2^h \times 2^h$ matrix product.

Indeed, assume that the 8 inputs $x_{00}, x_{01}, \cdots, y_{11}$ of Strassen's algorithm are not just numbers or indeterminates but that they are themselves 2×2 matrices. Then so are $p_0, p_1, \cdots, p_6, q_{00}, q_{01}, q_{10}, q_{11}$. The operations of the algorithm become operations over 2×2 matrices. Using Strassen's algorithm again in order to perform each of the seven 2×2 MM and summing and subtracting matrices in the straightforward way we need perform only $7^2 = 49$ multiplications and $7*18 + 4*18 = 198$ additions/subtractions. Note that the matrices X, Y, Q become 4×4 matrices, so that we arrive at an algorithm for 4×4 MM. As an illustration, here is the block representation of Q (and similarly for X, Y).

$$Q = \left[\begin{array}{cc|cc} q_{00} & q_{01} & q_{02} & q_{03} \\ q_{10} & q_{11} & q_{12} & q_{13} \\ \hline q_{20} & q_{21} & q_{22} & q_{23} \\ q_{30} & q_{31} & q_{32} & q_{33} \end{array}\right] = \begin{bmatrix} Q_{00} & Q_{01} \\ Q_{10} & Q_{11} \end{bmatrix}.$$

The broken lines separate the 2×2 blocks $Q_{00}, Q_{01}, Q_{10}, Q_{11}$ from each other.

Similarly substitute 4×4 matrices for the entries of the original matrices X, Y and continue the above recursive process of constructing algorithms for $2^h \times 2^h$ MM, $h = 3, 4, \cdots$ in this way. In this paragraph let $m(N)$ and $a(N)$ designate the numbers of multiplications and additions/subtractions, respectively, required for $N \times N$ MM in such recursive algorithms. Then we immediately deduce that $m(2^h) = 7^h$, $a(1) = 0$, $a(2^{h+1}) = 18*4^h + 7a(2^h)$, $h = 1, 2, \cdots$ Consequently, $a(h) = 6(7^h - 4^h)$, so that $m(h) + a(h) < 7^{h+1}$ for all h (compare [23] or, say, [4, p. 4]). The algorithm is extended to $N \times N$ MM for all N by padding matrices with zeros if N is not in the form 2^h. A total of at most $4.7N^\omega = O(N^\omega)$, $\omega \leqq \log_2 7 = 2.807 \cdots$ arithmetical operations suffice for $N \times N$ MM. Here the *exponent ω is defined by the number of multiplications in the basic algorithm.* (Additions/subtractions contribute little to the total computational cost because they are substantially simpler than multiplications if we perform all operations over large matrices as we do in this recursive construction.)

Since $2.807 \cdots < 3$, we have *asymptotically accelerated* the "natural" straightforward method for $N \times N$ MM which uses $2N^3 - N^2$ operations. The computation for several other important graph theoretical and linear algebra problems (in particular, for the transitive closure of a graph (see [1] or [6], compare also our §11) and for simultaneous linear equations (see our §11 or see [1], [4], [23])) can be reduced to MM and consequently accelerated, so that the exponent $\omega \leqq 2.807 \cdots$, soon after Strassen's discovery, became one of the most frequently cited quantities in the study of the time-complexity of algorithms. Strassen's result also raised the question whether the exponent $\log_2 7$ can be reduced further or, more generally, what is the greatest lower bound ω on the exponents, such that $N \times N$ MM can be performed involving a total of $O(N^{\omega+\varepsilon})$ arithmetical operations for arbitrary positive ε? (We cautiously wrote $O(N^{\omega+\varepsilon})$ rather than just $O(N^\omega)$ allowing ω to be a limiting point rather than just the minimum value.) We will call such ω the *limiting exponent of MM* or, simply, just the *exponent of MM.*

It is immediately seen that

$$\omega \geqq 2. \tag{1.1}$$

Indeed, each arithmetical operation involves 2 inputs and has 1 output while each $N \times N$ MM problem has $2N^2$ inputs and N^2 outputs. (1.1) remains the best lower bound on the exponent ω that is presently known.

2. Bilinear algorithms for MM. The development described in the previous section came into being due to the successful design for 2×2 MM in 7 multiplications. (Recall that the number of additions/subtractions in that basic algorithm does not influence the exponent.) More generally, if for some pair n, M there exists an algorithm for $n \times n$ MM that involves only M multiplications and gives rise to a Strassenlike recursive construction then

$$\omega \leqq \log_n M = \log M / \log n. \tag{2.1}$$

We have $n = 2$, $M = 7$, $\omega \leqq \log_2 7$ for Strassen's algorithm. Next we will define quite a general class of algorithms which can all be used as the bases of the recursive constructions. As in the case of Strassen's algorithm, we will require that the computation scheme work for all instances of the given input matrices, so that we will consider their entries independent variables (indeterminates).

DEFINITION 2.1. *Bilinear algorithms for* MM. Given two $m \times n$ and $n \times p$ matrices $X = [x_{ij}]$, $Y = [y_{jk}]$, then compute XY in the following order. First evaluate the linear forms in the x-variables and in the y-variables,

$$L_q = \sum_{i,j} f(i, j, q)x_{ij}, \qquad L'_q = \sum_{j,k} f'(j, k, q)y_{jk}, \tag{2.2}$$

then evaluate the products $P_q = L_q L_q'$ for $q = 0, 1, \cdots, M - 1$, and finally evaluate the entries $\sum_j x_{ij}y_{jk}$ of XY as the linear combinations

$$\sum_j x_{ij}y_{jk} = \sum_{q=0}^{M-1} f''(k, i, q)L_q L_q'. \tag{2.3}$$

Here f, f', f'' are constants such that (2.2), (2.3) are identities in the indeterminates x_{ij}, y_{jk} for all $i = 0, 1, \cdots, m - 1; j = 0, 1, \cdots, n - 1; k = 0, 1, \cdots, p - 1$. M, the total number of all multiplications of L_q by L'_q, is called the *rank of the algorithm* and the multiplications of L_q by L'_q are called *the main or equivalently bilinear steps of the algorithm*. The minimum rank of bilinear algorithms for $m \times n$ by $n \times p$ MM is called the *rank of the problem of* $m \times n$ *by* $n \times p$ MM and is designated $\rho(m, n, p)$. We will also *designate the problem* itself *by the triplet* (m, n, p).

Remark 2.1. The straightforward algorithm for $m \times n$ by $n \times p$ MM is bilinear and has rank mnp. Strassen's algorithm for 2×2 MM is bilinear and has rank 7. In both straightforward and Strassen's algorithms the constants f, f', f'' take only the values 0, 1, and (for Strassen's algorithm) -1. The reader may assume for simplicity that the constants should always be real or complex numbers but almost all our presentation can be applied to the case where the constants are required to be chosen from an arbitrary given field or even ring with unity, see [18]. Actually we will refer to such a possibility in Remarks 8.1 and 8.2 in §8.

In this paper we will consider bilinear algorithms for MM but actually bilinear algorithms are useful for a more general class of problems, whenever a set of bilinear forms must be evaluated (see [1], [4], [9], [26]). Here is a simple purely illustrative example.

Example 2.1. *Evaluation of the product of two complex numbers,* $(x_0 + ix_1)(y_0 + iy_1) = (x_0y_0 - x_1y_1) + i(x_0y_1 + x_1y_0)$, $i^2 = -1$. In this case we need to evaluate $x_0y_0 - x_1y_1$, $x_0y_1 + x_1y_0$. The straightforward algorithm is a bilinear one where $L_0L'_0 = x_0y_0$, $L_1L'_1 = x_1y_1$, $L_2L'_2 = x_0y_1$, $L_3L'_3 = x_1y_0$, $M = 4$. Here is a simple bilinear algorithm for the same problem where $M = 3$ and where the constants f, f', f'' are also 0, 1 and -1. $L_0L'_0 = x_0y_0$, $L_1L'_1 = x_1y_1$, $L_2L'_2 = (x_0 + x_1)(y_0 + y_1)$, $x_0y_0 - x_1y_1 = L_0L'_0 - L_1L'_1$, $x_0y_1 + x_1y_0 = L_2L'_2 - L_0L'_0 - L_1L'_1$.

The importance of bilinear algorithms for MM stems from the following theorem.

THEOREM 2.1. *Given a bilinear algorithm* (2.2), (2.3) *for* (m, n, p), *such that* $mnp > 1$, *then*

$$\omega \leqq 3 \log M / \log (mnp). \tag{2.4}$$

Next we will give a proof assuming that $m = n = p$. (The extension to the general case will be easily obtained in §7.) We will proceed as in the particular case $n = 2$, $M = 7$

of the previous section. Recursively for $h = 1, 2, \cdots$ substitute $n^h \times n^h$ matrices for all indeterminates x_{ij}, y_{jk} of the original algorithm. Note that L_q, L'_q, $L_qL'_q$ and all linear combinations of the x-variables, of the y-variables, and of $L_qL'_q$ in (2.2), (2.3) also become $n^h \times n^h$ matrices while X, Y and XY become $n^{h+1} \times n^{h+1}$ matrices. For such a new algorithm perform all steps $L_qL'_q$ using the previously defined algorithm for $n^h \times n^h$ MM. Perform other matrix operations (additions and subtractions of matrices and their multiplications by the constants f, f', f'') by the relatively inexpensive straightforward methods which involve only n^{2h} arithmetics over the entries per such a matrix operation. Count the arithmetical operations and derive (2.1). Note that the exponent *depends only on the number of the main (bilinear) steps of the algorithm* (2.2), (2.3) *or equivalently on the number of bilinear terms (products)* $L_qL'_q$ *in* (2.3) and that the *number of multiplications by the constants* f, f', f'' *involved in the basic bilinear algorithm does not influence the exponent* because again it is easier to multiply an $N \times N$ matrix by a constant than by another $N \times N$ matrix.

Now in the vast variety of all possible bilinear algorithms (2.2), (2.3) for all m, n, p, M we need to find one such that

$$3 \log M/\log (mnp) < \log_2 7. \tag{2.5}$$

3. The search for a basic algorithm and the history of the asymptotic acceleration of MM. The desired basic algorithms that would satisfy (2.5) seemed to be quite carefully hidden from the researchers for about 10 years after Strassen's discovery in 1968. At first it was proven that $M = 6$ is not possible for 2×2 MM (see [7], [25]). Then efforts were concentrated on the reduction of the rank M of bilinear algorithms for 3×3 MM. The desired value was $M = 21$ because $\log 21/\log 3 = 2.771 \cdots$ while $\log 22/\log 3 = 2.813 \cdots$ (greater than $\log_2 7$). However so far only $M = 23$ has been achieved (see [10]).

Progress resumed in 1978 with the design where $m = n = p = 70$, $M = 143640$ which yielded $\omega \leqq \log 143640/\log 70 = 2.795 \cdots$. Thereafter several new improvements followed, as is indicated in Fig. 1.

Actually the numbers $n = 70$, $M = 143640$ (see above) came from a *more general construction* where bilinear algorithms for $n \times n$ MM were *first defined for all even* n such that $M = M(n) = (n^3 - 4n)/3 + 6n^2$. Then it remained to choose the value of n that minimizes $\log ((n^3 - 4n)/3 + 6n^2)/\log n$ and this turns out to be $n = 70$. Note that our formula would imply only the trivial exponent if we choose $n \leqq 8$.

The latter construction and ultimately all so far known asymptotically fastest

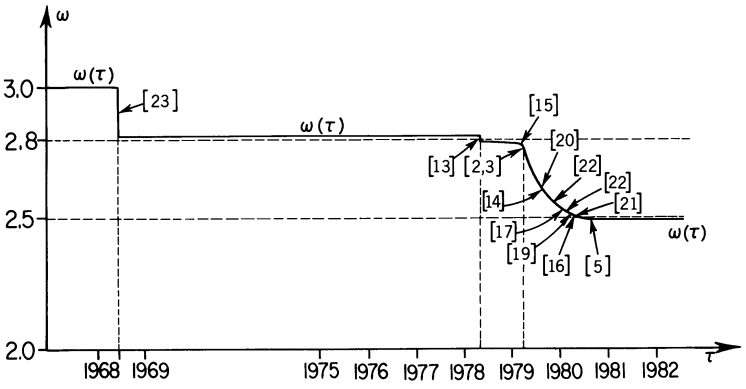

FIG. 1. $\omega(t)$ *is the best exponent announced by time* τ.

algorithms that imply $\omega < 2.5$ have originated in a simpler design of [12] where bilinear algorithms for $n \times n$ MM have been presented for all even n such that $M = 0.5n^3 + 3n^2$. That design remained practically unnoticed until 1978 because the application of (2.1) to the latter formula yields at best an exponent slightly below 2.85 (for $n = 34$). Nevertheless, we will be able to demonstrate all of the most important techniques for the asymptotic acceleration of MM by modifying and improving the latter design. We will present that design in a modified version as the bilinear algorithm (4.1), (4.2) below, although originally it appeared in the equivalent trilinear version and was applied to the evaluation of one rather than two matrix products.

4. The basic algorithm and the exponent 2.67. In this section we will outline how to reduce the exponent below 2.67. Our sketchy arguments will be justified in the next sections. We will start with the following bilinear algorithm for the simultaneous evaluation of two matrix products XY and UV.

(4.1)
$$\sum_j x_{ij}y_{jk} = \sum_j (x_{ij} + u_{jk})(y_{jk} + v_{ki}) - \sum_j u_{jk}y_{jk} - \sum_j (x_{ij} + u_{jk})v_{ki} \quad \text{for all } k, i.$$

(4.2)
$$\sum_k u_{jk}v_{ki} = \sum_k (x_{ij} + u_{jk})(y_{jk} + v_{ki}) - \sum_k u_{jk}y_{jk} - x_{ij}\sum_k (y_{jk} + v_{ki}) \quad \text{for all } i, j.$$

In this case we assume that for the four given matrices

$$X = [x_{ij}], \quad Y = [y_{jk}], \quad U = [u_{jk}], \quad V = [v_{ki}],$$
$$i = 0, \cdots, m-1; j = 0, \cdots, n-1; k = 0, \cdots, p-1,$$

of sizes $m \times n$, $n \times p$, $n \times p$, $p \times m$, respectively, we evaluate the mnp bilinear products $(x_{ij} + u_{jk})(y_{jk} + v_{ki})$ for all i, j, k; the mn products $x_{ij}\sum_k(y_{jk} + v_{ki})$ for all i, j; the np products $u_{jk}y_{jk}$ for all j, k, and the pm products $\sum_j(x_{ij} + u_{jk})v_{ki}$ for all k, i. This amounts to a total of

(4.3)
$$M_{m,n,p} = mnp + mn + np + pm$$

bilinear products of the form $L_qL'_q$, so that $M_{m,n,p}$ is the rank of the bilinear algorithm (4.1), (4.2). When all of the $M_{m,n,p}$ products $L_qL'_q$ have been found we can use (4.1), (4.2) in order to evaluate the matrix products XY and UV as linear combinations of those $L_qL'_q$ with the coefficients f'' equal to 0, 1 and -1.

We assume no relations among the entries of the four given matrices X, Y, U, V, so that we will consider those entries indeterminates and will call XY and UV two *disjoint matrix products.*

In the next sections we will generalize Theorem 2.1 and will see how a bilinear algorithm for the evaluation of disjoint matrix products also gives rise to a recursive construction and how this bounds the exponent. In particular the following bound will be deduced from the algorithm (4.1), (4.2),

(4.4)
$$\omega \leqq 3\log(M_{m,n,p}/2)/\log(mnp).$$

Comparing this with (2.4) we can see that the factor 2 appears because the algorithm (4.1), (4.2) simultaneously solves two equally hard problems (m, n, p) and (n, p, m). (In §9 we will show that these two problems have the same rank, $\rho(m, n, p) = \rho(n, p, m)$.)

In order to accentuate the power of the algorithm (4.1), (4.2), we will modify it further by introducing an auxiliary nonzero parameter λ as follows.

$$\sum_j x_{ij}y_{jk} = \sum_j (x_{ij} + u_{jk})(y_{jk} + \lambda v_{ki}) - \sum_j u_{jk}y_{jk} - \lambda \sum_j (x_{ij} + u_{jk})v_{ki}, \tag{4.5}$$

$$\sum_k u_{jk}v_{ki} = \lambda^{-1}\left\{\sum_k (x_{ij} + u_{jk})(y_{jk} + \lambda v_{ki}) - \sum_k u_{jk}y_{jk} - x_{ij}\sum_k (y_{jk} + \lambda v_{ki})\right\}. \tag{4.6}$$

For every value $\lambda \neq 0$ algorithm (4.5), (4.6) defines a bilinear algorithm of rank $M_{m,n,p}$ that evaluates XY and UV. On the other hand, we note that

$$\lim_{\lambda \to 0} \lambda \sum_j (x_{ij} + u_{jk})v_{ki} = 0.$$

Therefore if we delete the "vanishing" terms (products) $-\lambda\sum_j(x_{ij} + u_{jk})v_{ki}$ from (4.5) then we will have a bilinear algorithm of rank

$$M_{m,n,p,\lambda} = mnp + mn + np \tag{4.7}$$

that approximately evaluates XY and UV with a precision that can be made arbitrarily small by choosing λ small. This is an example of a *bilinear* λ*-algorithm* or equivalently of an *arbitrary precision approximation* (APA) *algorithm*. The class of APA-algorithms (λ-algorithms) for MM was first introduced in [3] where the following nontrivial λ-algorithm was presented.

Example 4.1.

$$p_1 = (\lambda x_{01} + x_{10})(\lambda y_{01} + y_{10}), \qquad p_2 = (\lambda x_{00} - x_{10})(y_{00} + \lambda y_{01}),$$

$$p_3 = (-\lambda x_{01} + x_{11})(y_{10} + \lambda y_{11}), \quad p_4 = x_{10}(y_{00} - y_{10}), \quad p_5 = (x_{10} + x_{11})y_{10},$$

$$q_{00} = x_{00}y_{00} + x_{01}y_{10} = \lambda^{-1}(p_1 + p_2 + p_4) - \lambda(x_{00} + x_{01})y_{01},$$

$$q_{10} = x_{10}y_{00} + x_{11}y_{10} = p_4 + p_5,$$

$$q_{11} = x_{10}y_{01} + x_{11}y_{11} = \lambda^{-1}(p_1 + p_3 - p_5) - \lambda x_{01}(y_{01} - y_{11}).$$

This is a λ-algorithm for Partial MM, because it evaluates 3 (out of 4) entries of the 2×2 matrix $Q = XY$ (with arbitrary precision as λ decreases). Ignoring the vanishing terms we reduce the rank to 5 while it is possible to prove that the rank of any conventional bilinear algorithm for the same problem is at least 6. Now consider the problem of the evaluation of the product Q^* of a 2×2 matrix X^* by a 2×3 matrix Y^*. Represent Q^* as the sum $Q^* = Q^0 + Q^1$ where Q^0, Q^1 take the forms

$$Q^0 = \begin{bmatrix} q^*_{00} & 0 & 0 \\ q^*_{10} & q^*_{11} & 0 \end{bmatrix}, \qquad Q^1 = \begin{bmatrix} 0 & q^*_{01} & q^*_{02} \\ 0 & 0 & q^*_{12} \end{bmatrix}.$$

Then evaluate Q^1 and the transpose of Q^0 by the above algorithm. This defines a bilinear λ-algorithm of rank 10 for Q^* and consequently defines the exponent $2.779\ldots$ It is not the best exponent that can be obtained from this design. It is possible to devise a clever and complicated recursive construction that starts directly from the design of rank 5 for Partial MM and finally derive the exponent $\omega \leqq 3\log_6 5 = 2.694\ldots$ (see [20], [22]).

Rather than delving into the latter approach we will generate a recursive construction starting from our λ-algorithm (4.5), (4.6) that defines the exponent

$$\omega \leqq 3\log(M_{m,n,p,\lambda}/2)/\log(mnp). \tag{4.8}$$

In order to obtain the best numerical value from (4.8), we substitute (4.7) into (4.8)

for $m = p = 7, n = 1$ and deduce that

$$\omega \leqq 3 \log 31.5/\log 49 = 2.669 \cdots . \tag{4.9}$$

In the sequel we will show that *the latter inequality bounds the number of arithmetical operations* not only in the best approximation algorithms for MM but *also in the conventional algorithms where the matrix products are evaluated exactly.*

The same design (4.5), (4.6) (if appropriately modified and recursively applied) implies the bound $\omega < 2.5$ (see [18]). However our next objective is to justify the bounds (4.8), (4.9) as a consequence of the identities (4.5), (4.6). This will also be an occasion to demonstrate some basic typically algebraic techniques customarily used for the design of efficient algorithms for algebraic computational problems.

5. Strassen's conjecture. The idea of the accumulation of the accelerating power of a basic algorithm by recursion. The exponential version of Strassen's conjecture. Our next objective is to justify the bound (4.4) as the first step towards (4.8). We could speculate that since the algorithm (4.1), (4.2) of rank $M_{m,n,p}$ computes both disjoint matrix products XY and UV then probably at least one of the two products, XY or UV, can be computed by a bilinear algorithm of a rank not greater than $M_{m,n,p}/2$. If such an algorithm existed we could apply Theorem 2.1 and derive (4.4). Generalizing that idea about the free transition from Disjoint MM to MM we come to the following famous *conjecture due to V. Strassen* (see [24]): given a bilinear algorithm of rank M for two disjoint problems (not even necessarily for problems of MM), then there exist two bilinear algorithms of ranks M_1 and M_2 for each of those two problems such that $M = M_1 + M_2$. Applying Strassen's conjecture to the algorithm (4.1), (4.2) and then applying Theorem 2.1 we immediately deduce (4.4).

It "remains" to prove the conjecture. However at this point our progress stops. To see that the conjecture is not easy to prove, consider the particular case of the algorithm (4.1), (4.2) where $m = n = p = 10$. Then the algorithm has rank 1300 and computes two disjoint 10×10 matrix products. Therefore the conjecture would imply that $\rho(10, 10, 10) \leqq 650$ while it is still unknown even if an algorithm of rank 700 for 10×10 MM exists.

Furthermore if we allowed the exclusion of the terms $\lambda \sum_j (x_{ij} + u_{jk}) v_{ki}$ that vanish as λ vanishes then the bilinear λ-algorithm (4.6), (4.7) would be a formal counterexample to such an extension of Strassen's conjecture (which was stated in [24] for conventional algorithms, not for λ-algorithms). Indeed, let $m = p$, $n = 1$. Then (4.6), (4.7) define a bilinear λ-algorithm of rank $m(m + 2)$ that computes both $m \times 1$ by $1 \times m$ and $1 \times m$ by $m \times m$ disjoint matrix products XY and UV with arbitrary precision. It is easy to show however that at least m^2 main steps or equivalently at least m^2 linearly independent terms $L_q L'_q$ are required in each bilinear algorithm that computes one of the two given matrix products with arbitrary precision. Since $2m^2 > m(m + 2)$ for $m > 2$, this disproves the extension of Strassen's conjecture to the class of bilinear λ-algorithms.

Thus we should look for a reduction of Disjoint MM to MM without using Strassen's conjecture. Of course, we may ignore all computed matrix products but one (say, use only the subalgorithm (4.1) and compute only XY). However such a *trivial reduction of Disjoint* MM *to* MM may cost a substantial part of the accelerating power of the original algorithm. (For instance, the rank of the subalgorithm for XY, (4.1) is $mnp + np + pm$ which is too large for $m \times n$ by $n \times p$ MM.) *In order to make use of our algorithm* (4.1), (4.2), *we will first accumulate its accelerating power by applying the algorithm recursively. After several recursive steps the relative cost of the trivial reduction to* MM *will become almost negligible and practically will not influence the exponent.* To see how

this works out, consider the case $m = n = p$. Using the trivial reduction to MM we derive (see Theorem 2.1) that $\omega \leqq \log M_{n,n,n}/\log n$ while Strassen's conjecture implies that there exists an algorithm of rank at most $Q = M_{n,n,n}/2$ (note that $M_{n,n,n} = n^3 + 3n^2$ is even, Q is integer) for the problem of $n \times n$ MM and therefore

$$\omega \leqq \log Q/\log n \quad \text{(cf. (4.4))}. \tag{5.1}$$

Let us deduce (5.1) without using Strassen's conjecture. The key step will consist of devising a recursive algorithm of rank $2Q^h$ for two disjoint problems of $n^h \times n^h$ MM for $h = 2, 3, \cdots$ (see (6.6) in §6). Of course, we can trivially reduce these algorithms to $h^h \times n^h$ MM. Applying Theorem 2.1 we obtain the bound

$$\omega \leqq \log(2Q^h)/\log(n^h)$$

which implies (5.1) as $h \longrightarrow \infty$. We will see that this argument can be extended to the case of arbitrary m, n, p in order to prove (4.4).

Note that if instead of the accumulation of the accelerating power of a basis algorithm by recursion (hereafter referred to as AAPR) we could apply Strassen's conjecture, then at least one of the two problems (m, n, p) or (n, p, m) must have an integer rank not greater than $M_{m,n,p}/2$. Then application of Theorem 2.1 would yield the same exponent (4.4) for even $M_{m,n,p}$ but the smaller exponent $\omega \leqq 3 \log((M_{m,n,p} - 1)/2)/\log(mnp)$ for odd $M_{m,n,p}$.

The informal rule is that the application of AAPR leads to the same exponent as the application of Strassen's conjecture would do if we "forgot" that the ranks must be integers. The formal (but equivalent!) version of this rule takes the form of the following theorem which will be called the *weak* or *exponential version* or *Strassen's conjecture.*

THEOREM 5.1 ([22]). *Given a bilinear algorithm of rank M for S disjoint problems of* MM, $(m(s), n(s), p(s))$, $s = 0, 1, \cdots, S-1$, *where* $m(0)n(0)p(0) > 1$, *then*

$$\omega \leqq 3\tau, \quad \sum_{s=0}^{S-1} (m(s)n(s)p(s))^{\tau} = M, \quad \tau \text{ is real.} \tag{5.2}$$

Let us show how to prove Theorem 5.1 using Strassen's conjecture. At first we will proceed independently of the conjecture. Just by using Theorem 2.1 we derive that

$$\omega \leqq 3\tau(s), \quad (m(s)n(s)p(s))^{\tau(s)} = \rho(m(s), n(s), p(s)), \quad s = 0, 1, \cdots, S-1.$$

Therefore

$$\sum_s (m(s)n(s)p(s))^{\omega/3} \leqq \sum_s \rho(m(s), n(s), p(s)). \tag{5.3}$$

If at this point we apply the conjecture then we obtain that

$$\sum_s \rho(m(s), n(s), p(s)) \leqq M. \tag{5.4}$$

(5.2) immediately follows from (5.3) and (5.4). □

In §§6 and 7 we will be able to prove Theorem 5.1 just by applying AAPR even without using the conjecture at all.

Actually a general idea of AAPR can successfully work for the asymptotic acceleration of MM whenever an original efficient algorithm deviates from the required regular format but can be used recursively and then reduced to the regular form. Particularly, in §8 using AAPR we will extend Theorem 5.1 to the case of λ-algorithms, that is we will prove the same bound (5.2) even if we start with a basis bilinear λ-algorithm of rank M

(we will say hereafter "*of* λ-*rank M*") rather than with a bilinear algorithm of rank M. Historically the idea and some techniques of AAPR were applied in order to utilize λ-algorithms for MM earlier than they were applied in order to utilize algorithms for Disjoint MM (compare [2] and [22]).

In the next three sections we will give formal definitions of the problem of Disjoint MM, of bilinear algorithms, and of bilinear λ-algorithms and will introduce some notation. We will also apply these definitions and notation in order to justify the bounds (4.4), (4.8), (5.2). Those readers who are most interested in the design of the efficient basic algorithms *may go directly to* §9 and if necessary consult with Theorems 5.1 and 8.4, Definitions 6.1, 6.2, 8.1, formula (8.3) and with the notation contained there. On the other hand, those readers who would like to see one of the two applications of AAPR may read either §§6 and 7 (in order to see the application to Disjoint MM) or §8 which is practically independent of §§6 and 7 (in order to see the application to λ-algorithms).

6. Recursive algorithms for MM and disjoint MM (definitions, notation and some basic facts). In this section we will define the format that we need in order to describe recursive algorithms for Disjoint MM. This will help us to proceed with AAPR starting with algorithms for Disjoint MM, such as (4.1), (4.2).

DEFINITION 6.1. *The problem of Disjoint* MM. Given S pairs of disjoint $m(s) \times n(s)$ and $n(s) \times p(s)$ matrices $X(s)$ and $Y(s)$ respectively, $s = 0, 1, \cdots, S-1$ (with no relations assumed among their entries). Then evaluate the S matrix products $X(s)Y(s)$, $s = 0, 1, \cdots, S-1$. This problem will be designated by

$$t = \bigoplus_s (m(s), n(s), p(s)).$$

In particular if $m(s) = m$, $n(s) = n$, $p(s) = p$ are invariant in s then we write $t = S \odot (m, n, p)$. For $S = 1$ we come back to the problem $(m, n, p) = 1 \odot (m, n, p)$ of $m \times n$ by $n \times p$ MM.

DEFINITION 6.2. *A bilinear algorithm of rank M for Disjoint* MM first evaluates M pairs L_q and L'_q for $q = 0, 1, \cdots, M-1$. Each L_q (each L'_q) is a linear combination of the entries of all matrices $X(s)$ (of all $Y(s)$ respectively). Then the algorithm evaluates M products $L_qL'_q$ for all q. Finally all of the entries of the matrix products $X(s)\,Y(s)$ for all s are evaluated as linear combinations of the products, $L_qL'_q$, $q = 0, 1, \cdots, M-1$. (Any bilinear algorithm for MM is a particular instance of this class where $S = 1$. (4.1), (4.2) is a particular instance of this class where $S = 2, X(0) = X, Y(0) = Y, X(1) = U, Y(1) = V, m(0) = m, n(0) = n, p(0) = p, m(1) = n, n(1) = p, p(1) = m, M = mnp + mn + np + pm$.) We will designate a bilinear algorithm of rank M for Disjoint MM by

$$\bigoplus_s (m(s), n(s), p(s)) \leftarrow M \odot (1, 1, 1). \tag{6.1}$$

The notation indicates that it suffices to perform the M main (bilinear) steps, that is the M multiplications of L_q by L'_q (each considered as an 1×1 MM) in order to evaluate the S matrix products $X(s)\,Y(s)$, provided that the linear steps of the evaluation of linear combinations of the x-variables, of the y-variables, and of the products $L_qL'_q$ *are considered free* (the latter assumption can be motivated by Theorem 2.1). In particular the algorithm (4.1), (4.2) is designated by

$$(m, n, p) \oplus (n, p, m) \leftarrow (mnp + mn + np + pm) \odot (1, 1, 1), \tag{6.2}$$

while the bilinear algorithm (2.2), (2.3) in the case where $m = n = p$ is represented by the following mapping:

$$(n, n, n) \leftarrow M \odot (1, 1, 1). \tag{6.3}$$

If we substitute $n \times n$ matrices for the input variables then we should change the sizes of the matrices on the left and on the right sides of the mapping (6.1), so that (6.1) takes the following form:

$$\oplus\ (m(s)n, n(s)n, p(s)n) \leftarrow M \odot (n, n, n).$$

This mapping will be considered as the product of the algorithm (6.1) and of the following trivial algorithm (mapping):

$$(n, n, n) \leftarrow (n, n, n). \tag{6.4}$$

The algorithm (6.4) can be interpreted as the *trivial reduction* of the problem of $n \times n$ MM to itself.

In the particular case where the algorithm (6.3) is given, we multiply the mappings (6.3) by the trivial reduction mapping (6.4) and derive the algorithm that operates over $n \times n$ matrices,

$$(n^2, n^2, n^2) \leftarrow M \odot (n, n, n).$$

Perform each bilinear step of the latter algorithm by using the algorithm (6.3) and obtain the bilinear algorithm

$$(n^2, n^2, n^2) \leftarrow M^2 \odot (1, 1, 1),$$

which can be considered as the square of (6.3).

Similarly we can define all powers of the algorithm (6.3),

$$(n^h, n^h, n^h) \leftarrow M^h \odot (1, 1, 1), \qquad h = 1, 2, 3, \cdots.$$

Actually we have already used this construction in our proof of Theorem 2.1 in §2. Now we have just simplified the matter by not caring about the cost-free linear steps.

Similar construction can be defined if we start with an algorithm for Disjoint MM. Then again we may substitute matrices of appropriate sizes for the variables. For instance, we may start with the algorithm (6.2) in the case $m = n = p$,

$$2 \odot (n, n, n) \leftarrow M_{n,n,n} \odot (1, 1, 1) \tag{6.5}$$

and obtain the algorithm

$$2 \odot (n^2, n^2, n^2) \leftarrow M_{n,n,n} \odot (n, n, n).$$

We note that the values $Q = M_{n,n,n}/2 = (n^3 + 3n^2)/2$ are integers for all n, apply the former algorithm in order to perform *each of the Q pairs of bilinear steps* of the latter algorithm, and derive that

$$2 \odot (n^2, n^2, n^2) \leftarrow Q \odot (2 \odot (n, n, n)) \leftarrow Q\, M_{n,n,n} \odot (1, 1, 1).$$

This way, starting with the algorithm (6.5), we recursively derive the sequence of bilinear algorithms

$$2 \odot (n^h, n^h, n^h) \leftarrow 2\, Q^h \odot (1, 1, 1), \qquad h = 1, 2, \cdots. \tag{6.6}$$

More generally we may start with the algorithm (6.1) and substitute rectangular matrices of size $m(r) \times n(r)$ for the x-variables and of size $n(r) \times p(r)$ for the y-variables of that algorithm. We may repeat this for S values of r choosing $r = 0, 1, \cdots, S - 1$. Then the S resulting algorithms (mappings)

$$\bigoplus_s (m(r)m(s), n(r)n(s), p(r)p(s)) \leftarrow M \odot (m(r), n(r), p(r)), \quad r = 0, 1, \cdots, S-1,$$

can be combined into the new algorithm

$$\bigoplus_{r,s} (m(r)m(s), n(r)n(s), p(r)p(s)) \leftarrow M \odot \left(\bigoplus_{r=0}^{S-1} (m(r), n(r), p(r))\right).$$

Thus the solution of the problem of Disjoint MM on the left side has been reduced to the solution of M problems on the right side. It remains to apply the original algorithm (6.1) to all of those M latter problems and derive the algorithm

$$\bigoplus_{r,s} (m(r)m(s), n(r)n(s), p(r)p(s)) \leftarrow M^2 \odot (1, 1, 1),$$

which will be considered the square of the algorithm (6.1). Similarly if we are given the algorithm (6.1) and the algorithm

$$\bigoplus_{s^*} (m^*(s^*), n^*(s^*), p^*(s^*)) \leftarrow M^* \odot (1, 1, 1)$$

where s^* ranges from 0 to $S^* - 1$ then we can derive the following algorithm (mapping), which we consider the product of the latter algorithm times (6.1),

$$\bigoplus_{s^*,s} (m^*(s^*)m(s), n^*(s^*)n(s), p^*(s^*)p(s)) \leftarrow M^* M \odot (1, 1, 1).$$

Summarizing we obtain the following result.

PROPOSITION 6.1. *Any two bilinear mappings (algorithms)* $t^* \leftarrow t^{*\prime}$ and $t \leftarrow t'$ *define their product, that is the bilinear mapping (algorithm)*

$$t^* \otimes t \leftarrow t^{*\prime} \otimes t',$$

where the product of two problems of MM *and of Disjoint* MM *is defined by the following formulae.*

$$(m^*, n^*, p^*) \otimes (m, n, p) = (m^*m, n^*n, p^*p), \tag{6.7}$$

$$\begin{aligned}\{\bigoplus_{s^*,s} (m^*(s^*), n^*(s^*), p^*(s^*))\} \otimes \{\bigoplus_s (m(s), n(s), p(s))\} \\ = \bigoplus_{s^*,s} (m^*(s^*)\, m(s), n^*(s^*)n(s), p^*(s^*), p(s)).\end{aligned} \tag{6.8}$$

Remark 6.1. Of course, the definition of the products is immediately extended to the powers $t^{\otimes 1} = t$, $t^{\otimes(h+1)} = t^{\otimes h} \otimes t$, $h = 1, 2, \cdots$. As follows from (6.7), (6.8), each of the three coordinates of the power of the problem $t = \oplus_s (m(s), n(s), p(s))$ can be expanded using the familiar formulae for the expansion of the powers of the linear forms $\sum_s m(s)x_s$, $\sum_s n(s)y_s$, $\sum_s p(s)z_s$. In particular, we have

$$(S \odot (m, n, p))^{\otimes h} = S^h \odot (m^h, n^h, p^h), \tag{6.9}$$

$$((m, n, p) \oplus (n, p, m))^{\otimes h} = \bigoplus_{g=0}^{h} \binom{h}{g} \odot (m^g n^{h-g}, n^g p^{h-g}, p^g m^{h-g}), \tag{6.10}$$

where $\binom{h}{g}$ are the binomial coefficients $h!/(g!(h-g)!)$, $g = 0, 1, \cdots, h$, $h = 1, 2, \cdots$.

We have derived Proposition 6.1 for the class of bilinear algorithms (mappings) defined by Definition 6.2 and for the trivial reduction algorithms of the form $(m, n, p) \leftarrow (m, n, p)$. These are the two cases that we will need to consider in our

recursive constructions. At the final step of AAPR (see the previous section and §§5 and 7) we will use *trivial reduction algorithms* (*mappings*) of the form

$$t \leftarrow t \oplus t^* \tag{6.11}$$

where t and t^* may represent the problems of MM and/or Disjoint MM. ((6.11) means that if t and t^* are solved then t is solved.) For the algorithms (mappings) defined by Definition 6.2 and by trivial reductions we obviously have the following *rule of composition* or of the transitivity of bilinear algorithms.

PROPOSITION 6.2. *Every two bilinear algorithms of the form* $t \leftarrow t'$ *and* $t' \leftarrow t''$ *define the new bilinear algorithm* $t \leftarrow t''$ (*called their composition*).

Propositions 6.1 and 6.2 enable us to represent the construction of recursive algorithms as a sequence of multiplications and compositions of mappings. (To see an illustration, recall how we earlier derived recursive algorithms from the algorithm (6.2) for $m = n = p$ (see (6.6)) and from the algorithm (6.3) and also compare the recursive algorithms in the next section).

Remark 6.2. As we will see in §9, each problem of MM or Disjoint MM can be naturally associated with a trilinear form and with the tensors of its coefficients. Then the products, the powers and the mappings of the problems will be associated with the products, the powers and the mappings of the tensors. This implies a simple extension of some results of our study of MM to the case of more general bilinear problems and bilinear algorithms which also can be defined by the associated tensors and by their mappings. We will not need such a generality in this paper and we do not assume any familiarity of the reader with the concept of tensors. However, for the sake of completeness, we will present the following tensor-free extension of our previous definitions. (We are not going to use this definition in the present paper.)

DEFINITION 6.3. Any set of bilinear forms in the x-variables and in the y-variables is called a *bilinear problem* (MM, Disjoint MM and Examples 2.1, 4.1 represent some particular instances of bilinear problems). Let a combination of three linear transformations of (i) the x-variables, (ii) the y-variables and (iii) the bilinear forms of a given bilinear problem t' define a new bilinear problem t. Then such a combination of the three linear transformations is called a *bilinear algorithm that reduces* t *to* t' *or equivalently a bilinear mapping that maps* t' *into* t and is designated by $t \leftarrow t'$.

7. Some applications of the recursive construction of bilinear algorithms. In this section we will apply our formalization of the recursive construction via the multiplications and the compositions of mappings (see Propositions 6.1 and 6.2). At first we will do that in order to complete the proof of Theorem 2.1 and then in order to justify the bound (4.4).

Proof of Theorem 2.1 for arbitrary m, n, p. In §9 we will show how to generate the two dual mappings, that is the two algorithms of rank M,

$$(n, p, m) \leftarrow M \odot (1, 1, 1), \qquad (p, m, n) \leftarrow M \odot (1, 1, 1)$$

whenever we are given the mapping $(m, n, p) \leftarrow M \odot (1, 1, 1)$, see (9.4). Multiplying all three of those mappings (see Proposition 6.1) we obtain the following algorithm for *square* MM,

$$(mnp, npm, pmn) \leftarrow M^3 \odot (1, 1, 1).$$

In the case of square MM we have already proven Theorem 2.1. Application of that theorem immediately yields (2.4) for all m, n, p.

PROPOSITION 7.1. *Let S divide M, so that $Q = M/S$ is an integer. Then the bilinear mapping (algorithm for Disjoint* MM)

$$S \odot (m, n, p) \leftarrow M \odot (1, 1, 1), \qquad mnp > 1 \tag{7.1}$$

defines the following bound,

$$\omega \leqq 3 \log (M/S)/\log (mnp). \tag{7.2}$$

Our *proof of Proposition* 7.1 will illustrate the idea of AAPR and the use of the products and the compositions of bilinear mappings. We will start with the algorithm (7.1) for Disjoint MM and after several recursive steps will reduce it to MM. We will proceed similarly to the derivation of the algorithm (mapping) (6.6) in the previous section.

At first multiply (7.1) by the trivial mapping $(m, n, p) \leftarrow (m, n, p)$, then compose the resulting mapping with (7.1) (or, informally speaking, substitute (7.1) on the right side) and derive the sequence of mappings

$$S \odot (m^2, n^2, p^2) \leftarrow Q\, S \odot (m, n, p) = Q \odot (S \odot (m, n, p))$$
$$\leftarrow Q\, M \odot (1, 1, 1) = Q^2 \odot (S \odot (1, 1, 1)).$$

Again multiply that sequence by the same trivial mapping, substitute (7.1) on the right side, and so on. After $h - 1$ steps obtain the following composition of mappings (which is still an algorithm for Disjoint MM but not for MM).

$$S \odot (m^h, n^h, p^h) \leftarrow Q^h\, S \odot (1, 1, 1).$$

Combine the latter mapping with the trivial reduction mapping

$$(m^h, n^h, p^h) \leftarrow S \odot (m^h, n^h, p^h).$$

Obtain the resulting mapping (which is already an algorithm for MM)

$$(m^h, n^h, p^h) \leftarrow Q^h\, S \odot (1, 1, 1)$$

for arbitrary h.

Apply Theorem 2.1 and derive that

$$\omega \leqq \omega(h) = 3 \log (Q^h S)/\log (mnp)^h$$
$$= 3 \log Q/\log (mnp) + 3 \log S/(h \log (mnp)) \quad \text{for all } h.$$

Hence,

$$\omega \leqq \lim_{h \to \infty} \omega(h) = 3 \log Q/\log (mnp).$$

This proves Proposition 7.1 because $Q = M/S$. □

Remark 7.1. We have just proven that $c(h)N^{\omega(h)}$ arithmetical operations suffice for $N \times N$ MM. Here $c(h)$ is a constant for a fixed h but $c(h)$ is unbounded for $h \to \infty$. Therefore $O(N^{\omega+\varepsilon})$ arithmetical operations suffice for $N \times N$ MM for arbitrary positive ε but not necessarily for $\varepsilon = 0$. Thus (7.2) defines a *truly limiting* exponent. We will see that the same is true for all exponents derived from λ-algorithms and from conventional bilinear algorithms for Disjoint MM that are not just algorithms for MM. Moreover, it is not very hard to prove that although the (least) exponent of MM is still unknown it *must be limiting* unless it is equal to 2 (in [5] it is shown that the exponent is either 2 or limiting in even a somewhat stronger sense).

To prove the bound (4.4), we will reduce it to Proposition 7.1. The next result will be our first step in that direction. Here again we will use the idea of AAPR and the recursive construction in the form of the products (in this case of the powers) and of the compositions (substitutions) of bilinear mappings. We will proceed similarly at the final step of the proof of (4.4).

PROPOSITION 7.2. (7.1) *implies* (7.2) *even if S does not divide M.*

Proof. Recursively applying Proposition 6.1 (see (6.9)), derive from (7.1) that

$$S^h \odot (m^h, n^h, p^h) \leftarrow M^h \odot (1, 1, 1), \qquad h = 1, 2, \cdots.$$

Then trivially reduce the left sides and obtain that

$$M^g \odot (m^h, n^h, p^h) \leftarrow M^h \odot (1, 1, 1)$$

where

$$M^g \leqq S^h < M^{g+1}, \qquad g \leqq h.$$

Apply Proposition 7.1 to the latter mappings and obtain that

$$\omega \leqq \omega(h) = 3 \log (M^h/M^g)/\log (mnp)^h = \log (M^h/S^h)/\log (mnp)^h + O(1/h)$$
$$= 3 \log (M/S)/\log (mnp) + O(1/h).$$

This implies (7.2) for $h \longrightarrow \infty$. □

To derive (4.4) in the general case, that is for all m, n, p, we need one more application of the idea of AAPR.

Recursively derive the powers of the algorithm (4.1), (4.2) and of the associated mapping (6.2),

$$((m, n, p) \oplus (n, p, m))^{\otimes h} \leftarrow M^h_{m,n,p} \odot (1, 1, 1), \qquad h = 1, 2, \cdots.$$

Expand the left sides (see (6.10)), apply trivial reductions, and deduce the following mappings, for $g = 0, 1, \cdots, h$ and $h = 1, 2, \cdots$,

$$\binom{h}{g} \odot (m^g n^{h-g}, n^g p^{h-g}, p^g m^{h-g}) \leftarrow M^h_{m,n,p} \odot (1, 1, 1).$$

By virtue of Proposition 7.2, it follows that

$$\omega \leqq 3 \log \left(M^h_{m,n,p}/\binom{h}{g}\right)/\log (mnp)^h = 3 \log \left(M_{m,n,p}/\binom{h}{g}^{1/h}\right)/\log (mnp).$$

Choose $h = 2g$. Then

$$\lim_{h \to \infty} \binom{h}{g}^{1/h} = 2.$$

Substitute this into the latter mapping and deduce (4.4) in the general case. □

We have derived the bound (4.4) starting with the basic algorithm (4.1), (4.2). If we start with the general algorithm (6.1) for Disjoint MM, we can similarly prove (5.2) without using Strassen's conjecture. Here is the *outline of the proof.* Expand the hth tensor power of the algorithm (6.1), then choose an appropriate trivial reduction to the mapping of the form (7.1), apply Proposition 7.2, and finally choose $h \longrightarrow \infty$. (Compare [18] for details.)

8. λ-algorithms and their applications to MM and disjoint MM. In order to prove (4.8), it remains to justify the deletion of the vanishing terms of the λ-algorithm (4.5),

(4.6). We will do that by purely algebraic methods which have nothing to do with the approximation by setting λ to be small. The final fast algorithms will be λ-free bilinear algorithms and will exactly evaluate the matrix products.

At first we will extend Definition 2.1 to the case of λ-algorithms.

DEFINITION 8.1. *A bilinear λ-algorithm of λ-rank M and of nonnegative degree d* for the evaluation of the product XY of two given matrices X, Y is defined by the following identities in the x-variables, in the y-variables, and in λ,

$$\sum_j x_{ij} y_{jk} = \lambda^{-d} \sum_{q=0}^{M-1} f''(k, i, q, \lambda) L_q L'_q + \lambda P \quad \text{for all } k, i. \tag{8.1}$$

Here

$$L_q = \sum_{i,j} f(i, j, q, \lambda) x_{ij}, \qquad L'_q = \sum_{j,k} f'(j, k, q, \lambda) y_{jk}, \qquad q = 0, 1, \cdots, M-1, \tag{8.2}$$

P is a polynomial in the x-variables, in the y-variables and in λ, and f, f', f'' are polynomials in λ with constant coefficients.

Similarly bilinear λ-algorithms for the evaluation of any set of bilinear functions and, in particular, for the evaluation of several disjoint matrix products $X(s)Y(s)$ (for Disjoint MM) are defined. In particular, the identities (4.5), (4.6) define a bilinear λ-algorithm of λ-rank $mnp + mn + np$ and of degree 1 for two disjoint matrix products. In Example 4.1 in §4 we encountered a bilinear λ-algorithm of λ-rank 5 and degree 1 for Partial MM which we immediately transformed into a bilinear λ-algorithm of λ-rank 10 and degree 1 for 2×2 by 2×3 MM.

We prefer to rewrite the identity (8.1) in the following (practically equivalent) form, which is obtained if we multiply all terms of (8.1) by λ^d and then reduce them modulo λ^{d+1}.

$$\lambda^d \sum_j x_{ij} y_{jk} = \sum_{q=0}^{M-1} f''(k, i, q, \lambda) L_q L'_q \quad \text{modulo } \lambda^{d+1}. \tag{8.3}$$

The multiplication by λ^d and the reduction modulo λ^{d+1} enabled us to get rid of the term λP in (8.3). In the case of the λ-algorithm (4.5), (4.6) this means that the terms $\lambda^2 \sum_j (x_{ij} + u_{jk}) v_{ki}$ indeed vanish after the reduction modulo λ^2. This is similarly true for the algorithm of Example 4.1.

Remark 8.1. Actually using (8.3) we interpret bilinear λ-algorithms for the evaluation of XY as conventional bilinear algorithms for the evaluation of $\lambda^d XY$ where the coefficients f, f', f'' are taken from the ring of polynomials in λ modulo λ^{d+1} rather than from the field of (complex or real) numbers.

If in (8.1)–(8.3) the degree d is 0 then the λ-algorithms turn into conventional bilinear algorithms. If $d > 0$ then we also can come back to bilinear algorithms but we need to increase their ranks. Indeed, start with the λ-algorithm (8.2), (8.3) and represent L_q, L'_q for all q as polynomials in λ modulo λ^{d+1}. Reduce each arithmetical operation over the polynomials in λ modulo λ^{d+1} to several operations over their coefficients. Each such coefficient is actually a λ-free linear form in the x-variables or in the y-variables. The linear operations over polynomials (that is their multiplications by constants, additions and subtractions) are reduced to similar linear operations over the coefficients. Each bilinear step of the λ-algorithm, that is each multiplication of the two polynomials in λ modulo λ^{d+1}, is reduced to at most $(d+1)^2$ multiplications of their coefficients, that is at most $(d+1)^2$ multiplications of λ-free linear forms in the x-variables by λ-free linear forms in the y-variables. This amounts to at most $(d+1)^2$ bilinear steps which all are

λ-free. Summarizing we obtain a conventional (λ-free) bilinear algorithm of rank at most $M(d+1)^2$ that evaluates all of the coefficients of the λ-polynomials $\sum_q f''(k,i,q,\lambda)L_qL'_q$ modulo λ^{d+1} for all k, i. In particular, the coefficients of λ^d of those polynomials, which are equal to the desired values of $\sum_j x_{ij}y_{jk}$, are evaluated for all k, i. Therefore we have obtained a bilinear algorithm of rank at most $M(d+1)^2$ for the given problem of MM. (Similarly for Disjoint MM.) We come to the following result.

PROPOSITION 8.1. *A bilinear λ-algorithm of degree d and λ-rank M for a problem of* MM *or Disjoint* MM *can be transformed into a conventional bilinear algorithm of rank $M(d+1)^2$ for the same problem.*

If we apply Proposition 8.1 and transform the λ-algorithm of degree 1 defined by (4.5), (4.6) into a conventional bilinear algorithm then the rank will increase $(d+1)^2 = 4$ times. Of course, this would be too big a sacrifice which would make the algorithm inefficient. Thus we will proceed with AAPR using the following modification of Proposition 6.1.

PROPOSITION 8.2. *A pair of bilinear λ-algorithms of λ-ranks M^* and M and of degrees d^* and d for the problems t^* and t respectively defines a bilinear λ-algorithm of λ-rank M^*M and degree $d^* + d$ for the problem $t^* \otimes t$.*

The *proof* is similar to the proof of Proposition 6.1. Just operate with polynomials modulo λ^{d^*+1}, λ^{d+1}, λ^{d^*+d+1} rather than with complex numbers while dealing with the coefficients f, f', f'' (see Remark 8.1 above) and also check that the power of λ on the left side of bilinear identities (compare (8.3)) becomes $d^* + d$ in the resulting algorithm. □

COROLLARY 8.1. *Recursive application of a bilinear λ-algorithm of λ-rank M and of degree d for a problem of* MM *or Disjoint* MM *defines a series of bilinear λ-algorithms of λ-ranks M^h and of degrees dh for the problems $(\oplus(m(s), n(s), p(s)))^{\otimes h}$ and consequently yields conventional bilinear algorithms of ranks $(dh+1)^2M^h$ for the same problems for all h, $h = 1, 2, \ldots$.*

We can easily verify that the factor $(dh+1)^2$ does not influence the exponents if we choose $h \to \infty$, that is each λ-algorithm of λ-rank M and of any degree for MM or Disjoint MM defines the same exponent as a conventional bilinear algorithm of the same rank M for the same problem does. More precisely, we may start with an arbitrary bilinear λ-algorithm for MM or Disjoint MM and applying Corollary 8.1 and Theorem 5.1 we obtain the following extension of Theorem 5.1 (which implies (4.8) and consequently (4.9) in the particular case where we start with the λ-algorithm (4.5), (4.6)).

THEOREM 8.1. (the exponential version of Strassen's conjecture for λ-algorithms). *Given a bilinear λ-algorithm of λ-rank M and of an arbitrary degree for the problem $\oplus_s(m(s), n(s), p(s))$ of* MM *or Disjoint* MM *where $m(0)n(0)p(0) > 1$, then*

$$\omega \leqq 3\tau, \quad \sum_s (m(s)n(s)p(s))^\tau = M, \quad \tau \text{ is real.}$$

Remark 8.2. Note that the exponential version of Strassen's conjecture for λ-algorithms holds true even though the conjecture itself has been formally disproved in the case of bilinear λ-algorithms (see §5). If we interpret the class of bilinear λ-algorithms as was suggested in Remark 8.1 then we may say that our example (4.5), (4.6) disproves Strassen's conjecture in the case of bilinear algorithms over the ring of polynomials in λ modulo λ^{d+1}.

9. Trilinear versions of bilinear algorithms and of bilinear λ-algorithms. So far we thoroughly exploited the power of our basic algorithm (4.1), (4.2) and of its λ-improvement (4.5), (4.6) using the idea of AAPR and developing appropriate techniques.

However, the basic algorithm itself has appeared ad hoc in §4. In the present and in the next sections we will describe a general approach to designing efficient basic algorithms. As the first step, we will define the equivalent trilinear versions of bilinear algorithms and of λ-algorithms in the form of some special decompositions of trilinear forms. For illustration we will start with the trilinear version of the rank 3 algorithm for the complex products (see our Example 2.1 in §2). We will introduce two auxiliary variables z_0, z_1, multiply the real part identity $x_0y_0 - x_1y_1 = x_0y_0 - x_1y_1$ by z_0, and the imaginary part identity $x_0y_1 + x_1y_0 = (x_0 + x_1)(y_0 + y_1) - x_0y_0 - x_1y_1$ by z_1 and sum those two products. The resulting trilinear identity

$$\begin{aligned} &x_0y_0z_0 - x_1y_1z_0 + x_0y_1z_1 + x_1y_0z_1 \\ &\quad = x_0y_0(z_0 - z_1) - x_1y_1(z_0 + z_1) + (x_0 + x_1)(y_0 + y_1)z_1 \end{aligned} \tag{9.1}$$

represents the original algorithm of rank 3 for the complex product. In order to go back to the bilinear identities, just equate the coefficients of each of z_0, z_1 on both sides of (9.1).

This is immediately extended to the following trilinear versions of an arbitrary bilinear algorithm (2.2), (2.3) for MM. (The extension to Disjoint MM is straightforward; see (9.5) below.)

$$\mathrm{Tr}(XYZ) = \sum_{i,j,k} x_{ij}y_{jk}z_{ki} = \sum_{q=0}^{M-1} L_qL'_qL''_q, \tag{9.2}$$

$$L''_q = \sum_{k,i} f''(k, i, q)\, z_{ki}. \tag{9.3}$$

Here Tr U is the trace of a matrix $U = [u_{qs}]$, $\mathrm{Tr}\ U = \sum_s u_{ss}$. $Z = [z_{ki}]$ is the matrix of auxiliary variables z_{ki}. The identity (9.2) is the sum of the identities (2.2) multiplied by z_{ki} for all k, i. For instance, Strassen's algorithm for 2×2 MM can be rewritten as the following decomposition of the class (9.2), (9.3),

$$\sum_{i,j,k=0}^{1} x_{ij}y_{jk}z_{ki} = \sum_{q=0}^{6} L_qL'_qL''_q.$$

Here

$$\begin{aligned} L_0L'_0L''_0 &= (x_{00} + x_{11})(y_{00} + y_{11})(z_{00} + z_{11}), \\ L_1L'_1L''_1 &= (x_{10} + x_{11})\, y_{00}\, (z_{01} - z_{11}), \\ L_2L'_2L''_2 &= x_{00}(y_{01} - y_{11})(z_{10} + z_{11}), \\ L_3L'_3L''_3 &= (-x_{00} + x_{10})(y_{00} + y_{01})z_{11}, \\ L_4L'_4L''_4 &= (x_{00} + x_{01})y_{11}(-z_{00} + z_{10}), \\ L_5L'_5L''_5 &= x_{11}(-y_{00} + y_{10})(z_{00} + z_{01}), \\ L_6L'_6L''_6 &= (x_{01} - x_{11})(y_{10} + y_{11})z_{00}. \end{aligned}$$

For the reverse transition from (9.2) to (2.2) consider the trilinear form as a linear form in z-variables and for all k, i equate the coefficients of z_{ki} on both sides of (9.2). If instead of that we equate the coefficients of x_{ij} or y_{jk} in (9.2) we obtain bilinear algorithms of the same rank M for the evaluation of YZ and ZX respectively.

This trick has interesting consequences in the case of rectangular MM. Whenever we have a bilinear algorithm for (m, n, p) we may first represent it as a trilinear identity and then derive two *dual algorithms* for (n, p, m) and (p, m, n) by equating the coefficients of

two other groups of variables. All three dual algorithms have the same rank. Consequently all three dual problems (m, n, p), (n, p, m), (p, m, n) have the same rank,

$$\rho(m, n, p) = \rho(n, p, m) = \rho(p, m, n). \tag{9.4}$$

The latter rank is also called the *rank of the trilinear form* $\mathrm{Tr}(XYZ)$ (and of *the tensor of its coefficients*) and is equal to the *minimum number of terms* in all possible decompositions (9.2), (9.3) for the given trilinear form $\mathrm{Tr}(XYZ)$.

Remark 9.1. Any bilinear algorithm for any bilinear problem can be equivalently represented as a trilinear identity. Then two dual algorithms can be derived. For instance, we have the trilinear identity (9.1) associated with the bilinear algorithm of Example 2.1 of §2. Equating the coefficients of x_0, x_1 on the both sides of (9.1) we derive one of the two dual algorithms associated with the original algorithm of Example 2.1,

$$L_0L_0' = y_0(z_0 - z_1), \quad L_1L_1' = -y_1(z_0 + z_1), \quad L_2L_2' = (y_0 + y_1)z_1,$$
$$y_0z_0 + y_1z_1 = L_0L_0' + L_2L_2', \qquad y_0z_1 - y_1z_0 = L_1L_1' + L_2L_2'.$$

Again, all three dual algorithms have the same rank (3 in this example). The duality technique enabled us to extend any successful bilinear algorithm to two new ones for two new problems, sometimes quite different from the original problem (see [9], [26]). Historically (9.4) was the first application of that idea (see [8], [12]).

Finally, here are trilinear representations of a general bilinear algorithm of rank M for the evaluation of disjoint matrix products $X(s)Y(s)$, $s = 0, 1, \cdots, S-1$ and of a general bilinear λ-algorithm of λ-rank M and degree d for the same problem.

$$\sum_{s=0}^{S-1} \mathrm{Tr}(X(s)\,Y(s)\,Z(s)) = \sum_{q=0}^{M-1} L_qL_q'L_q'', \tag{9.5}$$

$$\lambda^d \sum_{s=0}^{S-1} \mathrm{Tr}(X(s)\,Y(s)\,Z)(s)) = \sum_{q=0}^{M-1} L_qL_q'L_q'' \quad \text{modulo } \lambda^{d+1}. \tag{9.6}$$

In (9.5), (9.6) L_q, L_q', and L_q'' are linear forms in the x-variables, in the y-variables, and in the z-variables respectively whose coefficients are (complex or real) numbers in (9.5) and are polynomials in λ in (9.6).

In particular, the trilinear representations of the bilinear algorithm (4.1), (4.2) and of the bilinear λ-algorithm (4.5), (4.6) are as follows.

$$\begin{aligned}
\mathrm{Tr}(XYZ) + \mathrm{Tr}(UVW) &= \sum_{i,j,k} (x_{ij}y_{jk}z_{ki} + u_{jk}v_{ki}w_{ij}) \\
&= \sum_{i,j,k} (x_{ij} + u_{jk})(y_{jk} + v_{ki})(z_{ki} + w_{ij}) \\
&\quad - \sum_{i,j} x_{ij} \sum_{k} (y_{jk} + v_{ki})w_{ij} - \sum_{j,k} u_{jk}y_{jk} \sum_{i} (z_{ki} + w_{ij}) \\
&\quad - \sum_{k,i} \left(\sum_{j} (x_{ij} + u_{jk}) \right) v_{ki}z_{ki},
\end{aligned} \tag{9.7}$$

$$\begin{aligned}
\lambda(\mathrm{Tr}(XYZ) + \mathrm{Tr}(UVW)) &= \lambda \sum_{i,j,k} (x_{ij}y_{jk}z_{ki} + u_{jk}v_{ki}w_{ij}) \\
&= \sum_{i,j,k} (x_{ij} + u_{jk})(y_{jk} + \lambda v_{ki})(\lambda z_{ki} + w_{ij})
\end{aligned} \tag{9.8}$$

$$- \sum_{i,j} x_{ij} \sum_{k} (y_{jk} + \lambda v_{ki}) w_{ij}$$

$$- \sum_{j,k} u_{jk} y_{jk} \sum_{i} (\lambda z_{ki} + w_{ij}) \quad \text{modulo } \lambda^2.$$

10. Trilinear aggregating and the efficient basic designs. If we rewrite bilinear λ-algorithms as trilinear decompositions then the numbers of terms $L_q L'_q L''_q$ modulo λ^{d+1} on the right sides will equal the λ-ranks of those algorithms (similarly for the ranks of conventional bilinear algorithms). Therefore *our objective can be restated as the search for the trilinear decompositions with fewer terms.* We will reexamine decompositions (9.7), (9.8) from that point of view. We will classify all terms of (9.7) as follows. Each of the mnp terms $(x_{ij} + u_{jk})(y_{jk} + v_{ki})(z_{ki} + w_{ij})$ will be called the (trilinear) *aggregate* of the two *principal terms* $x_{ij}y_{jk}z_{ki}$ and $u_{jk}v_{ki}w_{ij}$. The trilinear aggregate can be considered a rank 1 "approximation" to the sum of two principal terms which has rank 2. The approach is somewhat similar to the well-known *linear* aggregation (compare [11]). *The key idea of our trilinear design is to correct the discrepancy between the sums of the mnp aggregates and of the 2mnp principal terms by using just a few (only mn + np + pm) correction terms.* Then the total number of terms will be reduced from the $2mnp$ principal terms to the $mnp + mn + np + pm$ terms on the right side of (9.7).

In order to compactly represent our design and conveniently analyze it, we will use Table 10.1.

AGGREGATING TABLE 10.1

x_{ij}	y_{jk}	z_{ki}
u_{jk}	v_{ki}	w_{ij}

If we multiply the 3 entries of the same row of Table 10.1 we will obtain a principal term. If we sum the entries along each of the 3 columns and multiply the 3 sums we will have an aggregate. All cross-products of the 3 entries in the 3 different columns which are not all in the same row are correction terms. Obviously this implies that the sum of all principal terms equals the difference between the sum of all aggregates and the sum of all correction terms. This holds if we sum the terms for fixed i, j and k and consequently if we sum then also in all i, j, k. Table 10.1 defines the algorithm (9.7) up to regrouping the correction terms. (For this particular table, the optimum regrouping is straightforward.)

A similar table is associated with the algorithm (9.8) and also defines that algorithm up to the straightforward regrouping of correction terms.

AGGREGATING TABLE 10.2

x_{ij}	y_{jk}	λz_{ki}
u_{jk}	λv_{ki}	w_{ij}

Again, the 2 products of the 3 entries of each row define 2 principal terms. Their aggregate is the product of the 3 sums, $x_{ij} + u_{jk}, y_{jk} + \lambda v_{ki}$ and $\lambda z_{ki} + w_{ij}$. For each triplet i, j, k the correction term $\lambda^2(x_{ij} + u_{jk})\, v_{ki}z_{ki}$ vanishes modulo λ^2. Thus comparing with the terms of Table 10.1 we can see the reduction of the number of terms.

We may hope to achieve further progress if we aggregate more than two principal terms because in that case each r-tuple of principal terms, $r > 2$, is reduced to one aggregate. On the other hand, correction terms become more numerous and require

greater care. However, using the cancellation modulo λ^{d+1} and some special canceling techniques we can get a further improvement along this line.

In particular, based on the *aggregation of triplets of principal terms we can reduce the exponent to* 2.516 $\cdots$. *If we aggregate more terms we can descend to slightly below* 2.5 (see the details in [5] or [18]). *However, in order to achieve a further substantial reduction (if such a reduction is possible), we probably would have to find a substantially different approach to the design of basic algorithms.*

11. Reduction of some other problems to MM. In this section we will show that our results on MM can be extended to other problems, in particular to *Boolean matrix multiplication* and *matrix inversion* (hereafter referred to as *Boolean* MM and MI respectively).

The Boolean product $C = [c_{ik}]$ of two $n \times n$ Boolean matrices $X = [x_{ij}]$ and $Y = [y_{ik}]$ with the entries 0 and 1 is defined by the usual formulae $c_{ik} = \sum_j x_{ij}y_{jk}$ but under the assumption that the arithmetical operations are performed in the closed semiring $\{0, 1\}$ using the following rules, $0 + 0 = 0$, $1 + 0 = 0 + 1 = 1 + 1 = 1$, $0*0 = 0*1 = 1*0 = 0$, $1*1 = 1$. In that case the evaluation of C is computationally equivalent to the evaluation of the transitive closure of a graph (see [1, pp. 202–204]). On the other hand, let $Q = [q_{ik}]$ be the usual matrix product of X and Y. Then, as is easily verified, we have for all i, k that either $c_{ik} = q_{ik} = 0$ or $c_{ik} = 1$, $1 \leqq q_{ik} \leqq n$. Therefore, the evaluation of C has been reduced to the evaluation of Q, which can be done by our asymptotically fast methods. The time-complexity of Boolean MM is measured by the number of bit-operations rather than the number of the arithmetical operations. However, evaluating Q we deal only with additions, subtractions and multiplications of integers which can be performed modulo $n + 1$. Therefore, the number of bit-operations involved is $t(n)$ times the number of arithmetical operations involved. Here the factor $t(n)$ equals the maximum number of bit-operations required in order to add, subtract or multiply two integers modulo $n + 1$. As is well known, $t(n) = O(\log n \log\log n \log\log\log n)$, see [1], [4], [9]. Of course, such a factor does not influence the exponent. Therefore, *the exponent of Boolean* MM *does not exceed the exponent of* MM. (Note that here we have actually applied our fast bilinear algorithms for MM over the constants from the ring of integers modulo $n + 1$ but our algorithms do work over that ring.) Presently the best asymptotic upper bounds on the time-complexity of Boolean MM and consequently of several combinatorial problems reducible to Boolean MM (including the graph transitive closure problem) are defined by the algorithms that reduce Boolean MM to MM. Note also that *all our asymptotically fast algorithms for both* MM *and Boolean* MM *require only* $O(N)$ *units of memory,* where N is the number of inputs, see [18].

On the other hand, *several linear algebra problems can be also reduced to* MM. *We will demonstrate the reduction of* MM *to* MI, *which immediately implies the similar reduction to* MM *of the solution of simultaneous linear equations and of the evaluation of the determinant of a matrix.*

In order to reduce MI to MM consider the following block-decompositions (see [23] or [1], [4], [9]) where X is an arbitrary $(2n) \times (2n)$ matrix to be inverted, $X_{11}, X_{12}, X_{21}, X_{22}$ are the $n \times n$ submatrices of X; O and I are the $n \times n$ null and identity matrices respectively.

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ X_{21}X_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} X_{11} & 0 \\ 0 & Z \end{bmatrix} \begin{bmatrix} I & X_{11}^{-1}X_{12} \\ 0 & I \end{bmatrix}, \tag{11.1}$$

$$X^{-1} = \begin{bmatrix} I & -X_{11}^{-1}X_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} X_{11}^{-1} & 0 \\ 0 & Z^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -X_{21}X_{11}^{-1} & I \end{bmatrix}, \tag{11.2}$$

where

$$Z = X_{22} - X_{21}X_{11}^{-1}X_{12}. \tag{11.3}$$

This implies the following recursive formula for the number $I(N)$ of arithmetics for $N \times N$ MI. $I(N) \leqq 2I(N/2) + 6M(N/2) + N^2/2$ for all even N where $M(N/2)$ is the number of arithmetics for $(N/2) \times (N/2)$ MM.

The latter recursive formula implies that $I(N) \leqq O(N^{\omega+\varepsilon})$ for all positive ε, so that the *exponent of* MI *does not exceed* ω, *the exponent of* MM.

Remark 11.1. The solution of a system $X\bar{u} = \bar{y}$ of linear equations in $\bar{u}$ can be reduced to MI because $\bar{u} = X^{-1}\bar{y}$. Also, using (11.1), (11.3) we derive that $\det X = \det X_{11} \det Z$. Hence $D(N) \leqq 2D(N/2) + 2M(N/2) + I(N/2) + N^2/4 + 1$ for all even N. Consequently $D(N) = O(N^{\omega+\varepsilon})$ for all positive ε. Here $D(N)$ is the number of arithmetics required for the evaluation of the determinant of an $N \times N$ matrix. Therefore the exponents of both latter problems (simultaneous equations and determinant) also do not exceed the exponent of MM. Actually the exponents of MM, of MI, and of the evaluation of the determinant of a matrix are known to coincide. Let us quickly show how MM can be reduced to MI. Apply the above formulae (11.2), (11.3) to the $(2n) \times (2n)$ matrix X where $X_{11} = I$. Note that in this case the lower right $n \times n$ submatrix of X^{-1} equals $Z^{-1} = (X_{22} - X_{21}X_{12})^{-1}$. Thus we have reduced the evaluation of the product of two arbitrary $n \times n$ matrices X_{21}, X_{12} to the inversion of the $(2n) \times (2n)$ matrix X and of the $n \times n$ matrix Z^{-1}, which is a submatrix of X^{-1}, and to the subtraction of the resulting $n \times n$ matrix Z from another (given) $n \times n$ matrix X_{22}. Therefore $M(n) \leqq I(2n) + I(n) + n^2$ for all n. □

Remark 11.2. Actually some additional care is required in order to assure that no divisions by 0 are ever implicitly introduced during our recursive transformation of the algorithm for MI via decompositions (11.2). That amounts to the requirement that, at the first recursive step, the matrices X_{11} and Z be invertible, and that similar conditions hold at all recursive steps. The latter properties are guaranteed if the given matrix X is a Hermitian positive definite matrix. Thus we will solve the problem if we invert X^HX and then evaluate $X^{-1} = (X^HX)^{-1}X$. Here X^H is the conjugate transpose of X, which has the complex conjugate of x_{ji} as its (i, j) entry. A slightly more complicated but even more efficient elimination of zero divisors can be obtained using permutations of the rows and/or columns of the given matrix X, see [1, pp. 232–242]. It is also not hard to handle the similar problem that might arise regarding our reduction of MM to MI.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman (1976), *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA.

[2] D. Bini (1980), *Relations between exact and approximate bilinear algorithms. Applications,* Calcolo, XVIII, pp. 87–97.

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

[4] A. Borodin and I. Munro (1975), *The Computational Complexity of Algebraic and Numeric Problems,* American Elsevier, New York.

[5] D. Coppersmith and S. Winograd (1981), *On the asymptotic complexity of matrix multiplication,* SIAM J. Comput., 11, pp. 472–492.

[6] M. J. Fisher and A. R. Meyer (1971), *Boolean matrix multiplication and transitive closure,*

Conference Record, 1971 IEEE 12th Annual Symposium on Switching and Automata Theory, pp. 129–131.

[7] J. E. HOPCROFT AND L. R. KERR (1969), *Some techniques for proving certain simple programs optimal,* Proc. 1969 IEEE Tenth Annual Symposium on Switching and Automata Theory, pp. 36–45.

[8] J. E. HOPCROFT AND J. MUSINSKI (1973), *Duality applied to the complexity of matrix multiplications and other bilinear forms,* SIAM J. Comput., 2, pp. 159–173.

[9] D. E. KNUTH (1981), *The Art of Computer Programming: Vol. 2, Semi-Numerical Algorithms,* Addison-Wesley, Reading, MA.

[10] J. D. LADERMAN (1976), *A noncommutative algorithm for multiplying* 3×3 *matrices using* 23 *multiplications,* Bull. AMS, 82, pp. 126–128.

[11] W. L. MIRANKER AND V. YA. PAN (1980), *Methods of aggregation,* Linear Algebra Appl., 29, pp. 231–257.

[12] V. YA. PAN (1972), *On schemes for the computation of products and inverses of matrices,* Uspekhi Mat. Nauk, 27(5), pp. 249–250. (In Russian.)

[13] ——— (1978), *Strassen algorithm is not optimal. Trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix multiplication,* Proc. 19th Annual Symposium on the Foundations of Computer Science, Ann Arbor, MI, pp. 166–176.

[14] ——— (1979), *Field extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplication,* Proc. 20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, pp. 28–38.

[15] ——— (1980), *New fast algorithms for matrix operations,* SIAM J. Comput., 9, pp. 321–342.

[16] ——— (1980), *Less than* 2.5161 *exponent for matrix multiplication,* Abstr. Amer. Math. Soc., 1, p. 384.

[17] ——— (1981), *New combinations of methods for the acceleration of matrix multiplication,* Compat. Math. Appl., 7(1), pp. 73–125.

[18] ——— (1982), *How can we mulply matrices faster?,* Technical Report 82-8 (revised), Computer Science Dept., State Univ. of New York, Albany (to appear as a monograph Springer-Verlag (1984)).

[19] F. ROMANI (1982), *Some properties of disjoint sums of tensors related to matrix multiplication,* SIAM J. Comput., 11, pp. 263–267.

[20] A. SCHÖNHAGE (1979), *Partial and total matrix multiplication,* TR(June 1979) Universität Tübingen.

[21] ——— (1980), (private communication).

[22] ——— (1981), *Partial and total matrix multiplication,* SIAM J. Comput., 10, pp. 434–456.

[23] V. STRASSEN (1969), *Gaussian elimination is not optimal,* Numer. Math., 13, pp. 354–356.

[24] ——— (1973), *Vermeidung von Division,* J. Reine Angew. Math., 264, pp. 184–202.

[25] S. WINOGRAD (1971), *On multiplication of* 2×2 *matrices,* Linear Alg. Appl., 4, pp. 381–388.

[26] ——— (1980), *Arithmetic Complexity of Computations,* CBMS Regional Conference Series on Applied Mathematics 33, Society for Industrial and Applied Mathematics, Philadelphia.