Improved Time and Space Bounds for Boolean Matrix Multiplication

Leonard Adleman
Kellogg S. Booth
Franco P. Preparata
Walter L. Ruzzo

July 1978

CS-78-21

Improved Time and Space Bounds for Boolean Matrix Multiplication

Leonard Adleman
Kellogg S. Booth
Franco P. Preparata
Walter L. Ruzzo

July 1978

Research Report CS-78-21

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Improved Time and Space Bounds for Boolean Matrix Multiplication

Leonard Adleman
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Kellogg S. Booth
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Franco P. Preparata
Coordinated Science Laboratory
Departments of Electrical Engineering and of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Walter L. Ruzzo
Department of Computer Science
University of Washington
Seattle, Washington 98195

Abstract. Using modular arithmetic we obtain the following improved bounds on the time and space complexities for $n \times n$ Boolean matrix multiplication: $O(n^{\log_2 7} \log n \,\mathrm{logloglog}\, n \,\mathrm{loglogloglog}\, n)$ bit operations and $O(n^2 \mathrm{loglog}\, n)$ bits of storage on a logarithmic cost RAM having no multiply or divide instruction; $O(n^{\log_2 7} (\log n)^{2 - \frac{1}{2}\log_2 7} (\mathrm{loglog}\, n)^{\frac{1}{2}\log_2 7 - 1})$ bit operations and $O(n^2 \log n)$ bits of storage on a RAM which can use indirect addressing for table lookups. The first algorithm can be realized as a Boolean circuit with $O(n^{\log_2 7} \log n \,\mathrm{logloglog}\, n \,\mathrm{loglogloglog}\, n)$ gates. Whenever $n \times n$ arithmetic matrix multiplication can be performed in less than $O(n^{\log_2 7})$ arithmetic operations, our results have corresponding improvements.

This work was supported in part by the Office of Naval Research under contract N00014-67-0204-0063, by the National Research Council of Canada under grant A4307, and by the National Science Foundation under grants MCS76-17321 and GJ-43332. A version of this report has been accepted for publication in *Acta Informatica*.

1. Introduction. Recall that Z_{n+1} is the ring of integers modulo $n+1$. Fischer and Meyer, Furman, and Munro [3,4,8] have all observed that $n \times n$ Boolean matrix multiplication can be implemented by embedding the $n \times n$ Boolean matrices, which do not form a ring, within the ring of $n \times n$ arithmetic matrices over Z_{n+1}, and then computing arithmetic products over Z_{n+1}, ultimately recovering the true Boolean matrix products by changing all nonzero entries to ones. Using Strassen's algorithm for arithmetic matrix multiplication [12] and the Schönhage-Strassen algorithm for integer multiplication over Z_{n+1} [11] we produce an asymptotically efficient algorithm.

Unless otherwise noted, all logarithms are to the base 2. The value of log 7 is approximately 2.81. If the model of computation is straight-line bitwise computation a time complexity of $O(n^{\log 7} \log n \operatorname{loglog} n \operatorname{logloglog} n)$ is achieved using the methods mentioned above [1]. An equivalent Boolean circuit of fan-in two can be constructed having the same number of gates. When the algorithm is implemented on a logarithmic cost RAM the time complexity is unchanged and the space complexity is $O(n^2 \log n)$ bits, if we assume that the RAM has the usual instruction set except for multiplication and division instructions.

Section 2 formalizes the models of computation with which we will be dealing and section 3 reviews some elementary number theory. In section 4 we show how to improve the two upper bounds stated earlier to $O(n^{\log 7} \log n \operatorname{logloglog} n \operatorname{loglogloglog} n)$ for the time complexity and $O(n^2 \operatorname{loglog} n)$ for the space complexity, using modular arithmetic. If we also use a table lookup scheme to compute integer products the time complexity drops to $O(n^{\log 7} \log n)$ and the space complexity stays the same. Using a more elaborate table-building technique attributed to Hopcroft in an exercise [1, problem 6.16], we further reduce the time complexity on a RAM to $O(n^{\log 7} (\log n)^{2 - \frac{1}{2}\log 7} (\operatorname{loglog} n)^{\frac{1}{2}\log 7 - 1})$ bit operations. In this last case the storage requirement increases back to the previous bound of $O(n^2 \log n)$ bits. Section 5 discusses the application of our techniques to the problem of arithmetic matrix multiplication.

A table lookup scheme is also central to the well-known Boolean matrix multiplication algorithm of Arlazarov, *et al.* [2]. The equivalent Boolean circuits for table lookup programs appear to require considerably more gates than the RAM time complexity, at least for their algorithm and for ours. For this reason we will draw a distinction between RAM programs which use indirect addressing for table lookup and those which do not.

Most of our results have corresponding improvements whenever the $O(n^{\log 7})$ upper bound for arithmetic matrix multiplication can be improved. We will thus state our results assuming that arithmetic matrix multiplication can be performed in $O(n^{2+\epsilon})$ arithmetic operations. The corresponding results using Strassen's upper bound will then be given as corollaries.

2. Models of Computation. We consider three models of computation: straight-line bitwise computation, RAM's having bitwise operations and restricted addressing, and the same RAM's with full addressing capability. These are essentially the models used by Aho, Hopcroft, and Ullman in their text [1]. *Straight-line bitwise computation* is the usual notion of straight-line code except that all variables are single bits and the only operations are the unary and binary Boolean connectives. This model is equivalent to Boolean circuits. The time complexity (number of operations) on a RAM is the same as the gate complexity of the corresponding circuit. We will not be concerned here with the depth of circuits, only with their number of gates.

Our RAM models use a logarithmic cost criterion and assume all of the bitwise Boolean, arithmetic, and shift instructions except for multiply and divide. These two operations are excluded because it is not known if they can be implemented to run in time linear in the length of their operands, a requirement if we want the logarithmic cost criterion to be a realistic assumption.

In the *RAM model* we allow unrestricted indirect addressing or indexing. This enables us to perform table lookup very easily. Our restricted RAM, which we call an *oblivious RAM* following Paterson, *et al.* [10], also has indirect addressing but is required to have its sequence of memory references a function only of the size of the input, not the actual values in the input. This assumption precludes the type of table lookup schemes employed later in this paper because the indirect addressing used to access the tables generates memory references which depend upon the input values.

Note that with an oblivious RAM we can always unroll loops to obtain either straight-line code or an equivalent Boolean circuit of the same gate complexity as the time complexity of the oblivious RAM. For a program using table lookup schemes (more generally, for a non-oblivious program) the number of gates in the equivalent Boolean circuit appears to be higher than the time complexity of the original program because we cannot unroll loops in the same way. Specifically, the algorithm of Arlazarov, *et al.* [2], which assumes a RAM model with full addressing capability, seems to require $O(n^4/\log n)$ gates when implemented directly in circuitry. We thus consider algorithms for Boolean matrix multiplication both on oblivious RAM's and on unrestricted RAM's so as to offer a fair comparison with existing algorithms.

The *reference machines* defined by Tarjan [13] are unable to perform generalized address calculations. However, our results for unrestricted RAM's carry over to reference machines if we assume a logarithmic cost criterion for all operations involving the reference machine's data registers. This is because reference machines can simulate logarithmic cost RAM's in real-time [13]. We will not consider reference machines separately here, but will simply note that the corresponding theorems are true.

3. Number-Theoretic Preliminaries. We recall a few facts from number theory. They are easily found in most elementary texts [5]. Throughout we will let p

stand for a prime number and let p_k stand for the k^{th} prime number. We recall two functions

$$\pi(x) = |\{p \leqslant x\}|$$

and

$$\theta(x) = \sum_{p \leqslant x} \ln p$$

which can be bounded on both sides, for suitable positive constants and sufficiently large x, as

$$a_1 \frac{x}{\log x} \leqslant \pi(x) \leqslant a_2 \frac{x}{\log x}$$

and

$$b_1 x \leqslant \theta(x) \leqslant b_2 x.$$

For our purposes the significance of these two inequalities is that n is appoximately equal to the product of the primes less than or equal to some fixed constant times $\log n$ and there are only about $\log n / \operatorname{loglog} n$ such primes. We state this more formally as the first lemma, which follows readily from the above facts.

Lemma 1: There exist constants c_1, c_2, and c_3 such that for all sufficiently large n there exists an m, depending only upon n, such that

$$m \leqslant c_1 \frac{\log n}{\operatorname{loglog} n},$$

$$\log(n+1) \leqslant \sum_{k=1}^{m} \log p_k \leqslant c_2 \log n,$$

and

$$p_k \leqslant c_3 \log(n+1), \quad 1 \leqslant k \leqslant m.$$

Proof:

If we choose $m = \pi\left(\frac{\ln(n+1)}{b_1}\right)$ then the three inequalities are easily verified if we let n be large enough. □

One consequence of lemma 1 is that we can immediately conclude, using the property that the logarithm of a product is the sum of the logarithms of its factors, that

$$n+1 \leqslant \prod_{k=1}^{m} p_k.$$

This means that we can invoke the Chinese Remainder Theorem to reduce a matrix multiplication over Z_{n+1} to a set of m matrix multiplications over the Z_{p_k}'s, each a ring of much smaller integers [1]. We state this as our second number-theoretic fact.

Lemma 2: Let $0 \leqslant x \leqslant n$. Then $x \equiv 0 \ (mod\ n+1)$ iff for all $1 \leqslant k \leqslant m$, $x \equiv 0 \ (mod\ p_k)$.

With these facts from number theory we are ready to state our main results. In the next section we present a series of algorithms which utilize these properties in order to achieve better upper bounds on the time and space complexities of Boolean matrix multiplication.

4. Algorithms. Given any algorithm for $n \times n$ matrix multiplication which runs in $M(n)$ arithmetic steps over an arbitrary ring, it is well-known that we can construct an algorithm for Boolean matrix multiplication by simply forming the arithmetic product of the two matrices of zeros and ones and then changing all nonzero entries in the product matrix to ones. Furthermore, it is sufficient to perform all of the calculations over Z_{n+1}, the ring of integers modulo $n+1$, because the entries in the arithmetic product are bounded above by $n+1$. The details of this construction are well known [1,3,4,8].

Let $M(n)$ be the number of arithmetic operations used in $n \times n$ matrix multiplication over a ring and let $B(n)$ be the number of bit operations used in $n \times n$ Boolean matrix multiplication. Performing all arithmetic over Z_{n+1} and using the Schönhage-Strassen integer multiplication algorithm [11] yields the well-known fact [1] that $B(n) = O(M(n) \log n \log\log n \log\log\log n)$. This is true in all of our models since the algorithm can be implemented on an oblivious RAM. It is easy to see that only $O(n^2 \log n)$ bits of storage need to be used by this method. We can do better than this, however, both in time and space.

Theorem 1: For an oblivious RAM (and hence for all of the other models) $B(n) = O(M(n) \log n \log\log\log n \log\log\log\log n)$ using only $O(n^2 \log\log n)$ bits of storage.

Proof:

It is easy to compute a list of the primes p_k for the first m primes using any number of techniques, such as the well-known sieve of Eratosthenes, in $o(n)$ bit operations [7]. The Boolean circuits will not, of course, perform such a computation, but will have the primes encoded within the circuit as a function of n. In any event, we assume that a list of the primes is available and we analyze only the remaining work performed by the algorithm.

If we perform each of the computations over Z_{p_k}, for $1 \leqslant k \leqslant m$, lemma 2 says that computing the inclusive OR of all of the bits in the (i,j)-entry of all of the m matrix products will give the correct value for the (i,j)-entry in the

Boolean product. Assuming again that the Schönhage-Strassen algorithm is used for computing the integer products in Z_{p_k}, for some constant c we can bound the number of bit operations by

$$B(n) \leqslant c \sum_{k=1}^{m} M(n) \log p_k \operatorname{loglog} p_k \operatorname{logloglog} p_k$$

since the time to obtain the inclusive OR of the various bits is dominated by the other costs and is negligible. Using the bounds given in lemma 1, we can conclude that

$$B(n) = O(M(n) \log n \operatorname{logloglog} n \operatorname{loglogloglog} n)$$

which is the desired upper bound for the time complexity.

The space complexity is also easy to analyze. On a RAM or straight-line model we can accumulate the partial result in an $n \times n$ matrix of single bits as the products over Z_{p_k} are computed, performing the inclusive OR as we go along. Thus we only need $O(\log p_k)$ bits per entry during the k^{th} computation and this storage can be reused for each k. The bound on p_k ensures that at most $O(n^2 \operatorname{loglog} n)$ bits of storage are necessary during the entire computation. □

Corollary 1: For oblivious RAM's, and hence for all of the other models

$$B(n) = O(n^{\log 7} \log n \operatorname{logloglog} n \operatorname{loglogloglog} n).$$

Proof:

The best known upper bound for $M(n)$ is provided by Strassen's algorithm for multiplying $n \times n$ matrices using only $O(n^{\log 7})$ arithmetic operations [12]. □

Corollary 2: There exist circuits for Boolean matrix multiplication which have only $O(n^{\log 7} \log n \operatorname{logloglog} n \operatorname{loglogloglog} n)$ gates.

We can reduce the time complexity of our algorithms even further if we allow our RAM's to perform table lookup. This permits us to generate multiplication tables from which we can obtain the product of two integers x and y in $O(\log x + \log y)$ operations under the logarithmic cost criterion.

Theorem 2: For unrestricted (non-oblivious) RAM's, $B(n) = O(M(n) \log n)$ using only $O(n^2 \operatorname{loglog} n)$ bits of storage.

Proof:

When we build a multiplication table for Z_{p_k} there are only p_k^2 ordered pairs of numbers which we may have to multiply. Our tables can be built using only additions since we are computing all possible products, and thus we require only $O(p_k^2 \log p_k)$ bit operations. The total time for constructing all of the tables is then $o(n)$, which can be ignored since $M(n)$ is clearly at least n^2.

Given the precomputed tables each multiplication requires $O(\log p_k)$ bit operations for products in Z_{p_k}, since we can simply lookup the product using the two factors as indices into the multiplication table. Thus

$$B(n) \leqslant \sum_{k=1}^{m} M(n) \log p_k,$$

and again using lemma 1 we obtain our result that $B(n) = O(M(n) \log n)$. Clearly the storage requirement is only increased by $o(n)$ over the previous algorithm, so the bound given in theorem 1 still applies. □

Corollary 3: for unrestricted RAM's $B(n) = O(n^{\log 7} \log n)$ using only $O(n^2 \log\log n)$ bits of storage.

In a supplement to his second volume [6], Knuth credits Schönhage and Strassen with a table lookup integer multiplication algorithm which is similar to the one proposed in the proof of theorem 2. Using this faster method would lead to an $O(M(n) \log n \log\log n)$ algorithm for Boolean matrix multiplication. Incorporating modular arithmetic would give a further improvement. Because of the table lookup, these algorithms are not oblivious.

Our final algorithm reduces the number of bit operations on a RAM even further than in theorem 2 at the expense of increasing the storage requirement back to $O(n^2 \log n)$ bits. The speedup may be viewed as modifying the basic Strassen matrix multiplication algorithm so that the recursion stops when the submatrices are sufficiently small, allowing the algorithm to finish up with an efficient table lookup scheme for multiplying the small submatrices. The cutoff size for the submatrices is a function of both n and p_k, so that the various calculations do not all have the same depth of recursion.

As before, we compute matrix products over the rings Z_{p_k}, for $1 \leqslant k \leqslant m$, but for each k we choose an appropriate way of viewing the problem. Thus for the k^{th} prime we consider each matrix to be an $(n/(\log_{p_k} n)^{1/2}) \times (n/(\log_{p_k} n)^{1/2})$ matrix whose elements are themselves $(\log_{p_k} n)^{1/2} \times (\log_{p_k} n)^{1/2}$ submatrices over the ring Z_{p_k}.

Applying Strassen's algorithm to the partitioned submatrices produces the same result as before. The trick is to find a fast way of multiplying the smaller matrices. We do this by a table lookup technique using a precomputed list of all possible products of such submatrices. This generalizes a technique attributed to Hopcroft [1, problem 6.16] for multiplying matrices over GF(2). Note that in our algorithm we have different matrix multiplication tables for each p_k.

Theorem 3: Let $M(n) = O(n^{2+\epsilon})$ for $\epsilon > 0$. For unrestricted RAM's $B(n) = O(n^{2+\epsilon}(\log n)^{1-\epsilon/2}(\log\log n)^{\epsilon/2})$ using only $O(n^2 \log n)$ bits of storage.

Proof:

We first verify that the time to precompute the multiplication tables is negligible. Note that there are only $p_k^{2\log_{p_k} n}$ distinct pairs of $(\log_{p_k} n)^{½} \times (\log_{p_k} n)^{½}$ matrices over Z_{p_k}. This is precisely n^2, independent of k, hence the time to build the tables is $o(n^{2+\epsilon})$, even if we use a crude algorithm to compute the products of submatrices (we assumed $\epsilon > 0$). The table-building cost will be dominated by the remaining work, and thus it can be safely ignored.

The actual number of bit operations is then given by

$$B(n) \leqslant \left[\sum_{k=1}^{m} M\left(\frac{n}{(\log_{p_k} n)^{½}} \right) \log_{p_k} n \log p_k \right] + O(n^2 \log n)$$

where the $\log_{p_k} n \log p_k$ factor is the cost of performing the table lookup for the submatrix products using indirect addressing or indexing to select the appropriate elements, each of which has that many bits. The low-order terms cover the computation of the Boolean product from the various arithmetic products over the rings Z_{p_k} when forming the inclusive OR of the bits.

Because $M(n)$ is at least n^2 we obtain the following, noting that $\log_{p_k} n \log p_k = \log n$:

$$B(n) = O\left(\sum_{k=1}^{m} \left(\frac{n}{(\log_{p_k} n)^{½}} \right)^{2+\epsilon} \log_{p_k} n \log p_k \right)$$

$$= O\left(n^{2+\epsilon} \log n \sum_{k=1}^{m} \left(\frac{\log p_k}{\log n} \right)^{1+\epsilon/2} \right)$$

$$= O\left(n^{2+\epsilon} (\log n)^{-\epsilon/2} \sum_{k=1}^{m} (\log p_k)^{1+\epsilon/2} \right).$$

We can transform the summation in the previous line as

$$\sum_{k=1}^{m} (\log p_k)^{1+\epsilon/2} = \sum_{k=1}^{m} \log p_k \ (\log p_k)^{\epsilon/2}$$
$$\leqslant d_1 (\log\log n)^{\epsilon/2} \sum_{k=1}^{m} \log p_k$$

$$\leqslant d_2 \log n (\log\log n)^{\epsilon/2}$$

by lemma 1, for suitable constants d_1 and d_2.

Thus $B(n) = O(n^{2+\epsilon}(\log n)^{1-\epsilon/2}(\log\log n)^{\epsilon/2}$ as claimed. The storage bound is $O(n^2 \log n)$ because each multiplication table requires that many bits and only one table is needed at a time. Every table has n^2 matrices with $\log_{p_k} n$ entries, each having $\log p_k$ bits. The remaining storage requirements are as in theorem 2. □

Corollary 4: $B(n) = O(n^{\log 7}(\log n)^{2-½\log 7}(\log\log n)^{½\log 7-1})$ using only $O(n^2 \log n)$ bits of storage on an unrestricted RAM.

5. Fast Arithmetic Products. The results in theorem 3 can be extended to the case in which we want to compute the arithmetic product of two $n \times n$ matrices having small entries or for which we are only interested in a modular product of two matrices, for small moduli.

Theorem 4: Let $M(n) = O(n^{2+\epsilon})$ for $\epsilon > 0$. For RAM's the arithmetic product of two $n \times n$ matrices over Z_{n+1} can be computed in $O(n^{2+\epsilon}(\log n)^{1-\epsilon/2}(\log\log n)^{\epsilon/2})$bit operations using only $O(n^2 \log n)$ bits of storage.

Proof:

The time bound follows from the proof of theorem 3 and the observation that a table of residues of the numbers in Z_{n+1}, modulo all of the primes p_k, can be built in $O(n \log n)$ bit operations. Each entry has a field of bits representing a number in Z_{n+1} followed by fields representing each of the number's residues modulo the primes $p_1, p_2, \ldots, p_m$. By lemma 1 the total length of an entry is $O(\log n)$ bits. The table is easily built in $O(n \log n)$ bit operations starting with the string of all zeros and successively adding ones in every field (modulo the appropriate p_k) to obtain the next entry.

We could sort the residues table using the residue fields as keys in $O(n \log n)$ bit operations with a bucket sort but we don't actually need a sort because the residues can be placed directly into the table using indexing. We then use the (i,j)-entries in the m modular matrix products as indices into the table to look up the corresponding (i,j)-entry of the matrix product over Z_{n+1}.

The storage bound follows immediately from theorem 3 and the fact that for each modulus p_k we produce and store a matrix of n^2 entries each with log p_k bits, for a total of

$$n^2 \sum_{k=1}^{m} \log p_k \leqslant c_2 n^2 \log n$$

bits of storage. The residues table is also of this size, so the total space requirement is only $O(n^2 \log n)$ bits of storage. □

Corollary 5: For RAM's the arithmetic product of two $n \times n$ matrices over Z_{n+1} can be computed in $O(n^{\log 7}(\log n)^{2 - \frac{1}{2}\log 7}(\log\log n)^{\frac{1}{2}\log 7 - 1})$ bit operations using only $O(n^2 \log n)$ bits of storage.

6. Concluding Remarks. We have shown improved time and space bounds for Boolean matrix multiplication using straight-line bitwise computation and logarithmic cost RAM's without multiplication or division. Our table lookup techniques can be used to provide an $O(n^{2+\epsilon}(\log\log n)^{\epsilon/2}/(\log n)^{\epsilon/2})$ upper bound for Boolean matrix multiplication on RAM's with the uniform cost criterion. This is $o(n^{2+\epsilon})$ but is superseded by the $O(n^2 \log n)$ bound implied by Weicker [14]. It should be pointed out that Weicker makes a somewhat unfair application of the uniform cost by using very large integers (order of $n \log n$ bits) whereas our algorithm uses much smaller integers (order of $\log n$ bits), a situation in which the uniform cost criterion seems more justifiable.

Recently the $O(n^{\log 7})$ upper bound for matrix multiplication over an arbitrary ring has been improved. Pan [9] has announced that the product of two 64×64 matrices can be computed using 111,872 multiplications. This leads to a Strassen-like algorithm for $n \times n$ matrix multiplcation with an asymptotic running time of $O(n^{\log_{64} 111{,}872})$ arithmetic operations, where the value of the exponent is approximately 2.795, improving upon the 2.81 of Strassen. This and any subsequent improvements will further reduce the bounds cited in corollaries 1-5. In fact, any improved upper bound for the complexity of multiplying matrices whose elements are themselves matrices of integers can be applied to our results.

The results for oblivious RAM's apply equally well to oblivious k-tape Turing machines for $k \geqslant 2$, because the algorithms can be implemented within the same time and space bounds on a multitape Turing machine, as pointed out by Knuth [6]. With a little additional care, they can be implemented obliviously on a 2-tape Turing machine.

Acknowledgement. We wish to thank both the referee, for a number of suggestions which have improved this paper, and Professor Michael Fischer, for pointing out some very useful references.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

[2] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, I. A. Faradžev, On economical construction of the transitive closure of an oriented graph, *Soviet Mathematics - Doklady* **11**:5 (1970), 1209-1210.

[3] M. J. Fischer and A. R. Meyer, Boolean matrix multiplication and transitive closure, *IEEE* 12th *Annual Symposium on Switching and Automata Theory* (1971), 129-131.

[4] M. E. Furman, Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph, *Soviet Mathematics - Doklady* **11**:5 (1970), 1252.

[5] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press (1956).

[6] D. E. Knuth, The Art of Computer Programming - *errata et addenda*, STAN-CS-71-194, Stanford University (1971), 25-26. The relevant information also appears in the later versions of Knuth's second volume (beginning with the second printing), *The Art of Computer Programming*, Vol. 2: *Seminumerical Algorithms*, Addison-Wesley (1969), 274-275.

[7] H. G. Mairson, Some new upper bounds on the generation of prime numbers, *Communications of the ACM* **20**:9 (September, 1977), 664-669.

[8] I. Munro, Efficient determination of the transitive closure of a directed graph, *Information Processing Letters* **1**:2 (1971), 56-58.

[9] V. Pan, Strassen's algorithm is not optimal, to appear in *IEEE* 19th *Annual Symposium on Foundations of Computer Science* (1978).

[10] M. S. Paterson, M. J. Fischer, and A. R. Meyer, An improved overlap argument for on-line multiplication, *SIAM-AMS Proceedings* **7** (1974), 97-111.

[11] A. Schönhage and V. Strassen, Schnelle Multiplikation grosser Zahlen, *Computing* **7** (1971), 281-292.

[12] V. Strassen, Gaussian elimination is not optimal, *Numerishe Mathematik* **13** (1969), 354-356.

[13] R. E. Tarjan, Reference machines require non-linear time to maintain disjoint sets, *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing* (1977), 18-29.

[14] R. Weicker, The influence of the machine model on the complexity of context-free language recognition, in *Lecture Notes in Computer Science*, Vol. 53, *Mathematical Foundations of Computer Science 1977*,

Springer-Verlag (1977), 560-569. An abstract for these results appears as R. Weicker, Context free language recognition by a RAM with uniform cost criterion in time $n^2 \log n$, in J. F. Traub (ed.) *Symposium on New Directions and Recent Results in Algorithms and Complexity*, Academic Press (1976).