BOUNDS ON THE SPEED-UP OF PARALLEL EVALUATION OF RECURRENCES

L. Hyafil and H. T. Kung

September, 1975

This research was supported in part by the National Science Foundation under Grant GJ32111 and the Office of Naval Research under Contract N00014-67-A-0314-0010, NR 044-422.

1. INTRODUCTION

To understand the performance of parallel computers such as ILLIAC IV and C.mmp, we must know the largest speed-up that can be obtained for a given task. If there are k processors, the largest speed-up that can be achieved is k and we call this optimal speed-up. The speed-ups in general depend on the parallel decomposition of a particular computing task and the various aspects of the multiprocessing system, including memory contention, process communication, operating system overhead, etc. In this paper, we concentrate on the issue of decomposing tasks, and assume that the multiprocessing system is idealized so that it causes no delays at all. We shall show that even under this idealized assumption, there are problems for which, because the parallel decompositions are inherently difficult, the optimal speed-up cannot be achieved.

This paper studies bounds on speed-ups for a particular problem, i.e., the problem of evaluating (or solving) recurrences, which is defined as follows:

Input: $x_0, x_{-1}, \ldots, x_{-p+1}$ and rational function r_i, $i \geq 1$.

Output: x_n, which is defined by $x_i = r_i(x_{i-1}, \ldots, x_{-p+1})$, $i \geq 1$.

Since the x_i are defined iteratively, the problem appears on the surface to be highly serial. Hence it is interesting to investigate how parallel algorithms can be designed and what are the theoretical limits of using parallelism for the problem. We consider the recurrence problem also because it is important in practice and is simply stated so that we might obtain some insight into the nature of parallel computation by studying it. We shall survey a number of results in connection with bounds on the speed-ups of parallel evaluation of various kinds of recurrences, especially when the size n of the problem is large, or when $n \to \infty$. For simplicity we assume that each arithmetic operation takes one unit of time. Consider a k-processor machine. We shall see, for example, that the speed-up for the first order linear recurrence problem is at most $(2/3)k + (1/3)$ even under the idealized assumption. Of course, the actual speed-up obtained from a real k-processor machine would be $\leq (2/3)k + (1/3)$. The difference between $(2/3)k + (1/3)$ and k is rather significant. For example, if $k = 16$, 64, then the speed-up for the problem is at most 11, 43, respectively, no matter how efficient the k-processor machine is. The reason that we get at most 70 percent of the speed-up we might expect for the problem is the inherent dependence of variables in the recurrence. Nonlinear recurrences are even worse. It is shown that the speed-ups for a certain class of nonlinear recurrence problems are always bounded by a constant no matter how many processors are used and how large the size of the problem is. Hence the dependency relationships within the variables of these nonlinear recurrences are even stronger. We believe that the study of these dependency relationships is fundamental for understanding parallel computation.

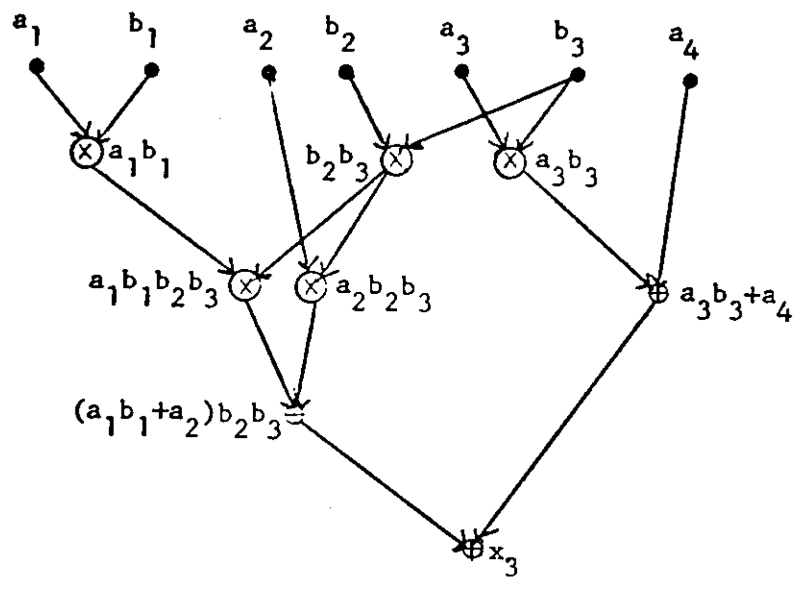
The kind of results which are to be presented in the paper could be useful in the following two ways. First, the theoretical bounds on speed-ups provide grounds for testing the efficiencies of algorithms and the multiprocessing system. (For example, it would be very helpful if tight theoretical bounds on speed-ups are known for benchmark tasks.) Second, the constructions of the algorithms designed for the idealized machine are instructive and often lead to useful insights into the nature of designing efficient algorithms for real machines.

2. DEFINITIONS AND NOTATION

An algorithm for evaluating x_n is defined to be a directed acyclic graph in a natural way. For example, the following graph defines a parallel algorithm using three processors for evaluating x_3, which is defined by

$$x_0 = a_1,$$
$$x_i = b_i x_{i-1} + a_{i+1}, \quad i = 1,2,3.$$

(Note that $x_3 = ((a_1b_1+a_2)b_2+a_3)b_3+a_4$.)



Consider the directed graph which defines an algorithm. We define the depth of the graph to be the <u>time</u>, define

$$T_k(x_n) = \text{minimum time needed to evaluate } x_n \text{ by an algorithm using k processors.}$$

and define the <u>speed-up</u> of the problem of evaluating x_n by using k processors to be

$$S_k(x_n) = \frac{T_1(x_n)}{T_k(x_n)} .$$

(In [9] these definitions are given in a more rigorous way.)

By a simple simulation argument, one can easily see that $T_1(x_n) \leq k\ T_k(x_n)$. Hence

$$S_k(x_n) \leq k, \quad \forall k, \forall n.$$

k is a trivial upper bound on $S_k(x_n)$. Bounds smaller than k are nontrivial. We shall show some nontrivial upper bounds on $S_k(x_n)$ in the following sections.

3. FIRST ORDER LINEAR RECURRENCES

A first order linear recurrence is defined by

(1) $x_i = a_i x_{i-1} + b_i,\ i \geq 1.$

It is the most fundamental recurrence, in the sense that algorithms for solving it often form basic algorithms for solving other types of recurrences. The trivial algorithm which computes $x_1, x_2, \ldots, x_n$ iteratively according to (1) is the optimal sequential algorithm, since it takes time 2n and any algorithm has to take time at least 2n for using all the inputs. Hence

(2) $T_1(x_n) = 2n.$

The algorithm, however, is not suitable for parallel computers because it does not provide any parallelism. New algorithms are needed for parallel computers. Various parallel algorithms have been developed by many people [3, 4, 12, 13, 14, 15, 17, 20, 21, 23]. The basic idea of these algorithms can be explained as follows:

Note that (1) is equivalent to

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ 1 \end{bmatrix}, \quad i \geq 1.$$

Hence

$$(3) \quad \begin{bmatrix} x_n \\ 1 \end{bmatrix} = \left(\prod_{1}^{n} \begin{bmatrix} a_i & b_i \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x_0 \\ 1 \end{bmatrix},$$

which can clearly be computed in parallel. Using the fact that the multiplication of two matrices of the form $\begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix}$ takes three operations and results in a matrix of the same form, while the multiplication of $\begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} x \\ 1 \end{bmatrix}$ uses two operations and results in a vector of the form $\begin{bmatrix} x \\ 1 \end{bmatrix}$, in [11] a parallel algorithm based on (3) is derived and establishes that

$$(4) \quad T_k(x_n) \leq \frac{3n}{k+1/2} + c_1 \log k$$

for some constant $c_1 > 0$. (4) is an improvement over the corresponding result in [23] when n is large and k is fixed.

In [9] it is shown that if an algorithm computes x_n in time t with w operations, then

$$(5) \quad w \geq 3n - \frac{t}{2}.$$

Suppose that $t < T_1(x_n) = 2n$. Then by (5) $w > 2n$. Hence if a parallel algorithm is faster than the optimal sequential algorithm, then it must perform more operations than the sequential algorithm. This turns out to be the basic reason why the optimal speed-up cannot be achieved for the problem. Indeed, lower bounds on $T_k(x_n)$ can be easily derived from (5) as follows. Suppose that k processors are used. Observe that for any algorithm, $kt \geq w$. Hence by (5) we have

(6) $T_k(x_n) \geq \frac{3n}{k+1/2}, \forall k, \forall n.$

Suppose that $w \geq 2^{\lceil \log k \rceil}$. (This is true when, say, $n \geq k$.) In [8] it is pointed out that in this case by the same argument as used in [18, Theorem 1], the bound in (6) can be slightly improved. In fact, we have

$$k(t-\lceil \log k \rceil) + 2^{\lceil \log k \rceil} - 1 \geq w,$$

which together with (5) yields

$$T_k(x_n) \geq \frac{3n}{k+1/2} + (k\lceil \log k \rceil + 1 - 2^{\lceil \log k \rceil})/(k+1/2).$$

Hence

(7) $T_k(x_n) \geq \frac{3n}{k+1/2} + c_2 \log k - c_3, \quad n \geq k$

for some constants $c_2 > 0$ and $c_3 > 0$. From (4) and (7), we know that the bounds are essentially sharp for $n \geq k$.

From (2), (4), (6) and (7) we have the following

Theorem 1

For the first order linear recurrence defined by (1),

$$\begin{array}{ll} (8) & \\ (9) & \frac{2k+1}{3 + \frac{c_1(k+1/2)\log k}{n}} \leq S_k(x_n) \leq \begin{cases} \frac{2}{3}k + \frac{1}{3}, & \forall k, \forall n \\ \frac{2k+1}{3 + \frac{(c_2 \log k - c_3)(k+1/2)}{n}}, & \forall k, \forall n \geq k. \end{cases} \end{array}$$

The upper bound in (8) implies that even for the simplest recurrence defined by (1), we can get at most 70 percent of the optimal speed-up.

The algorithm used to establish the bound in (4) can be extended to solve first order vector linear recurrences, defined by

(10) $\underline{x}_i = A_i\underline{x}_{i-1} + \underline{b}_i, \quad i \geq 1,$

where the $\underline{x}$'s and the $\underline{b}$'s are p-vectors and the A's $p \times p$ matrices. The upper bounds on time for solving these vector recurrences can similarly be obtained.

4. PTH ORDER LINEAR RECURRENCES

A pth order linear recurrence is defined by

(11) $$x_i = \sum_{j=i-p}^{i-1} a_{ij}x_j + b_i, \quad i \geq 1.$$

The problem for solving such a recurrence in parallel has been considered in [5, 12, 13].

The following theorem generalizes the upper bound result in (8).

Theorem 2 [11].

For the pth order linear recurrence defined by (11),

(12) $$S_k(x_n) \leq \frac{2p}{2p+1} k + c_4, \quad \forall p, \forall k, \forall n.$$

for some constant c_4.

Since $\frac{2p}{2p+1} < 1$, the theorem implies that we cannot essentially obtain the optimal speed-up ratio for solving pth order linear recurrences for any p, when k is large.

We now consider parallel algorithms for solving the recurrence defined by (11). The idea is to convert it into a first order vector linear recurrence of the form (10), which can then be solved by algorithms used in the preceding section.

The naive approach for the conversion would be the following way: Define vectors

$$\underline{x}_i = \begin{bmatrix} x_i \\ x_{i-1} \\ \vdots \\ x_{i-p+1} \end{bmatrix}, \ i \geq 0$$

then (11) is equivalent to

(13) $\underline{x}_i = A_i \underline{x}_{i-1} + \underline{b}_i, \quad i = 1,2,\ldots,n$

where the A_i are certain companion matrices. Then algorithms for solving first order vector linear recurrences can be applied to compute $\underline{x}_n$ (and hence x_n) from (13). We shall use another conversion technique, which will lead us to p times faster algorithms for the case that k, p are fixed and $n \to \infty$. The idea is explained in the following for the case of $p = 3$. We can write a 3rd order linear recurrence as

$$x_i = -\sum_{j=i-3}^{i-1} \bar{a}_{ij} x_j + b_i, \quad i \geq 1$$

where $\bar{a}_{ij} = -a_{ij}$. Then for computing, say, x_6 from x_0, x_{-1}, x_{-2} we have

$$\left[\begin{array}{ccc|ccc|ccc} \bar{a}_{1,-2} & \bar{a}_{1,-1} & \bar{a}_{10} & 1 & & & & & \\ & \bar{a}_{2,-1} & \bar{a}_{20} & \bar{a}_{21} & 1 & & & & \\ & & \bar{a}_{30} & \bar{a}_{31} & \bar{a}_{32} & 1 & & & \\ \hline & & & \bar{a}_{41} & \bar{a}_{42} & \bar{a}_{43} & 1 & & \\ & & & & \bar{a}_{52} & \bar{a}_{53} & \bar{a}_{54} & 1 & \\ & & & & & \bar{a}_{63} & \bar{a}_{64} & \bar{a}_{65} & 1 \end{array}\right] \left[\begin{array}{c} x_{-2} \\ x_{-1} \\ x_{0} \\ \hline x_1 \\ x_2 \\ x_3 \\ \hline x_4 \\ x_5 \\ x_6 \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ \hline b_4 \\ b_5 \\ b_6 \end{array}\right]$$

If we partition the matrix and vectors into blocks as indicated above, then we have

$$\begin{bmatrix} A_1 & T_1 & \bigcirc \\ \bigcirc & A_2 & T_2 \end{bmatrix} \begin{bmatrix} \underline{x}_0 \\ \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \end{bmatrix}$$

Hence

$$\underline{x}_1 = -T_1^{-1}A_1\underline{x}_0 + T_1^{-1}\underline{b}_1,$$
$$\underline{x}_2 = -T_2^{-1}A_2\underline{x}_1 + T_2^{-1}\underline{b}_2,$$

which is a first order vector linear recurrence. Using the same idea, for general p, we have

$$(14) \quad \underline{x}_i = (T_i^{-1}A_i\ )\underline{x}_{i-1} + T_i^{-1}\underline{b}_i, \ i = 1,2,\ldots,m$$

where $m = \lceil n/p \rceil$, T_i, A_i, $\underline{x}_i$, $\underline{b}_i$ are of size p, and T_i are triangular. We shall first compute $T_i^{-1}A_i$ and $T_i^{-1}\underline{b}_i$ for $i \le m$, and then use algorithms in Section 3 to solve the recurrence (14). Since $m = \lceil n/p \rceil$, the recurrence

(14) is shorter than the recurrence (13) by a factor of p. Thus we get faster algorithms. (It turns out that the cost of computing $T_i^{-1}A_i$ and $T_i^{-1}\underline{b}_i$ is not crucial.) From this approach it is immediate to prove that

$$(15) \quad T_k(x_n) \le c_5(\frac{p^\alpha}{k} n + p^\alpha \log n),$$

for some constant $c_5 > 0$, where $\alpha = 2$ when the usual matrix multiplication algorithm is used and $\alpha = 1.82$ when the Strassen's matrix multiplication algorithm [22] is used. (In [11] it is shown that the bound in (15) also holds for the problem of solving n×n <u>band</u> linear system with bandwidth p.) Since $T_1(x_n) \ge (p+1)n$, taking $\alpha = 1.82$ in (15) we have that for any k and p,

$$(16) \quad S_k(x_n) \ge \frac{1}{c_5}(\frac{k}{p^{.82}}), \quad \text{as } n \to \infty,$$

for the problem of solving pth order linear recurrences. Does S_k/k indeed decrease as p increases? The question is still open. We only know that by (12) S_k is always less than k for large k. We believe that as p increases, more dependency relationships on the x's defined by (11) will be introduced and hence S_k/k will decrease.

<u>Conjecture</u>

<u>Consider the problem of solving pth order linear recurrences defined by (11). Let the maximal speed-up ratio achievable by using k processors to be</u>

$$\underline{\bar{S}_k(p) = \max_n S_k(x_n).}$$

<u>Then there exists a monotonically decreasing function λ such that</u>

$$\underline{\bar{S}_k(p) \le \lambda(p)k, \quad \forall k,}$$

<u>and $\lambda(p) \to 0$ as $p \to \infty$</u>.

The following theorem relates our conjecture on speed-ups to the matrix multiplication problem.

<u>Theorem 3</u> [11]

<u>If the conjecture is true then $O(n^2/\lambda(n))$ is a lower bound on the number of arithmetic operations needed to multiply two $n \times n$ matrices</u>.

Note that the question of whether or not matrix multiplication can be done in $O(n^2)$ operations has been open for some years.

5. GENERAL LINEAR RECURRENCES

A general linear recurrence is defined by

$$(17) \quad x_i = \sum_{j=0}^{i-1} a_{ij}x_j + b_i, \qquad i \ge 1.$$

The problem of solving general linear recurrences is reducible to that of solving triangular linear systems. Heller [6] first considered the problem of solving (17) in parallel and gave algorithms which take time $O(\log^2 n)$ and use $O(n^4)$ processors. It was shown later that the problem in fact could be done in time $O(\log^2 n)$ with $O(n^3)$ processors by a number of people in at least three different ways (see, e.g., [2, 5, 7, 19].

For the case of using small parallelism it is shown in [10] that

(18) $T_k(x_n) \le \frac{n^2}{k} + c_6 n$ if $k \le n$,

$$T_k(x_n) \le \begin{cases} c_7 n^{2-r} \log n & \text{if } k = \lceil n^r \rceil \text{ and } 1 < r < \frac{3}{2}, \\ c_8 n^{1-\frac{r}{3}} \log^2 n & \text{if } k = \lceil n^r \rceil \text{ and } \frac{3}{2} \le r < 3. \end{cases}$$

where c_6, c_7, c_8 are positive constants.

Since there are $n(n+1)/2$ inputs for the recurrence (17), we have

$$T_1(x_n) \ge \frac{n(n+1)}{2} - 1,$$

while the trivial sequential algorithm establishes that

$$T_1(x_n) \le n(n+1).$$

There is a gap between the lower and upper bounds on $T_1(x_n)$. We believe that $T_1(x_n) = n^2 + O(n)$. Suppose that is true. Then from (18), we have $S_k(x_n) \to k$ as $n \to \infty$, i.e., optimal speed-up is achieved asymptotically, which would be in interesting contrast with pth order linear recurrences, where optimal speed-ups are not asymptotically achieved.

6. NONLINEAR RECURRENCES

A nonlinear recurrence is defined by

(19) $x_i = \varphi(x_{i-1}, x_{i-2}, \ldots, x_{i-p})$, $i \ge 1$,

where φ is a nonlinear rational function. Write $\varphi = \varphi_1/\varphi_2$ where φ_1 and φ_2 are polynomials which are relatively prime. Define the degree of a nonlinear recurrence to be

$$\deg \varphi = \max(\deg \varphi_1, \deg \varphi_2).$$

Hence, for example, the well known recurrence,

$$(20) \quad x_{i+1} = \frac{1}{2}(x_i + \frac{A}{x_i}),$$

for approximating $\sqrt{A}$ has degree 2. For linear recurrences in general we can have unbounded speed-up when $k \to \infty$ and $n \to \infty$. For example, by (9) we know that if $k = n$ the first order linear recurrence can be sped-up by a factor of $n/\log n$, which is unbounded as $n \to \infty$. The following theorem shows that the theory of nonlinear recurrences of degree > 1 is completely different from that of linear recurrences.

Theorem 4 [16]

For the recurrence defined by (19), if $\deg \varphi > 1$, then

$$S_k(x_n) \leq c_9, \forall k, \forall n,$$

for some constant c_9.

The theorem implies that, e.g., the recurrence defined by (20) cannot essentially be sped up by using parallelism.

The only nonlinear recurrences which can possibly have unbounded speed-up by using parallelism are of the form

$$(21) \quad x_i = \left(\sum_{j=i-p}^{i-1} a_{ij}x_j + b_i\right) \bigg/ \left(\sum_{j=i-q}^{i-1} c_{ij}x_j + d_i\right),$$

which is of degree one. Indeed, the recurrence

$$(22) \quad x_i = a_i + \frac{b_i}{x_{i-1}},$$

i.e., a continued fraction, can be sped up.

Theorem 5 [11, 12, 14, 23]

For the recurrence defined by (22),

$$(23)\quad \frac{1}{2}k + c_{10} \geq S_k(x_n) \geq \begin{cases} c_{11}\left(\frac{n}{\log n}\right) & \text{if } k \geq n, \\ \frac{2}{5}k \text{ as } n \to \infty, & \forall k. \end{cases}$$

for some constants c_{10}, c_{11}.

By Theorem 1 and (23) we note that recurrences with division seem to be more difficult than those without division in parallel computation. The same observation can also be made to the problem of evaluating arithmetic expressions (see [4, 23]).

It is clear that the recurrence

$$x_i = \frac{a_i x_{i-1} + b_i}{c_i x_{i-1} + d_i}$$

can also be sped up by using parallelism, since it can be transformed into a continued fraction. However, by the following theorem we know general recurrences defined by (21) cannot essentially be sped up.

Theorem 6 [11]

For the recurrence defined by (21) if either $q \geq 1$ or $p \geq 1$, then

$$S_k(x_n) \leq c_{12}, \ \forall k,n$$

for some constant c_{12}.

7. SUMMARY AND CONCLUSIONS

We have shown a number of results on the theoretical limitation of using parallelism for solving recurrences. For pth order linear recurrences, with k processors the speed-ups are shown to be bounded by $ck + d$ for some constants c, d, with $c < 1$, _no matter how large the size of the problems_. The sharp upper bound is obtained for first order linear recurrences. For nonlinear recurrences of degree > 1, the speed-ups are shown to be bounded by a constant, _no matter how many processors are used and how large the size of the problems_. This is probably the first and may be the only known example of a nontrivial problem which cannot be essentially sped up. By these results we wish to demonstrate that the gain from parallelism very much depends upon the nature of individual problems, e.g., the dependency relationships among the variables of the problems. We believe that to identify properties which prevent us from getting good speed-ups is fundamental for understanding parallel computation.

REFERENCES

[1] Barnes, G. H., et al., "The ILLIAC IV Computer," IEEE Trans. Comp., Vol. C-17, 1968, 746-757.

[2] Borodin, A. and I. Munro, Computational Complexity of Algebraic and Numeric Problems, American Elsevier, 1975.

[3] Brent, R. P., "On the Addition of Binary Numbers," IEEE Trans. Comp., Vol. C-19, 1970, 758-759.

[4] Brent, R. P., "The Parallel Evaluation of General Arithmetic Expressions," JACM, Vol. 21, 1974, 201-206.

[5] Chen, S. C. and D. J. Kuck, "Time and Parallel Processor Bounds for Linear Recurrence Systems," IEEE Trans. Comp., Vol. C-24, 1975, 701-717.

[6] Heller, D., "A Determinant Theorem with Applications to Parallel Algorithms," SIAM J. Numer. Anal. 11, 1974, 559-568.

[7] Heller, D., "On the Efficient Computation of Recurrence Relations," Report, ICASE, NASA Langley Research Center, Hampton, Virginia, 1974.

[8] Heller, D., "A Survey of Parallel Algorithms in Numerical Linear Algebra," 1975.

[9] Hyafil, L. and H. T. Kung, "The Complexity of Parallel Evaluation of Linear Recurrences," Proc. 7th Annual ACM Symposium on Theory of Computing, 1975, 12-22. To appear in JACM.

[10] Hyafil, L. and H. T. Kung, "Parallel Algorithms for Solving Triangular Linear Systems with Small Parallelisms.," contributed paper at 2nd Langley Conference on Scientific Computing, October, 1974. Also available as a Computer Science Department Report, Carnegie-Mellon University.

[11] Hyafil, L. and H. T. Kung, "Parallel Evaluation of Recurrences and Parallel Algorithms for Solving Band Linear Systems, Computer Science Department Report, Carnegie-Mellon University, 1975.

[12] Kogge, P. M., "Parallel Solution of Recurrence Problems," IBM J. Res. Develop. 18, 1974, 138-148.

[13] Kogge, P. M. and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Trans. Comp., Vol. C-22, 1973, 786-793.

[14] Kuck, D. J. and K. M. Maruyama, "The Parallel Evaluation of Arithmetic Expressions of Special Forms," Report RC 4276, IBM Research Center, Yorktown Heights, New York, 1973.

[15] Kuck, D. J. and K. M. Maruyama, "Time Bounds on the Parallel Evaluation of Arithmetic Expressions," SIAM Journal on Computing, 4(1975), 147-162.

[16] Kung, H. T., "New Algorithms and Lower Bounds for the Parallel Evaluation of Certain Rational Expressions," Proc. 6th Annual ACM Symposium on Theory of Computing, 1974, 323-333. To appear in JACM.

[17] Lambiotte, J. J., Jr. and R. G. Voigt, "The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer, Report, ICASE, NASA Langley Research Center, Hampton, Virginia, 1974.

[18] Munro, I. and M. Paterson, "Optimal Algorithms for Parallel Polynomial Evaluation," JCSS 7, 1973, 189-198.

[19] Orcutt, S. E., "Parallel Solution Methods for Triangular Linear Systems of Equations," Report 77, Digital Systems Lab., Stanford University, 1974.

[20] Stone, H. S., "An Efficient Parallel Algorithm for the Solution of a Tridiagonal System of Equations," JACM, Vol. 20, 1973, 27-38.

[21] Stone, H. S., "Parallel Tridiagonal Solvers," Digital Systems Lab., Stanford University, 1974.

[22] Strassen, V., "Gaussian Elimination is Not Optimal," Numerische Mathematik, Vol. 13, 1969, 354-356.

[23] Winograd, S., "On the Parallel Evaluation of Certain Arithmetic Expressions," Report RC 4808, IBM Research Center, Yorktown Heights, New York, 1974. To appear in JACM.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BOUNDS ON THE SPEED-UPS OF PARALLEL EVALUATION OF RECURRENCES		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) L. Hyafil and H. T. Kung		8. CONTRACT OR GRANT NUMBER(s) N00014-67-A-0314-0010, NR 044-422
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Department Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE September 1975
		13. NUMBER OF PAGES 19
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

None