The Complexity of Parallel Evaluation of Linear Recurrences

L. HYAFIL

IRIA-Laboria, Rocquencourt, France

AND

H. T. KUNG

Carnegie-Mellon University, Pittsburgh, Pennsylvania

ABSTRACT The problem of evaluating x_n defined by the linear recurrence $x_i = x_{i-1}b_i + a_{i+1}$, $i \geq 1$, $x_0 = a_1$, or equivalently evaluating the Horner expression $(\cdots(a_1b_1 + a_2)b_2 + \cdots + a_n)b_n + a_{n+1}$ is considered It is shown that by using an idealized k-processor parallel computer the speedup for the problem is at most $\frac{2}{3}k + \frac{1}{3}$ rather than k The bound is essentially sharp

KEY WORDS AND PHRASES complexity, parallel computation, speedup, parallel evaluation, linear recurrences, directed graph, arithmetic expressions

CR CATEGORIES 4 32, 5 10, 5 25, 5 32

1. *Introduction*

The concept of computers such as C.mmp and ILLIAC IV is to achieve computational speedup by performing several operations simultaneously with parallel processors. This type of computer organization is referred to as a parallel computer. In this paper we prove upper bounds on speedups achievable by parallel computers for a particular problem, the solution of first-order linear recurrences. We consider this problem because it is important in practice and also because it is so simply stated that we may obtain some insight into the nature of parallel computation by studying it.

Consider a k-processor parallel computer. It is understood that because of possible memory conflicts, delays due to interprocessors communication, etc., k-fold speedup is rarely achieved. In this paper we show that the speedup for a linear recurrence problem is at most $\frac{2}{3}k + \frac{1}{3}$, even under the assumption that the k-processor machine is idealized so that there are no memory conflicts or communication delays. We also show that the bound $\frac{2}{3}k + \frac{1}{3}$ is almost the best possible. Of course the actual speedup obtained from a real k-processor machine would be less than or equal to $\frac{2}{3}k + \frac{1}{3}$. The difference between $\frac{2}{3}k + \frac{1}{3}$ and k is rather significant. For example, if $k = 16, 64$, then the speedup 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 speedup we might expect for the problem is the inherent dependency of variables in the linear recurrence A related result in Kung [7] states that many *nonlinear* recurrences can be sped up at most by a constant

This research was supported in part by IRIA, France, and in part by the National Science Foundation under Grant GJ-32111 and the Office of Naval Research under Contract N0014-67-A-0314-0010, NR 044-422 Much of this work was carried out at IRIA while the second author was visiting there in the summer of 1974

Authors' addresses L Hyafil, IRIA-Laboria, Rocquencourt, France, H T Kung, Carnegie-Mellon University, Pittsburgh, PA 15213

factor, no matter how many processors are used. Hence these nonlinear recurrences must involve even more dependency relationships than the linear ones We believe that the study of these dependency relationships is fundamental for understanding parallel computation.

A graph representation of parallel algorithms is given in the Section 2. In Section 3 we give the results of this paper and discuss their significance. The proof of our main theorem (Theorem 2) is given in Section 4. In Section 5 we give a summary and make some remarks about this work. An illustration of the algorithm used in the proof of Theorem 2′, a generalized version of Theorem 2, is provided in the Appendix.

2. *Model of Computation*

Let λ be an infinite field and a_i, b_i, $i = 1, 2, \ldots$, indeterminates over λ. Suppose that we perform computations in the extension field $\lambda(a_1, b_1, a_2, b_2, \ldots)$. We consider the problem of evaluating rational expressions E in $\lambda(a_1, b_1, a_2, b_2, \ldots)$ given an input set I such that $\{a_1, b_1, a_2, b_2, \ldots\} \cup \lambda \subseteq I \subseteq \lambda(a_1, b_1, a_2, b_2, \ldots)$. (The recurrence considered in this paper in fact is an expression E of a special form. See Section 3.)

An *algorithm* for evaluating E given I is defined by a direct acyclic graph such that:

(a) The in-degree of any node is either 0 or 2. A node whose in-degree is zero is called an *input node*.

(b) There exists exactly one node whose out-degree is 0, and it is called the *output node*. The out-degree of any other node is greater than or equal to 1.

(c) Each input.node contains a value which is an element from the input set I. Each of the rest of the nodes contains an operation $+$, $-$, $\times$, or $/$ and a value which is the result of the operation on the values of the two predecessors. The value of the output node is E. (We often refer to a node by its value or its operation if there is no ambiguity.)

(d) Each node belongs to one of $t + 1$ sets, called levels and denoted by $l_0, l_1, \ldots, l_t$, such that (1) all input nodes are at level l_0, (2) for $i \geq 1$, the predecessors of any node at level l_i are from levels $l_0, \ldots, l_{i-1}$, (3) no level is empty

Then it can be easily verified that the output node is the only node at level l_t and that t is bounded below by the maximal length of a path from any input node to the output node. We define t to be the *time*, $k = \max_{1\leq i\leq t} |l_i|$ to be the *number of processors*, and $w = \sum_{1\leq i\leq t} |l_i|$ to be the *number of total operations* needed by the algorithm. An algorithm is called a sequential algorithm if $k = 1$ and a parallel algorithm if $k \geq 2$.

Example 2.1. The graph shown in Figure 1 defines a parallel algorithm for evaluating the Horner expression of seven variables,

$$H_7 = ((a_1b_1 + a_2)b_2 + a_3)b_3 + a_4,$$

given $I = \{a_1, b_1, a_2, b_2, a_3, b_3, a_4\}$ in time $t = 4$, with $k = 3$ processors and $w = 8$ operations

It seems to us that this model is very general and includes any algorithm for evaluating expressions. One should note, however, that we have implicitly assumed that each operation takes one unit of time and the following machine idealizations:

(a) Each processor can perform any of the four binary operations, but all processors do not necessarily perform the same operation at any time.

(b) No time is required to communicate data between processors.

Since we are interested in lower bounds on time and upper bounds on speedup, as we mentioned in Section 1, the results obtained in this paper for the idealized machine are certainly applicable for any real machine.

We now define some notation. For an expression E and an input set I, define $T_k(E \bmod I)$ = minimum time needed to evaluate E given I by an algorithm using k processors, and define the *speedup* of the problem of evaluating E given I by using k processors to be

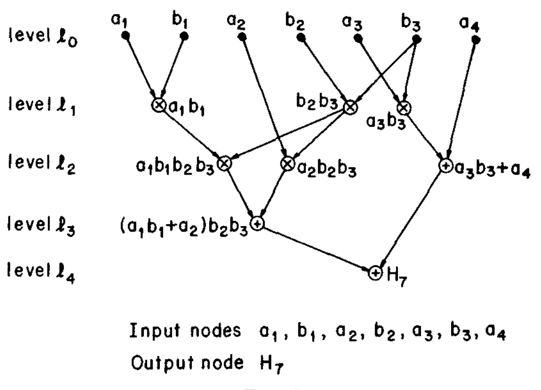


FIG 1

$$S_k(E \bmod I) = T_1(E \bmod I)/T_k(E \bmod I).$$

For simplicity we often write $T_k(E)$, $S_k(E)$ for $T_k(E \bmod I)$, $S_k(E \bmod I)$ if there is no ambiguity. In this paper we obtain lower bounds on $T_k(E)$ and upper bounds on $S_k(E)$ for E defined by linear recurrences.

3. *Results*

The problem considered in this paper is the following first-order linear recurrence problem.

$$\text{given } a_1, \ldots, a_{n+1}, b_1, \ldots, b_n, \quad x_0 = a_1, \quad x_i = x_{i-1}b_i + a_{i+1} \text{ for } i \geq 1, \tag{3.1}$$
compute x_n.

This is one of the most frequently occurring recurrences in practice. Consider the Horner expression of $2n + 1$ variables:

$$H_{2n+1} = ((\ldots((a_1b_1 + a_2)b_2 + a_3)b_3 + \cdots + a_{n-1})b_{n-1} + a_n)b_n + a_{n+1}.$$

It is clear that $x_n = H_{2n+1}$. Hence the recurrence problem is just the problem of evaluating H_{2n+1} given the input set

$$I = \{a_1, \ldots, a_{n+1}, b_1, \ldots, b_n\} \cup \lambda.$$

It is easy to see that the minimum time sequential algorithm is the obvious one defined by the recurrence (3.1). Hence we have

$$T_1(H_{2n+1}) = 2n. \tag{3.2}$$

Various parallel algorithms using $O(n)$ processors for the problem have appeared in many papers, including Brent [1], Kogge [5], Kuck and Maruyama [6], and Stone [9]. In this paper we are interested in algorithms using k processors, where k is a positive integer independent of n Brent [2] had the first result along this line. He showed that for a general arithmetic expression E_n of n variables and without division, if the input set is the set of all variables in E_n, then

$$T_k(E_n) \leq 2n/k + O(\log n). \tag{3.3}$$

Winograd [10] recently improved Brent's result for the case that $k \leq O(n/(\log n)^2)$. He showed

$$T_k(E_n) \leq [3/(2k)]n + O((\log n)^2). \tag{3.4}$$

Since H_{2n+1} is a special instance of E_{2n+1}, by (3.4) we have

$$T_k(H_{2n+1}) \leq (3/k)n + O((\log n)^2).$$

For the case that $k \ll n$, this bound can be easily improved by utilizing the special form of H_{2n+1}. We have the following theorem.

THEOREM 1.[1]

$$T_k(H_{2n+1}) \le [3/(k + \tfrac{1}{2})]\, n + O(\log k). \tag{3.5}$$

PROOF. Equation (3.1) is equivalent to

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = \begin{bmatrix} b_i & a_{i+1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ 1 \end{bmatrix}, \quad i \ge 1$$

Write

$$\begin{bmatrix} x_n \\ 1 \end{bmatrix} = \left(\prod_{l+1}^{n} \begin{bmatrix} b_i & a_{i+1} \\ 0 & 1 \end{bmatrix} \right) \left(\prod_{1}^{l} \begin{bmatrix} b_i & a_{i+1} \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x_0 \\ 1 \end{bmatrix},$$

where l is to be determined later. We use $k - 1$ processors to compute

$$\begin{bmatrix} b & a \\ 0 & 1 \end{bmatrix} = \prod_{l+1}^{n} \begin{bmatrix} b_i & a_{i+1} \\ 0 & 1 \end{bmatrix},$$

which takes time t_1, and one processor to compute

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = \begin{bmatrix} b_i & a_{i+1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ 1 \end{bmatrix}$$

for $i = 1, \ldots, l$, which takes time t_2.

Note that the multiplication of two matrices of the form $[\begin{smallmatrix} x & x \\ 0 & 1 \end{smallmatrix}]$ takes three operations and results in a matrix of the same form. Hence the obvious parallel algorithm gives

$$t_1 \le 3(\lceil (n - l)/(k - 1) \rceil - 1 + \lceil \log(k - 1) \rceil).$$

Note next that the multiplication of $[\begin{smallmatrix} x & x \\ 0 & 1 \end{smallmatrix}]$ and $[\begin{smallmatrix} x \\ 1 \end{smallmatrix}]$ uses two operations and results in a vector of the form $[\begin{smallmatrix} x \\ 1 \end{smallmatrix}]$. Hence

$$t_2 \le 2l.$$

Choose $l = \lceil 3n/(2k + 1) \rceil$. Then both $[\begin{smallmatrix} b & a \\ 0 & 1 \end{smallmatrix}]$ and $[\begin{smallmatrix} x_l \\ 1 \end{smallmatrix}]$ can be computed in time

$$\max(t_1, t_2) \le [3/(k + \tfrac{1}{2})]n + O(\log k).$$

By using the fact that

$$\begin{bmatrix} x_n \\ 1 \end{bmatrix} = \begin{bmatrix} b & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_l \\ 1 \end{bmatrix},$$

x_n can be obtained in two additional operations. □

The fundamental result of this paper is the following theorem.

THEOREM 2. *For any algorithm which evaluates H_{2n+1} given the input set I, we have*

$$w \ge 3n - t/2. \tag{3.6}$$

We defer the proof of this theorem until Section 4. We have the following two corollaries:

COROLLARY 1.

$$T_k(H_{2n+1}) \ge [3/(k + \tfrac{1}{2})]\, n, \ \forall k, \forall n. \tag{3.7}$$

PROOF The proof follows from (3.6) and the fact that $kt \ge w$. □

COROLLARY 2. *For any algorithm which evaluates H_{2n+1} given the input set I, if $t < 2n$, then*

$$w > 2n. \tag{3.8}$$

We now explain the significance of these corollaries. From Corollary 1 and (3.2), we obtain the main result of the paper:

[1] Theorem 1 has been proven independently by Chen [3] with a different approach

$$S_k(H_{2n+1}) \leq \tfrac{2}{3}k + \tfrac{1}{3}, \forall k, \forall n; \tag{3.9}$$

i.e. the speedup of the problem is at most $\frac{2}{3}k + \frac{1}{3}$ *rather than k when a k-processor machine is used.* Also, by (3.5) and (3.7),

$$[3/(k + \tfrac{1}{2})]\, n \leq T_k(H_{2n+1}) \leq [3/(k + \tfrac{1}{2})]\, n + O(\log k).$$

Hence the bounds in (3.5) and (3.7) are asymptotically the best possible bounds as $n \to \infty$, and so is the bound in (3.9). Furthermore one should observe that by (3.2) the bound in (3.7) is sharp for $k = 1$.

We now turn to Corollary 2. By the corollary, any algorithm which takes time less than $2n$ must perform more than $2n$ operations. Since the sequential algorithm uses $2n$ operations, we conclude that *if a parallel algorithm is faster than the sequential algorithm, then it requires more work than the sequential algorithm does.* This result plays a crucial role when we want to convert a parallel algorithm using an unlimited number of processors into one using k processors (see Lemma 2 in Brent [2] and Winograd [10]). The tradeoff between t and w has been observed in other problems (see, e.g Hyafil and Kung [4]). It becomes particularly significant for the pipeline computers such as CDC STAR-100, where both t and w are important. (An excellent discussion on this matter can be found in Lambiotte and Voigt [8]) We view Theorem 2 as a start toward understanding the tradeoff between t and w in parallel computation.

4 *Proof of Theorem* 2

In this section we consider generalized Horner expressions:

$$G_{2n+1} = ((\cdots((A_1B_1 + A_2)B_2 + A_3)B_3 + \cdots + A_{n-1})B_{n-1} + A_n)B_n + A_{n+1},$$

where

$$A_i \in L(a_i) = \{\mu a_i + \nu \mid \mu, \nu \in \lambda\}, \quad B_i \in L(b_i) = \{\mu b_i + \nu \mid \mu, \nu \in \lambda\},$$

and A_i, $B_i \notin \lambda$. Let

$$J = \bigcup_{1 \leq i \leq n} [L(a_i) \cup L(b_i)].$$

Instead of proving Theorem 2, we prove the following *stronger* theorem.

THEOREM 2′. *For any algorithm which evaluates G_{2n+1} given the input set J, $w \geq 3n - t/2$.*

We first establish three lemmas. The first lemma says that, for evaluating G_{2n+1}, if no input node is used more than once and if all input nodes are essentially distinct, then the operation involving a_1 and b_1 is the only possible operation which can be performed at level l_1.

LEMMA 1. *In the directed graph of any algorithm for evaluating G_{2n+1} given J, if the out-degree of each input node is exactly equal to* 1 *and if each a_i or b_i does not appear in more than one input node, then there is only one node at level l_1 and its value is a rational expression of a_1 and b_1 over λ.*

PROOF Let V be the value of a node at level l_1. Then V is a rational expression of a_i, b_j, of a_i, a_j, or of b_i, b_j. We shall only prove the lemma for the case that V is a rational expression of a_i, b_j, since the proofs for the other cases are similar.

Because the out-degree of every input node is 1 and each of a_i and b_j does not appear in more than one input node, the value of the output node, G_{2n+1}, is a rational expression R of V, a_h, and b_k for $h \neq i$ and $k \neq j$ By comparing the coefficients of a_i and b_j in G_{2n+1} and in R, one can check that V must be a rational expression of a_1 and b_1. This also proves that there is only one node at level l_1. In fact it is possible to show that the only operation at level l_1 is a multiplication of the form $(\mu_1 a_1 + \nu_1)(\mu_2 b_1 + \nu_2)$ where $\mu_i, \nu_i \in \lambda$ □

The following lemma is similar to Lemma 1 and hence the proof is omitted

LEMMA 2. *If in addition to the assumption of Lemma* 1, *we assume that the out-degree* *of the only node at level l_1 is 1, then there is only one node at level l_2.*

One of the most important properties of generalized Horner expressions is their reproducing property, which is stated in the following lemma.

LEMMA 3. *If in G_{2n+1}, a_i is replaced by α and b_i by β for some $1 \leq i \leq n$, with $\alpha, \beta \in \lambda$, the form of generalized expressions still remains; more precisely, G_{2n+1} is reduced to G_{2n-1}.*

PROOF OF THEOREM 2′. Let D be the directed graph of any algorithm which evaluates G_{2n+1} given J. The proof of the theorem is based on a reduction algorithm, called Algorithm A, on the graph D. Algorithm A first eliminates a_{n+1} in D and then eliminates the pairs (a_i, b_i) one at a time. (Eliminations are done by substituting suitable constants in λ for the indeterminants, the a_i and b_i.) To be precise, Algorithm A is defined in the following. Also see the Appendix, where Algorithm A has been worked out with respect to the graph used in Example 2.1.

ALGORITHM A

1. [Initialize.]
   - 1.1 Replace a_{n+1} by α_{n+1} in the graph D, where $\alpha_{n+1} \in \lambda$ is so chosen that after the replacement the value of each node in D is well defined, i.e. no division of zero occurs. (It is easy to see that such an α_{n+1} exists.)
   - 1.2 Let P be $\{1, 2, \ldots, n\}$. (P is the set of the indexes of those indeterminants a_i and b_i which have not been eliminated from D.)
2. [Check the conditions of Lemma 1.] If the conditions of Lemma 1 are satisfied, then let i be the smallest integer in P and go to step 4.
3. [Choose i for step 4.] Choose i in P such that a_i or b_i appears either in more than one input node or in an input node with out-degree greater than 1. (Such an i exists, since conditions of Lemma 1 are not satisfied.)
4. [Eliminate (a_i, b_i).] Replace a_i by α_i and b_i by β_i in D, where $\alpha_i, \beta_i \in \lambda$ are so chosen that after the replacements the value of each node in D is well defined.
5. [Reduce the graph D.] Eliminate those nodes in D whose successors all have values in J, and eliminate the arcs connected to them. (Note that by Lemma 3 the reduced graph remains an algorithm for evaluating a generalized Horner expression, which is obtained by substituting α_i for a_i and β_i for b_i in the original expression.)
6. [Update P and check if it is empty.] Set $P \leftarrow P - \{i\}$. If $P = \emptyset$, terminate the algorithm, otherwise return to step 2.

The following three facts can be easily verified.

(a) In step 1 at least one nonscalar operation in D is converted into a scalar operation. ("x op y" is defined to be a scalar operation if op $\in \{+, -, \times\}$ and at least one of x and y belongs to λ, or if op $= /$ and $y \in \lambda$.)

(b) If the conditions of Lemma 1 are not satisfied in step 2, then at least three nonscalar operations in D are converted into scalar operations in step 4.

(c) If the conditions of Lemma 1 are satisfied in step 2, then there are two cases:

Case 1. The additional condition in Lemma 2 is satisfied. In this case, by Lemma 2 the depth of the reduced graph obtained in step 5 is less than that of the original graph by at least 2. Furthermore in step 4 at least two nonscalar operations are converted into scalar operations when $|P| > 1$ and one nonscalar operation into a scalar operation when $|P| = 1$.

Case 2. The additional condition in Lemma 2 is not satisfied. This implies that the out-degree of the only node at level l_1 is at least 2. Hence in step 4 at least three nonscalar operations are converted into scalar operations when $|P| > 1$ and two nonscalar operations into scalar operations when $|P| = 1$. Moreover the depth of the reduced graph obtained in step 5 is less than that of the original graph by at least 1.

The main loop of the algorithm (steps 2–5) is executed n times since n pairs of (a_i, b_i) have to be eliminated. Let u be the number of times that the conditions of Lemma 1 are not satisfied in step 2 and v the number of times that the conditions are satisfied. Then $n = u + v$. Furthermore let $v_1 (v_2)$ be the number of times that the conditions of Lemma 1 are satisfied in step 2 and case 1 (case 2) is true, respectively. Then $v = v_1 + v_2$. By (a), (b), and (c) the number of nonscalar operations which are converted in scalar operations by Algorithm A is at least $3u + 2v_1 + 3v_2$. We have $w \geq 3u + 2v_1 + 3v_2$. Hence $w \geq 3n - v_1$. By (c), $t \geq 2v_1 + v_2$. Therefore,

$$w \geq 3n - t/2 + v_2/2 \geq 3n - t/2.$$

Equality is only possible if $v_2 = 0$. □

5. *Summary and Concluding Remarks*

We have proved that for the first-order linear recurrence problem, the speedup achievable by a k-processor parallel computer is at most $\frac{2}{3}k + \frac{1}{3}$, no matter how large the size n of the problem The technique used in the proof appears to be new in complexity theory It is expected that the technique can be used for other problems.

We wish to demonstrate by this result that the gain from parallelism very much depends upon the nature of individual problems, e.g. the dependency relationships among the variables of the problem. We believe that identifying properties which prevent us from getting good speedups is fundamental for understanding parallel computation.

Appendix

We illustrate Algorithm A with respect to the graph shown in Figure 2

1. $a_4 \leftarrow 1$. We obtain Figure 3.

2. $a_3 \leftarrow 2$, $b_3 \leftarrow 3$. We obtain Figure 4. The graph in Figure 4 is reduced to that in Figure 5.

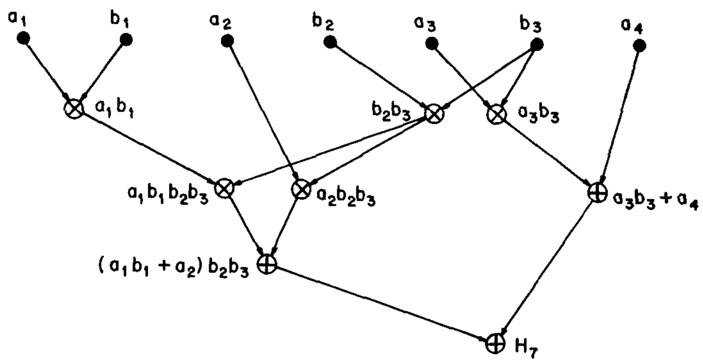


FIG 2
● = input node, ⊗, ⊕ = node with nonscalar operation

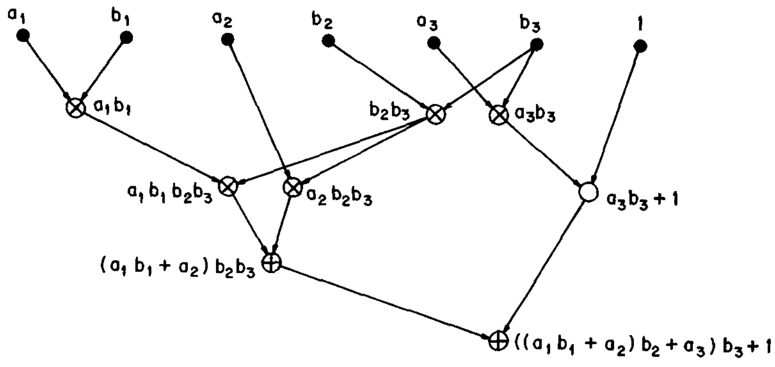


FIG 3
● = input node, ○ = node with scalar operation, ⊗, ⊕ = node with nonscalar operation

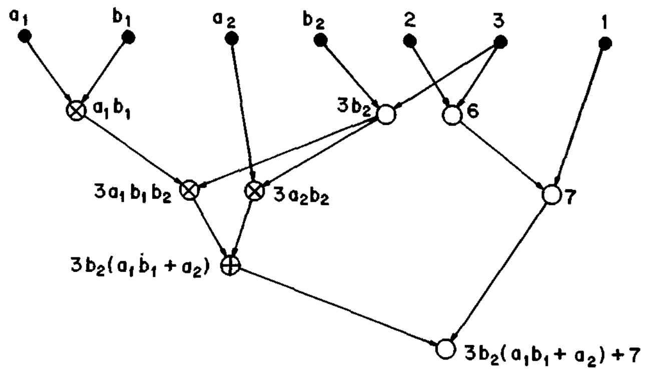
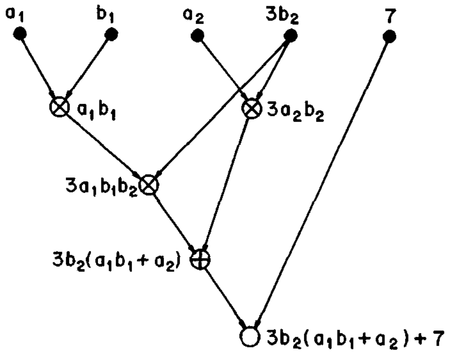


FIG 4

● = input node, ○ = node with scalar operation, ⊗, ⊕ = node with nonscalar operation



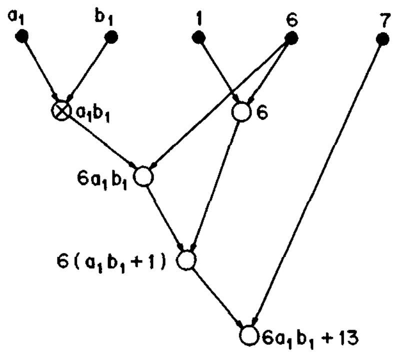
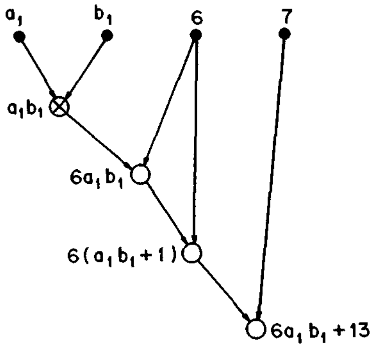


FIG 5 FIG 6

● = input node, ○ = node with scalar operation, ⊗, ⊕ = node with nonscalar operation



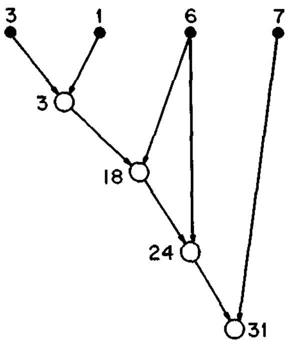


FIG 7 FIG 8

● = input node, ○ = node with scalar operation, ⊗ = node with nonscalar operation

3. $a_2 \leftarrow 1$, $b_2 \leftarrow 2$. We obtain Figure 6. The graph in Figure 6 is reduced to that in Figure 7.
4. $a_1 \leftarrow 3$, $b_1 \leftarrow 1$. We obtain Figure 8.
5. The graph in Figure 8 is reduced to 31.

ACKNOWLEDGMENTS. The authors would like to thank Ron Rivest for helpful discussions and J.F. Traub for his comments.

REFERENCES

1 BRENT, R P On the addition of binary numbers *IEEE Trans Comptrs. C-19* (1970), 758–759.
2 BRENT, R P The parallel evaluation of general arithmetic expressions *J ACM 21,* 2 (April 1974), 201–206
3 CHEN, S C Speedup of iterative programs in multiprocessing systems. Ph.D Th , Rep 694, Dep. Comptr Sci , U of Illinois at Urbana-Champaign, Urbana, Ill , Jan 1975
4 HYAFIL, L , AND KUNG, H T Parallel algorithms for solving triangular linear systems. Comptr Sci. Dep Rep , Carnegie-Mellon U , Pittsburgh, Pa , Oct 1974
5 KOGGE, P M Parallel solution of recurrence problems *IBM J Res Develop 18* (1974), 138–148
6 KUCK, D J , AND MARUYAMA, K M The parallel evaluation of arithmetic expressions of special forms Rep RC 4276, IBM Res Ctr , Yorktown Heights, N Y , March 1973
7 KUNG, H T New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences *J ACM 23,* 2 (April 1976), 252–261
8 LAMBIOTTE, J J JR , AND VOIGT, R G The solution of tridiagonal linear systems on the CDC STAR-100 computer *ACM Trans Math Software 1,* 4 (Dec. 1975), 308–329
9. STONE, H S An efficient parallel algorithm for the solution of a tridiagonal system of equations. *J. ACM 20,* 1 (Jan 1973), 27–38
10 WINOGRAD, S On the parallel evaluation of certain arithmetic expressions *J ACM 22,* 4 (Oct 1975), 477–492

RECEIVED DECEMBER 1974, REVISED SEPTEMBER 1976