Parallel Prefix Computation

RICHARD E. LADNER AND MICHAEL J. FISCHER

University of Washington, Seattle, Washington

ABSTRACT The prefix problem is to compute all the products $x_1 \circ x_2 \circ \cdots \circ x_k$ for $1 \leq k \leq n$, where $\circ$ is an associative operation A recursive construction is used to obtain a product circuit for solving the prefix problem which has depth exactly $\lceil \log_2 n \rceil$ and size bounded by $4n$ An application yields fast, small Boolean circuits to simulate finite-state transducers. By simulating a sequential adder, a Boolean circuit which has depth $2\lceil \log_2 n \rceil + 2$ and size bounded by $14n$ is obtained for n-bit binary addition The size can be decreased significantly by permitting the depth to increase by an additive constant

KEY WORDS AND PHRASES automaton, binary addition, circuit, combinational complexity, depth, fanout, parallelism, size, transducer

CR CATEGORIES 5.22, 5 25, 6 1, 6 32

1. *Introduction*

Many algorithmic problems are easy to solve sequentially with finite memory. Examples are the addition of two binary numbers and the division of a binary number by a constant. By way of contrast, efficient parallel solutions to these same problems (restricted to inputs of a fixed length) seem complicated and mysterious and highly dependent on special properties of the particular problem. For example, the "carry lookahead" circuit for binary addition [6, 10] seems to rely on the details of carry propagation for its operation.

In this paper we give a general method for deriving efficient parallel solutions to the fixed-length version of any problem solved by a finite-state transducer. Our construction consists of two parts. First we exhibit a class of efficient parallel solutions to a fundamental abstract problem, the prefix problem. We then show how to use such a solution to "simulate" a finite-state transducer efficiently. The result is an efficient parallel solution to the original problem solved by the finite-state transducer.

Let $\circ$ be an associative operation on a domain D. The *prefix problem* is to compute, for given $x_1, \ldots, x_n \in D$, each of the products $x_1 \circ x_2 \circ \cdots \circ x_k$, $1 \leq k \leq n$.

By analogy with Boolean combinational circuits [7, 8], we consider *product circuits*, which are directed acyclic oriented graphs. Each node of indegree 2 represents a product of its two inputs. All other nodes have indegree 0 and are labeled with an integer between 1 and n. These are the input nodes. With each node v we associate an element of D in the obvious way.

We consider two complexity measures on a product circuit $\mathcal{N}$. $C(\mathcal{N})$, the *size*, is the number of product nodes in $\mathcal{N}$, and $D(\mathcal{N})$, the *depth*, is the maximum number of product nodes on any directed path in $\mathcal{N}$. For example, the circuit of Figure 1 has depth 3, size 4, and computes $x_1 \circ x_3 \circ x_3 \circ x_2 \circ x_3$. Note that it also computes $x_1 \circ x_3 \circ x_3 \circ x_2$, $x_1 \circ x_3$, and $x_3 \circ x_2$.

A preliminary version of this paper was presented at the 1977 International Conference on Parallel Processing, Bellaire, Mich , August 1977

This material is based on work supported by the National Science Foundation under Grant DCR 74-12997-A01, through a subcontract from M.I.T , and Grants MCS 74-12764 and MCS 77-02474

Authors' address. Department of Computer Science, FR-35, University of Washington, Seattle, WA 98195

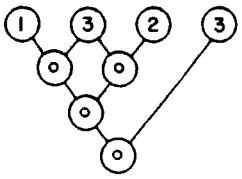

FIG. 1. A product circuit. (All arcs are directed downward.)

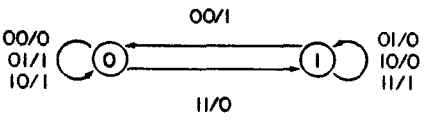

FIG. 2. A sequential adder.

The depth of a circuit corresponds to the computation time in a parallel computation environment, whereas the size represents the amount of hardware required. For the prefix problem it is straightforward to construct a circuit of the minimum possible size, $n - 1$, but its depth is also $n - 1$. Similarly, it is not difficult to find a circuit of depth exactly $\lceil \log_2 n \rceil$, the minimum possible depth, but the immediate recursive construction yields a circuit of size $\Omega(n \log n)$.[1] In Section 2 we find a solution to the prefix problem of minimum depth $\lceil \log_2 n \rceil$ and size $<4n$.

In Section 3 we obtain a family of circuits for simulating a given arbitrary finite-state transducer on inputs of length n which all have depth $O(\log n)$ and size $O(n)$. In Section 4 we apply those constructions to the simple machine for binary addition of Figure 2 and analyze the constants carefully. One result of our general methods is a circuit of size $8n + 6$ and depth $4\lceil \log_2 n \rceil + 2$ which is essentially the same as the "carry lookahead" adder [10]. Changing the parameters of the construction decreases the depth to only $2\lceil \log_2 n \rceil + 2$, while the size increases to $14n$, which is possibly advantageous in certain practical situations. Asymptotically, there is still a factor-of-2 gap between the depth achieved by our general methods and the best depth obtainable for addition. Brent has an adder of depth $\log_2 n + O(\sqrt{\log_2 n})$, but its size is $\Omega(n \log_2 n)$ [2]. Krapchenko achieves the same bound on depth with a linear size circuit [5, 8].

2. *Circuits for the Prefix Problem*

In this section we define a family of circuits $\mathscr{P}_k(n)$ for solving the prefix problem on n inputs. For each k the depth $D(\mathscr{P}_k(n)) \leq k + \lceil \log_2 n \rceil$. The size $C(\mathscr{P}_k(n)) \leq 2(1 + 1/2^k)n - 4$ for all $n \geq 1$ and $0 \leq k \leq \lceil \log_2 n \rceil$. For small n the size is substantially smaller than this bound would suggest.

The recursive construction of $\mathscr{P}_0(n)$ is shown in Figure 3, and the construction of $\mathscr{P}_k(n)$ for $k \geq 1$ is shown in Figure 4. When $n = 1$, $\mathscr{P}_k(n)$ is simply a single input node and contains no products. In the figures, circles represent concatenation nodes.

Figure 5 illustrates the construction of $\mathscr{P}_k(n)$ for small values of n.

ANALYSIS OF SIZE AND DEPTH. That the constructions achieve the desired depth follows easily by induction, given the additional fact, also proved by induction, that the last output in $\mathscr{P}_k(n)$ has depth exactly $\lceil \log_2 n \rceil$, even when $k > 0$. The correctness of the construction is also easily shown by induction and is left to the reader.

Let $S_k(n) = C(\mathscr{P}_k(n))$. Then S satisfies the following recurrences:

$$S_0(n) = S_1\left(\left\lceil \frac{n}{2} \right\rceil\right) + S_0\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \left\lfloor \frac{n}{2} \right\rfloor, \qquad n \geq 2;$$

$$S_k(n) = S_{k-1}\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 1, \qquad n \text{ even} \quad \text{and} \quad n \geq 2, \quad k \geq 1;$$

$$S_k(n) = S_{k-1}\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 2, \qquad n \text{ odd} \quad \text{and} \quad n \geq 3, \quad k \geq 1;$$

$$S_k(1) = 0, \qquad k \geq 0.$$

[1] $f = \Omega(g)$ iff $g = O(f)$.

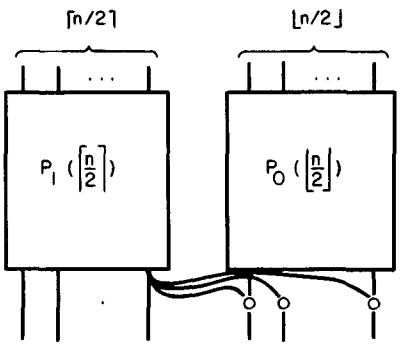

FIG 3. The construction of $\mathscr{P}_0(n)$.

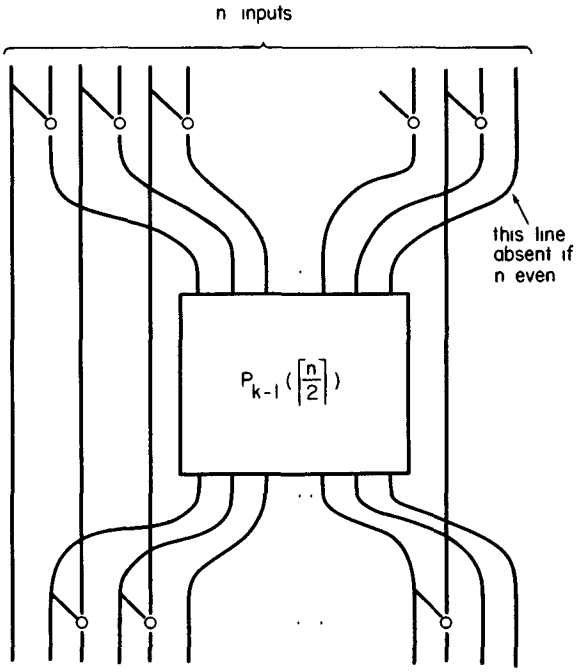

FIG 4 The construction of $\mathscr{P}_k(n)$, $k \geq 1$

When n is a power of 2, we get exact solutions

$$S_0(n) = 4n - F(5 + \log_2 n) + 1,$$
$$S_1(n) = 3n - F(4 + \log_2 n),$$

and more generally, when $0 \leq k \leq \log_2 n$,

$$S_k(n) = S_0\left(\frac{n}{2^k}\right) + n \cdot \left(2 - \frac{1}{2^{k-1}}\right) - k$$
$$= 2\left(1 + \frac{1}{2^k}\right)n - F(5 + \log_2 n - k) + 1 - k.$$

Here $F(m)$ denotes the mth Fibonacci number, and $F(m) = (\phi^m - \hat{\phi}^m)/\sqrt{5}$, where $\phi = (1 + \sqrt{5})/2$ and $\hat{\phi} = (1 - \sqrt{5})/2$ (cf. [3]). Thus for large n and fixed k, $S_k(n)$ is bounded by $2(1 + 1/2^k)n - a_k \cdot n^{0.69424}$, where $a_k > 0$ is a constant depending only on k. Some values of $S_k(n)$ are shown in Figure 6.

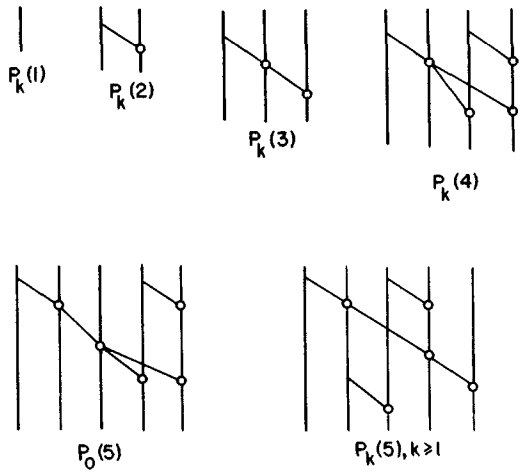

FIG. 5 The $\mathscr{P}_k(n)$ circuits for $1 \leq n \leq 5$

n \ k	0	1	2	3	4	5	6	7
1	0							
2	1	1						
4	4	4	4					
8	12	11	11	11				
16	31	27	26	26	26			
32	74	62	58	57	57	57		
64	168	137	125	121	120	120	120	
128	369	295	264	252	248	247	247	247

FIG 6 $S_k(n)$ for n a small power of 2

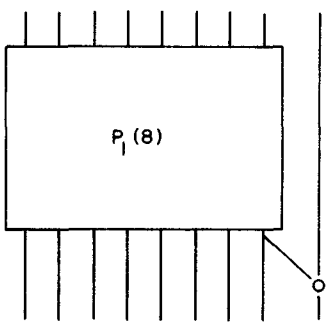

FIG. 7. A solution to the 9-input prefix problem

When n is not a power of 2, we do not have an exact solution, but it is easily verified by induction that $S_k(n) < 2(1 + 1/2^k)n - 2$, $n \geq 1$. In fact, we know that $\mathscr{P}_k(n)$ is not optimal for n not a power of 2. For example, $C(\mathscr{P}_0(9)) = 13$, but the circuit of Figure 7 has size only 12 since $S_1(8) = 11$, and it also has minimal depth 4. It is an open problem to determine just how to split the circuit to optimize the construction using the methods of Figures 3 and 4.

There is an analogy between product circuits and addition chains [4, 9]. Let D be the natural numbers, $\circ$ be ordinary addition, and fix each input to 1. Then the minimum size circuit for computing a number n is exactly the length of the shortest addition chain for m. A prefix circuit on n inputs under this interpretation constructs each of the integers from 1 to n. Unlike most of the work on addition chains, we are interested in the depth as well as in the size. As with addition chains, analysis becomes much more difficult for n not a power of 2.

ANALYSIS OF FANOUT. The *fanout* of an input or product node in a circuit is its outdegree, and the fanout of a circuit is the maximum fanout of any node. In some applications, fanout is an important consideration along with size and depth.

For the circuits $\mathscr{P}_k(n)$, we happen to be able to give an exact characterization of the fanout. To begin, define the ith *output* node to be the one which computes the ith output value of the network. A node which is neither an input nor an output node is said to be *internal*. (In Figures 3–5 and 7 the output nodes are identified by vertical lines leading up from the bottom, but these lines are not counted in the fanout calculations.)

In $\mathscr{P}_k(n)$ the first output node is the first input node, and the other outputs are product nodes. Every input node has fanout ≤ 2, and for every internal node there is an output node of fanout at least as great. These facts are easily verified by induction on n, where the induction hypothesis is strengthened to show that the first input has fanout ≤ 1 and no output node has fanout exceeding $n-1$. We conclude that the fanout of $\mathscr{P}_k(n)$ equals the maximum fanout of an output node unless that quantity is 1. That happens, as we shall see, only for $n \leq 3$, in which case the fanout is easily determined from Figure 5.

Let $\mathrm{fo}(k, n, i)$ be the fanout of the ith output node of $\mathscr{P}_k(n)$. An easy induction establishes that $\mathrm{fo}(k, n, 1) = 1$ for all $n \neq 1$ and $\mathrm{fo}(k, n, n) = 0$ for all n. Also, $\mathrm{fo}(k, 3, 2) = 1$. For $n \geq 4$ and $1 < i < n$, $\mathrm{fo}(k, n, i)$ satisfies the following recurrences:

$$\text{If } k = 0, \qquad \mathrm{fo}(k, n, i) = \begin{cases} \mathrm{fo}\left(1, \left\lceil \frac{n}{2} \right\rceil, i\right) & \text{if } i < \left\lceil \frac{n}{2} \right\rceil; \\ \left\lfloor \frac{n}{2} \right\rfloor & \text{if } i = \left\lceil \frac{n}{2} \right\rceil; \\ 0 & \text{if } i > \left\lceil \frac{n}{2} \right\rceil. \end{cases}$$

$$\text{If } k > 0, \qquad \mathrm{fo}(k, n, i) = \begin{cases} 0 & \text{if } i \text{ is odd and } i \neq 1; \\ \mathrm{fo}\left(k-1, \left\lceil \frac{n}{2} \right\rceil, \frac{i}{2}\right) + 1 & \text{if } i \text{ is even and } i < n-1; \\ \mathrm{fo}\left(k-1, \left\lceil \frac{n}{2} \right\rceil, \frac{i}{2}\right) & \text{if } i \text{ is even and } i \geq n-1. \end{cases}$$

Let $B(k, n) = \lfloor (n + 2^k - 1)/2^{k+1} \rfloor + k$. It can be shown that for all $n > 2^{k+1}$, $\mathrm{fo}(k, n, i) \leq B(k, n)$. Moreover, this bound is best possible; that is, there is an $i(k, n)$ such that $\mathrm{fo}(k, n, i(k, n)) = B(k, n)$. $i(k, n)$ is given by the formula

$$i(k, n) = \begin{cases} 2^{k-1} \cdot \left\lceil \frac{n}{2^{k+1}} \right\rceil & \text{if } 2^{k+1} < n \leq 2^{k+1} + 2^{k-1}; \\ 2^k \cdot \left\lceil \frac{n}{2^{k+1}} \right\rceil & \text{if } n > 2^{k+1} + 2^{k-1}. \end{cases}$$

Putting these results together with the fact that when $n \leq 2^{k+1}$ and $k \geq 1$, then $\mathscr{P}_k(n) = \mathscr{P}_{k-1}(n)$, we have a complete characterization of the fanout of $\mathscr{P}_k(n)$ for all k and n.

3. *Application to Finite-State Machines*

A classic example of a sequential process is a finite-state transducer (cf. [1]). Given an input of length n and an initial state, we show below how to compute in parallel the output and final state. This method leads to the construction of fast Boolean circuits that simulate finite-state transducers.

We use the Mealy model of a *finite-state transducer* which is a five-tuple $M = (Q, \Sigma, \Delta, \delta, \gamma)$, where Q is a finite set of states, Σ is the input alphabet, Δ is the output alphabet, $\delta: Q \times \Sigma \to Q$ is the transition function, and $\gamma: Q \times \Sigma \to \Delta$ is the output function.

For each input symbol a we define a function $M_a: Q \to Q$ by $qM_a = \delta(q, a)$. (The argument to M_a is on the left.) Given an input word $a_1 a_2 \cdots a_k$, the state $qM_{a_1} \circ M_{a_2} \circ \cdots \circ M_{a_k}$ is the state of M after reading $a_1 \cdots a_k$ starting in state q, where $\circ$ denotes functional composition.

A parallel algorithm for computing the output and final state given the input $a_1 a_2 \cdots a_k$ and the initial state q_0 is

1 Compute $M_{a_1}, M_{a_2}, \ldots, M_{a_n}$ in parallel.

2 Compute $N_1 = M_{a_1}, N_2 = M_{a_1} \circ M_{a_2}, \ldots, N_n = M_{a_1} \circ M_{a_2} \circ \cdots \circ M_{a_n}$ by a parallel prefix algorithm

input xy	function gp
00	00
10	01
01	01
11	10

$g = x \wedge y$
$p = x \oplus y$

FIG. 8 Computation of the function from the inputs.

function g_1p_1 \ function g_2p_2	00	01	10
00	00	00	10
01	00	01	10
10	00	10	10

$g = g_2 \vee (g_1 \wedge p_2)$
$p = p_1 \wedge p_2$

FIG. 9 Composition table

3. Compute $q_1 = q_0N_1$, $q_2 = q_0N_2$, ..., $q_n = q_0N_n$ in parallel
4. Compute $b_1 = \gamma(q_0, a_1)$, $b_2 = \gamma(q_1, a_2)$, ..., $b_n = \gamma(q_{n-1}, a_n)$ in parallel

The output is $b_1b_2 \cdots b_n$ and the final state is q_n.

Let c_1 (d_1) be the size (depth) of computing M_a, c_2 (d_2) be the size (depth) of computing functional composition, c_3 (d_3) be the size (depth) of computing functional evaluation, and c_4 (d_4) be the size (depth) of computing $\gamma(q, a)$. Given an input of length n and an initial state, the size and depth for computing the output and final state is

$$\text{SIZE} \leq c_2c(n) + (c_1 + c_3 + c_4)n,$$
$$\text{DEPTH} \leq d_2d(n) + d_1 + d_3 + d_4,$$

where $c(n)$ ($d(n)$) is the size (depth) of a product circuit for solving the prefix problem. (Note: we assume that the state q_0 can be coded or decoded at no cost.)

There are several ways of obtaining Boolean circuits from this method. One simple way is to represent the M_a's as $s \times s$ Boolean matrices, where s is the number of states. Functional composition is Boolean matrix multiplication, and functional evaluation is the Boolean product of a matrix and a vector. For this representation, using the standard matrix multiplication algorithm and the prefix circuit $\mathscr{P}_0$ (or $\mathscr{P}_k$ for fixed k), we can construct a Boolean circuit for inputs of length n with linear size and depth $(1 + \log_2 s)\log_2 n + d$, where d is a constant depending only on M.

For a fixed finite-state machine there may be a particularly good representation for the functions which would lead to a smaller or faster circuit. In the next section we find such a representation for the addition finite-state machine of Figure 2.

4. *Application to Binary Addition*

Consider the finite-state transducer A of Figure 2. There are three functions A_{00}, $A_{01} = A_{10}$, A_{11} on states which are closed under composition. We repesent them by a pair of bits g, p (for generate and propagate, respectively) as shown in Figure 8. The composition table is shown in Figure 9, and the evaluation table in Figure 10.

From Figure 8 the inputs can be represented by the initial g, p pair, so we get the output table shown in Figure 11.

By observation we can calculate the constants

$$c_1 = 2, \quad d_1 = 1,$$
$$c_2 = 3, \quad d_2 = 2,$$
$$c_3 = 2, \quad d_3 = 2,$$
$$c_4 = 1, \quad d_4 = 1.$$

The basic costs for addition are $\text{SIZE} \leq 3c(n) + 5n$ and $\text{DEPTH} \leq 2d(n) + 4$. There are certain refinements that can be made.

state s	function g p: 00	01	10
0	0	0	1
1	0	1	1

$t = g \vee (s \wedge p)$

FIG. 10 Evaluation table

state t	input g p: 00	01	10
0	0	1	0
1	1	0	1

$z = t \oplus p$

FIG 11. Output table.

number of bits	k=0 DEPTH	k=0 SIZE	k=1 DEPTH	k=1 SIZE	k=2 DEPTH	k=2 SIZE	k=3 DEPTH	k=3 SIZE
4	6	20	8	20	10	20		
8	8	52	10	49	12	49	14	49
16	10	125	12	113	14	110	16	110
32	12	286	14	250	16	238	18	235
64	14	632	16	539	18	503	20	491
128	16	1363	18	1141	20	1048	22	1012

FIG 12 DEPTH and SIZE of small adders

(1) Let the input state be the constant 0. The evaluation table reduces to $t = g$. There is no "evaluation," so there is no need to compute p at the last level before step 3. This results in a total savings of $3n$ in size and 2 in depth, so SIZE $\leq 3c(n) + 2n$ and DEPTH $\leq 2d(n) + 2$.

(2) We may obtain an n-bit adder with the state as an additional "carry-in" input by forming an $(n + 1)$-bit adder which starts in state 0 and uses the lowest order bits to simulate the incoming state. This observation leads to an adder of SIZE $\leq 3c(n + 1) + 2n$ and DEPTH $\leq 2d(n + 1) + 2$.

(3) These techniques can also be used to construct ones-complement adders. Because of the "end-around" carry the input state is a function of the input numbers. The input state is computed in step 2, which makes it available for step 3 where it is used. In this case the adder has SIZE $\leq 3c(n) + 5n$ and DEPTH $\leq 2d(n) + 4$.

Using the results of Section 2 and observation (1) above, it is seen that there exist Boolean circuits to compute n-bit sums (with no carry in) of

$$\text{SIZE} \leq \left(8 + \frac{6}{2^k}\right)n \qquad \text{and} \qquad \text{DEPTH} \leq 2\log_2 n + 2k + 2$$

for $0 \leq k \leq \log_2 n$.

Notice that if we set $k = \log_2 n$, then we obtain a circuit of SIZE $\leq 8n + 6$ and DEPTH $\leq 4 \log_2 n + 2$. These bounds are similar to those obtained for the "carry-lookahead" adder [10]. We believe that our circuit $\mathscr{P}_k(n)$ for $k = \log_2 n$ is essentially the same as the "carry-lookahead" adder.

The table of Figure 12 illustrates the trade-offs that can be made between size and depth in small adders. The numbers of Figure 12 are based on those of Figure 6 together with observation (1).

ACKNOWLEDGMENTS. We wish to thank Glenn Goodrich and Garret Swart for several helpful discussions.

REFERENCES

1. BOOTH, T.L. *Sequential Machines and Automata Theory.* Wiley, New York, 1967.
2. BRENT, R. On the addition of binary numbers. *IEEE Trans. Comput. C-19*, 8 (1970), 758–759
3. KNUTH, D.E. *The Art of Computer Programming, Vol. 1.* Addison-Wesley, Reading, Mass., 1968
4. KNUTH, D.E. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, Reading, Mass, 1969.
5 KRAPCHENKO, V.M. Asymptotic estimation of addition time of a parallel adder *Syst. Theory Res. 19* (1970), 105–122 [*Probl Kibern. 19,* 107–122 (Russ.)].
6 OFMAN, YU. On the algorithmic complexity of discrete functions. *Sov Phys Dokl 7* (1963), 589–591
7. PATERSON, M S An introduction to Boolean function complexity. *Société Math de France Astérisque 38–39* 1976, 183–201 Also Tech. Rep STAN-CS-76-557, Computer Science Department, Stanford Univ, Stanford, Calif, August 1976
8. SAVAGE, J E. *The Complexity of Computing.* Wiley, New York, 1976
9 SCHONHAGE, A A lower bound for the length of addition chains. *Theor Comput. Sci 1* (1975), 1–12
10 TUNG, C Arithmetic In *Computer Science,* A F Cardenas, L. Presser, and M A Marin, Eds, Wiley-Interscience, New York, 1972

RECEIVED JULY 1978, REVISED NOVEMBER 1979; ACCEPTED DECEMBER 1979