

**NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:**  
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

NEW ALGORITHMS AND LOWER BOUNDS FOR THE  
PARALLEL EVALUATION OF CERTAIN  
RATIONAL EXPRESSIONS AND RECURRENCES

H. T. Kung  
Carnegie-Mellon University  
Pittsburgh, Pennsylvania

A preliminary version of this paper has been presented at the Sixth Annual  
ACM Symposium on Theory of Computing, Seattle, May, 1974.

Author's Address: Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Pa. 15213.

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.

KEYWORDS AND PHRASES: Parallel algorithms, lower bounds, parallel evaluation,  
rational expressions, recurrence problem.

CR CATEGORIES: 3.15, 5.10, 5.25.

## ABSTRACT

The parallel evaluation of rational expressions is considered. New algorithms which minimize the number of multiplication or division steps are given. They are faster than the usual algorithms when multiplication or division takes more time than addition or subtraction. It is shown, for example, that  $x^n$  can be evaluated in two steps of parallel division and  $\lceil \log_2 n \rceil$  steps of parallel addition, while the usual algorithm takes  $\lceil \log_2 n \rceil$  steps of parallel multiplication.

Lower bounds on the time required are obtained in terms of the degree of the expressions to be evaluated. From these bounds, the algorithms presented in the paper are shown to be asymptotically optimal. Moreover, it is shown that by using parallelism the evaluation of any first order rational recurrence of degree  $> 1$ , e.g.,  $y_{i+1} = \frac{1}{2}(y_i + \frac{a}{y_i})$ , and any nonlinear polynomial recurrence can be sped up at most by a constant factor, no matter how many processors are used and how large the size of the problem is.

## 1. INTRODUCTION

In this paper we consider the parallel evaluation of certain rational expressions. We assume that several processors which can perform four arithmetic operations, +, -,  $\times$ , /, are available, and that the time required for accessing data and communicating between processors can be ignored. This problem has been studied by many people. (See the surveys written by Brent [3] and Kuck [12].)

Almost all papers in this field assume that every arithmetic operation takes the same time. However, this assumption is false for two reasons. For many processors, floating number multiplication takes more time than addition. Furthermore, if we deal with expressions involving, for example, matrices or multiple-precision numbers then multiplication is likely more expensive than addition. (Here we interpret arithmetic operations as matrix or multiple-precision number operations.) In Section 3 of this paper, we assume that multiplication takes more time than addition. Hence, to get better algorithms, we should avoid using multiplications. We derive new algorithms for the parallel evaluations of  $x^n$ ,  $\{x^2, x^3, \dots, x^n\}$ ,  $\prod_{i=1}^n (x+a_i)$ ,  $\sum_{i=0}^n a_i x^i$ , etc., where the  $a_i$  are scalars. Each of the algorithms minimizes the time needed for the multiplications to within a constant and can be shown to be faster than the best previously known algorithm for large  $n$ . Moreover, all the algorithms, except the one associated with Theorem 3.4, have the following two characteristics:

1. To run the algorithms each processor is either masked or performing the same operation at any time. Hence the algorithm can be run on single-instruction stream-multiple-data stream (SIMD) machines (Flynn [4]) such as ILLIAC IV.

2. The algorithms require a very simple interconnection pattern.

All we need is a binary tree network between processors.

In Section 4 we prove lower bounds on the time needed for the parallel evaluation of certain rational expressions, under the assumption that all processors can perform different operations at any time. This assumption corresponds to multiple-instruction stream-multiple-data stream (MIMD) machines (Flynn [4]) such as C.mmp, the multi-mini-processor system at Carnegie-Mellon University (Wulf and Bell [19]). It is clear that lower bounds with respect to MIMD machines also hold with respect to SIMD machines. The lower bounds obtained in the paper imply that the algorithms introduced in Section 3 are asymptotically optimal with respect to MIMD machines, although most of these algorithms can be run on SIMD machines, as noted above.

Section 5 deals with the problem of the parallel evaluation of rational expressions defined by recurrences. We show that, by using parallelism, the evaluation of an expression defined by any first order rational recurrence of degree  $> 1$  or any nonlinear polynomial recurrence can be sped up at most by a constant factor, no matter how many processors are used. Consider, for example, the evaluation of the  $y_n$  defined by the recurrence,

$$y_{i+1} = \frac{1}{2}(y_i + \frac{a}{y_i}), \quad i=0,1,2,\dots,n-1,$$

which is the well-known recurrence for approximating  $\sqrt{a}$ . We show that for evaluating  $y_n$  any parallel algorithm using any number of processors cannot be essentially faster than the obvious sequential algorithm for any  $n$ . Thus the theory for nonlinear recurrences is completely different from the

theory for linear recurrences, where good speed-ups have been obtained (for example, Heller [5], Kogge [9], Kogge and Stone [10], Maruyama [13], Munro and Paterson [14] and Stone [15]).

In the next section, we give basic definitions and an abstract formulation of the general evaluation problem considered in the paper.

## 2. ABSTRACT FORMULATION AND DEFINITIONS

Let  $F$  be an algebraically closed field, e.g.,  $F$  is the field  $\mathbb{C}$  of complex numbers, and let  $x$  be an indeterminate over  $F$ .  $F[x]$  and  $F(x)$  denote the ring of polynomials and the field of rational expressions in  $x$  over  $F$ , respectively. Our problem is to evaluate a set of polynomials in  $F[x]$ ,  $\{f_1(x), f_2(x), \dots, f_m(x)\}$ , under the following assumptions:

1. By evaluating  $\{f_1(x), \dots, f_m(x)\}$  we mean computing the values of  $f_1(x), \dots, f_m(x)$  over  $F(x)$ , given  $F \cup \{x\}$ . The four binary operations,  $+$ ,  $-$ ,  $\times$ ,  $/$ , associated with the field  $F(x)$  are the ones we are allowed to use.
2. The elements in  $F$  are called scalars. A multiplication of two elements in  $F(x)$  is called a scalar multiplication if one of the two elements is a scalar; otherwise it is called a nonscalar multiplication. Scalar or nonscalar addition (subtraction) is similarly defined. A division whose dividend is a scalar is called a scalar division. Let  $M, M_s, A, A_s$  denote the time needed for one nonscalar multiplication, scalar multiplication, nonscalar addition (subtraction), scalar addition (subtraction), respectively. Let  $D, D_s$  denote the time needed for a division whose dividend is a nonscalar, scalar, respectively.

3. At any given time, up to  $k$  operations may be performed. This means that there are  $k$  processors which can perform the operations,  $+$ ,  $-$ ,  $\times$ ,  $/$ , at any time but some processors may be idle. If in some time interval all processors, except the ones masked, perform the same operation, say, addition, then we refer to that time interval as a parallel step of addition.

If the positive integer  $k$  in (3) is greater than one, we say  $\{f_1(x), \dots, f_m(x)\}$  is to be evaluated in parallel; while if  $k$  is equal to one, we say it is to be evaluated sequentially. We define  $T_k(f_1(x), \dots, f_m(x))$  to be the minimum time needed to evaluate  $\{f_1(x), \dots, f_m(x)\}$  with  $k$  processors.

To illustrate our notation given in (2), we consider an example. Let  $F = \mathbb{C}$  and let  $x$  be a  $\ell \times \ell$  matrix  $A$  whose entries are in  $\mathbb{C}$ . Suppose that we use an  $O(\ell^3)$  algorithm for matrix multiplication and inversion. (Here we interpret division as matrix inversion.) Then  $M = O(\ell^3)$ ,  $M_s = O(\ell^2)$ ,  $A = O(\ell^2)$ .  $A_s = O(\ell)$ ,  $D = O(\ell^3)$ ,  $D_s = O(\ell^3)$ .

3. NEW ALGORITHMS WHICH USE DIVISIONS FOR THE PARALLEL EVALUATION OF  $x^n$ ,  
 $\{x^2, x^3, \dots, x^n\}$ ,  $\prod_1^n (x+a_i)$ ,  $\sum_0^n a_i x^i$ , etc.

In this section we assume that  $M > A$ . We first consider a well known problem, that of evaluating  $x^n$ . Knuth [11, § 4.6.3] gives a rather detailed survey of sequential algorithms for solving this problem. It is known that there exists a sequential algorithm which takes time  $\left[ \log n + O\left(\frac{\log n}{\log \log n}\right) \right] M$ . (In this paper all logarithms are taken to base 2.) However, as pointed out in Borodin and Munro [1], it is easy to show the following:



Lemma 3.1.

If division is not used,  $\lceil \log n \rceil M$  is a lower bound on the time for the parallel evaluation of  $x^n$ , no matter how many processors are used.

Hence, if division is not used, any parallel algorithm cannot be essentially faster than the fastest sequential algorithm. In the proof of the following theorem we give an algorithm for the parallel evaluation of  $x^n$  which uses divisions and which takes time less than  $\lceil \log n \rceil M$  when  $n$  is large.

Theorem 3.1.

If  $k \geq n$ ,  $x^n$  can be evaluated in two steps of parallel division and  $\lceil \log n \rceil + 2$  steps of parallel addition. More precisely,

$$(3.1) \quad \underline{T_n(x^n)} \leq \lceil \log n \rceil A + 2(A_s + D_s).$$

Proof

We establish the theorem by exhibiting an algorithm.

Algorithm 3.1. [An algorithm for the parallel evaluation of  $x^n$ .]

1. Compute  $A_i = x - r_i$ ,  $i=1, \dots, n$ , in parallel, where the  $r_i$  are in  $F$  and are the  $n$  distinct zeros of  $x^n - r$  for any non-zero element  $r$  in  $F$ .
2. Compute  $B_i = s_i / A_i$ ,  $i=1, \dots, n$ , in parallel, where  $s_i = r_i / (nr)$ .
3. Compute  $C = \sum_{i=1}^n B_i$  in parallel.
4. Compute  $D = 1/C$ .
5. Compute  $E = D+r$ .

$$\begin{aligned} \text{Note that } C &= \sum_1^n B_i = \sum_1^n s_i/A_i = \sum_1^n r_i/[\text{nr}(x-r_i)] \\ &= P(x)/[\text{nr}(x^n-r)] \end{aligned}$$

where  $P(x) = \sum_{i=1}^n r_i \prod_{j \neq i} (x-r_j)$ . Evaluating the first derivative of  $x^n-r = \prod_{i=1}^n (x-r_i)$  at  $r_i$ , we have  $\text{nr}_i^{n-1} = \prod_{j \neq i} (r_i-r_j)$ . Thus  $P(r_i) = r_i \prod_{j \neq i} (r_i-r_j) = \text{nr}$ ,  $i=1, \dots, n$ . This implies that  $P(x) \equiv \text{nr}$ , since the degree of  $P(x)$  is  $n-1$ .

Hence  $C = 1/(x^n-r)$  and so  $E = D + r = 1/C + r = x^n$ . Therefore Algorithm 3.1 indeed evaluates  $x^n$ . Since the number of available processors is  $\geq n$ , steps 1, 2, 3, 4, 5 can be done in time  $A_s, D_s, \lceil \log n \rceil A, D_s, A_s$ , respectively. So Algorithm 3.1 takes time  $\lceil \log n \rceil A + 2(A_s + D_s)$ . ■

Note that  $\lceil \log n \rceil A + 2(A_s + D_s) < \lceil \log n \rceil M$  when  $\lceil \log n \rceil > 2(A_s + D_s)/(M-A)$ .

In fact,

$$\lim_{n \rightarrow \infty} \lceil \log n \rceil M / [\lceil \log n \rceil A + 2(A_s + D_s)] = M/A.$$

Hence we have sped up the evaluation of  $x^n$  by a factor  $M/A$  for large  $n$ .

#### Remarks on Algorithm 3.1.

1. The choice of  $r$  in step 1 depends on the application of the algorithm. For instance, if the algorithm is used to compute  $A^n$  for a real matrix  $A$  then the number  $r$  should be chosen such that  $A - r_i I$  is nonsingular for all  $i$ ; otherwise the algorithm would break down at step 2, where we have to compute  $s_i(A - r_i I)^{-1}$  for all  $i$ . (Note that for matrix computation, in the algorithm divisions should be interpreted as matrix inversions, and scalars,  $r_i, r$ , should be interpreted as  $r_i I, rI$ , respectively, where  $I$  is the identity matrix.)

2. Since the constants,  $r_i, s_i$ , are in  $F$  and it is assumed in Section 2 that elements in  $F$  are given as free, Theorem 3.1 does not count the time needed to compute  $r_i$  and  $s_i$ . In practice, these constants have to be either stored in a table or computed. (We find a similar situation in the fast Fourier transform where certain constants, i.e., powers of an  $n$ th root of unity, are needed.) Strictly speaking, the algorithm is really a form of "preconditioning". The same remark holds for algorithms below.
3. The algorithm raises  $x$  to the  $n$ th power without using any multiplications but with two divisions. This may be surprising to those who are dealing only with sequential algorithms. This again demonstrates that there exists intrinsic difference between sequential and parallel computation (see Stone [16] for other examples).

Using the same ideas, we can immediately obtain the following

Theorem 3.2.

Let  $a_1, \dots, a_n$  be  $n$  distinct elements in  $F$ . If  $k \geq n$ , then  $\prod_{i=1}^n (x+a_i)$  can

be evaluated in two steps of parallel division and  $\lceil \log n \rceil + 1$  steps of parallel addition. More precisely,

$$(3.2) \quad T_n \left( \prod_{i=1}^n (x+a_i) \right) \leq \lceil \log n \rceil A + A_s + 2D_s.$$

Proof

We establish the theorem by exhibiting an algorithm.

Algorithm 3.2. [An algorithm for the parallel evaluation of  $\prod_{i=1}^n (x+a_i)$ .]

1. Compute  $A_i = x + a_i$ ,  $i=1, \dots, n$ , in parallel.
2. Compute  $B_i = b_i/A_i$ ,  $i=1, \dots, n$ , in parallel, where  $b_i = \left[ \prod_{j \neq i} (a_j - a_i) \right]^{-1}$ .
3. Compute  $C = \sum_{i=1}^n B_i$  in parallel.
4. Compute  $D = 1/C$ .

Note that  $C = \sum_{i=1}^n B_i = \sum_{i=1}^n b_i/A_i = 1/\prod_{i=1}^n (x+a_i)$ . Hence the algorithm indeed evaluates  $\prod_{i=1}^n (x+a_i)$ . Since the algorithm clearly takes time  $[\log n]A + A_s + 2D_s$  with  $n$  processors, we have proven (3.2). ■

The obvious algorithm for the parallel evaluation of  $\prod_{i=1}^n (x+a_i)$  does the following:

1. Compute  $A_i = x+a_i$ ,  $i=1, \dots, n$ , in parallel.
2. Compute  $D = \prod_{i=1}^n A_i$  in parallel.

It takes time  $[\log n]M + A_s$ . Hence Algorithm 3.2 achieves a speedup factor  $M/A$  for large  $n$  without significantly complicating the algorithm. It is conceivable that in general a computer organization which is suitable for executing the obvious algorithm is also suitable for executing Algorithm 3.2.

It should be noted that Theorem 3.2 and Algorithm 3.2 can be extended to cover the general expression  $\prod_{i=1}^n (x+a_i)^{m_i}$  where the  $a_i$  are  $n$  distinct elements in  $F$  and the  $m_i$  are positive integers, since partial fraction expansions can still be used when factors are raised to powers greater than one. The extension is straightforward and will not be given in detail here.

Corollary 3.1.

If  $P(x)$  is the  $n$ th degree Chebyshev polynomial with respect to some interval, then

$$(3.3) \quad \underline{T_n(P(x)) \leq \lceil \log n \rceil A + A_s + 2D_s.}$$

Proof

Since the zeros of  $P(x)$  are distinct and are known analytically, the (3.3) follows from Theorem 3.2. ■

There are several potential applications of Algorithms 3.1 and 3.2. For example, by using Algorithms 3.1 and 3.2 we can compute  $A^n$  and  $P(A)$ , respectively, where  $A$  is a matrix and  $P(x)$  is some Chebyshev polynomial.  $A^n$  and  $P(A)^n$  can then be used to approximate the dominant eigenvectors of  $A$ . (See, for instance, Wilkinson [18, Chapter 9].)

Lemma 3.2.

If  $k \geq \frac{1}{2}n(n+1) - 1$ , then the set  $\{x^2, x^3, \dots, x^n\}$  can be evaluated in two steps of parallel division and  $\lceil \log n \rceil + 2$  steps of parallel addition. More precisely,

$$(3.4) \quad \underline{T_k(x^2, x^3, \dots, x^n) \leq \lceil \log n \rceil A + 2(A_s + D_s)}$$

provided  $k \geq \frac{1}{2}n(n+1) - 1$ .

Proof

We establish the lemma by exhibiting an algorithm.

Algorithm 3.3. [An algorithm for the parallel evaluation of  $\{x^2, \dots, x^n\}$  by using at least  $\frac{1}{2}n(n+1) - 1$  processors.]

1. Assign  $i$  processors for the evaluation of  $x^i$  for each  $i=2, \dots, n$ .  
Use Algorithm 3.1 to evaluate  $x^i$  for each  $i$ . Since  $k \geq \frac{1}{2}n(n+1) - 1$ ,  $x^2, \dots, x^n$  can be evaluated simultaneously.
2. Step 4 of Algorithm 3.1 will not be performed for the evaluation of  $x^2, \dots, x^{n-1}$  until the time when step 4 of Algorithm 3.1 is ready to be performed for the evaluation of  $x^n$ .

Clearly, the lemma can be proven from Algorithm 3.3. ■

Theorem 3.3.

If  $k \geq n$ , then the set  $\{x^2, x^3, \dots, x^n\}$  can be evaluated in five steps of parallel nonscalar multiplication or division and  $\lceil \log n \rceil + 5$  steps of parallel addition. More precisely,

$$(3.5) \quad \underline{T_n(x^2, x^3, \dots, x^n)} \leq \lceil \log n \rceil A + A + 4(A_s + D_s) + M.$$

Proof

We establish the theorem for the case  $n \geq 9$  by exhibiting an algorithm. Using the same ideas of the algorithm, the theorem can be easily proven for  $n \leq 8$ .

Algorithm 3.4. [An algorithm for the parallel evaluation of  $\{x^2, x^3, \dots, x^n\}$  by using  $n$  processors.]

1. Compute  $A_i = x^i$ ,  $i=2, \dots, m$  by Algorithm 3.3, where  $m = \lceil \sqrt{n} \rceil$ ;

2. Compute  $B_i = A_m^i$ ,  $i=2, \dots, m$  by Algorithm 3.3;
3. Compute  $C_{i,j} = B_i \cdot A_j$ ,  $i, j=1, \dots, m-1$ , in parallel, where  $A_1 = x$  and  $B_1 = A_m$ .

Note that  $C_{i,j} = A_m^i \cdot A_j = x^{im+j}$  and that  $\{x^2, \dots, x^n\} \subset \{B_m\} \cup \{C_{i,j} \mid i, j=1, \dots, m-1\}$ . Hence Algorithm 3.4 indeed evaluates  $\{x^2, \dots, x^n\}$ . Also note that since  $\frac{1}{2}m(m+1) - 1 \leq n$  for  $n \geq 9$ , there are enough processors to perform steps 1 and 2 by Algorithm 3.3. The total time needed for steps 1 and 2 is  $2[\lceil \log m \rceil A + 2(A_s + D_s)]$ . Since  $(m-1)^2 \leq n$ , step 3 can be done in time  $M$ . Therefore Algorithm 3.4 takes time  $\lceil \log n \rceil A + A + 4(A_s + D_s) + M$ . ■

The following corollary shows how the above results can be used to produce efficient parallel algorithms with small parallelism.

Corollary 3.2.

If  $n > k > 1$ , then  $x^n$  can be evaluated in  $6\ell$  steps of parallel non-scalar multiplication or division and  $(\lceil \log k \rceil + 5)\ell$  steps of parallel addition, where  $\ell = \left\lceil \frac{\log n}{\log k} \right\rceil$ . More precisely,

$$\underline{T_k(x^n) \leq \ell[\lceil \log k \rceil A + A + 4(A_s + D_s) + 2M],}$$

for  $n > k > 1$ .

Proof

We establish the corollary by exhibiting an algorithm.

Algorithm 3.5. [An algorithm for the parallel evaluation of  $x^n$  by using  $k$  processors, where  $n > k > 1$ .]

Since  $\ell = \left\lceil \frac{\log n}{\log k} \right\rceil$ ,  $n \leq k^\ell$ . We have the following two cases:

Case 1.  $n = k^\ell$ .

Let  $y_0 = x$ . For  $i=0, \dots, \ell-1$ ,

- 1.1. compute  $y_i^k$  by Algorithm 3.1;
- 1.2. set  $y_{i+1} \leftarrow y_i^k$ .

Clearly,  $y_\ell = x^n$ . By Theorem 3.1,

$$T_k(x^n) \leq \ell[\lceil \log k \rceil A + 2(A_s + D_s)].$$

Case 2.  $n < k^\ell$ .

Let  $n = \sum_{i=0}^{\ell-1} a_i k^i$ , where  $0 \leq a_i < k$ . The algorithm for Case 1 can be modified as follows:

Let  $y_0 = x$  and  $z_0 = 1$ . For  $i=0, \dots, \ell-1$ ,

- 1.1. compute  $\{y_i^2, y_i^3, \dots, y_i^{a_i k}\}$  by Algorithm 3.4;
- 1.2. compute  $z_{i+1} = z_i y_i^{a_i}$ ;
- 1.3. set  $y_{i+1} \leftarrow y_i^k$ .

It is straightforward to show that  $z_\ell = x^n$ . By Theorem 3.3, we have

$$T_k(x^n) \leq \ell[\lceil \log k \rceil A + A + 4(A_s + D_s) + 2M]. \quad \blacksquare$$

It is possible to slightly improve the bounds in Corollary 3.2 by using more complicated algorithms than Algorithm 3.5.



Corollary 3.3.

If  $k \geq n$ , then a general  $n$ th degree polynomial  $\sum_0^n a_i x^i$  can be evaluated

---

by one step of parallel scalar multiplication, five steps of parallel non-scalar multiplication or division and  $2\lceil \log n \rceil + 6$  steps of parallel addition. More precisely,

$$(3.6) \quad \underline{T_n(\sum_0^n a_i x^i) \leq (2\lceil \log n \rceil + 2) A + 4(A_s + D_s) + M + M_s.}$$

Proof

The theorem is proven by an algorithm which computes  $\{x^2, \dots, x^n\}$  in time  $\lceil \log n \rceil A + A + 4(A_s + D_s) + M$  by using Algorithm 3.4, then  $\{a_0, a_1 x, \dots, a_n x^n\}$  in one step of scalar multiplication and finally combine these in further  $\lceil \log n \rceil + 1$  steps of parallel addition. ■

Note that the dominant term of the upper bound in (3.6) is  $2\lceil \log n \rceil A$ , while all other upper bounds we have derived so far have the dominant term  $\lceil \log n \rceil A$  (see (3.1) ~ (3.5)). In the following theorem we show that the upper bound in (3.6) may be improved to have  $\lceil \log n \rceil A$  as the dominant term by using  $2n$  processors.

Theorem 3.4.

$$\underline{T_{2n}(\sum_0^n a_i x^i) \leq (\log n)A + O((\log n)^{\frac{1}{2}})M.}$$

Proof

We apply a recursive evaluation procedure due to Brent [2], Maruyama [13] and (independently) Munro and Paterson [14, Algorithm A]. The procedure

will not be described here. However, we note that the procedure requires  $x^{2^i}$  at time  $iA + \text{constant}$ , for  $i=1, \dots, \lfloor \log n \rfloor$ . We then assign  $n$  processors for the procedure and another  $n$  processors for the evaluation of  $x^{2^i}$  for all  $i$  by using Algorithm 3.1 for each  $i$ . Hence at time  $iA + \text{constant}$ ,  $x^{2^i}$  is always available. ■

#### 4. LOWER BOUNDS

In this section we assume that different processors may perform different operations at any time. We shall prove lower bounds under this general assumption. Let  $f(x)$  be a rational expression in  $F(x)$ . Define the degree of  $f(x)$  to be

$$\deg f = \max(\deg f_1, \deg f_2),$$

where  $f_1(x), f_2(x)$  are two relatively prime polynomials in  $F[x]$  such that  $f = f_1/f_2$ .

##### Lemma 4.1.

Let  $f(x), g(x) \in F(x)$  and  $h(x) = f(x) \text{ op } g(x)$  where  $\text{op} \in \{+, -, \times, /\}$ .

Then if  $\text{op}$  is a nonscalar addition, multiplication or division then

$\deg h \leq \deg f + \deg g$ , otherwise  $\deg h = \max(\deg f, \deg g)$ .

##### Proof

Assume that  $\text{op}$  is a nonscalar multiplication. Then

$$h = (f_1/f_2) \text{ op } (g_1/g_2) = (f_1 \cdot g_1)/(f_2 \cdot g_2),$$

and hence  $\deg h \leq \max(\deg f_1 + \deg g_1, \deg f_2 + \deg g_2) \leq \deg f + \deg g$ .

Since the proofs for other cases are similar, they will be omitted. ■

Theorem 4.1.

Let  $f(x) \in F(x)$  with  $\deg f = n$ . Then

$$\underline{T_k(f(x)) \geq \lceil \log n \rceil U \quad \forall k,}$$

where  $U = \min(A, M, D)$ .

Proof

The proof follows from a growth argument on degree. Consider an arbitrary algorithm for the parallel evaluation of  $f(x)$  by using arbitrary number of processors. Let  $R_i$  denote the set of rational expressions which can be created by the algorithms in time  $iU$ . It suffices to show by induction that elements in  $R_i$  have degrees at most  $2^i$ . Obviously, the statement holds for  $i = 1$ . Suppose that it holds for  $i \leq j$ . Let  $r_1 \in R_{j+1}$ . We want to prove  $\deg r_1 \leq 2^{j+1}$ . If  $r_1 \in R_j$  then  $\deg r_1 \leq 2^j < 2^{j+1}$ . We are done. Suppose that  $r_1 \notin R_j$ . Let us consider how  $r_1$  is computed from  $R_j$  by the algorithm. Since  $r_1$  is created by the algorithm,  $r_1$  is the result of a binary operation  $op_1$  of the algorithm with operands  $r_{1,1}$  and  $r_{1,2}$ . Similarly, for  $i=1,2$ , if  $r_{1,i} \notin R_j$ ,  $r_{1,i}$  is the result of another binary operation  $op_{1,i}$  of the algorithm with operands  $r_{1,i,1}$  and  $r_{1,i,2}$ . Hence  $r_1$  is associated with a binary tree whose internal nodes represent results of the binary operations and whose leaves represent the elements in  $R_j$  which are used for computing  $r_1$ . By the construction of the tree, the rational expressions associated with internal nodes are not in  $R_j$ . (It is clear that the tree is finite, since there is a positive lower bound on the time needed for every operation.) We note that if the binary operation associated with an internal node is a nonscalar addition, multiplication or division then the two successors of the node must be leaves. Hence along each path of the tree there

is at most one node with which a nonscalar addition, multiplication or division is associated. Then by Lemma 4.1 and the induction hypothesis one can easily show that  $\deg r_j \leq 2^{j+1}$ . The induction is complete. ■

By Theorem 4.1 and the results obtained in Section 3, we have the following

Corollary 4.1.

If  $M > A$  and  $D > A$ , then

$$\begin{array}{l}
 \lceil \log n \rceil A \leq \left\{ \begin{array}{l}
 \frac{T_n(x^n) \leq \lceil \log n \rceil A + 2(A_s + D_s)}{n} \\
 \frac{T_n(\prod_{i=1}^n (x+a_i)) \leq \lceil \log n \rceil A + A_s + 2D_s}{1} \\
 \frac{T_n(x^2, x^3, \dots, x^n) \leq \lceil \log n \rceil A + A + 4(A_s + D_s) + M}{1} \\
 \frac{T_{2n}(\sum_{i=0}^n a_i x^i) \leq (\log n)A + O((\log n)^{\frac{1}{2}})M, \text{ where}}{1}
 \end{array} \right. \\
 \underline{a_n \neq 0.}
 \end{array}$$

Hence the lower and upper bounds are asymptotically optimal as  $n \rightarrow \infty$ .

Suppose that we have a problem for which  $D, D_s, M$  are much greater than  $A$  or  $M_s$ . Hence we want to minimize the number of divisions and nonscalar multiplications. The following theorem gives a lower bound on the time needed for divisions and nonscalar multiplications.

Theorem 4.2.

Suppose that we do not count the time needed for addition, subtraction and scalar multiplication. Let  $f(x) \in F(x)$  with  $\deg f = n$ . Then

$$\underline{T_k(f(x)) \geq \left\lceil \frac{\log n}{\log(k+1)} \right\rceil V}$$

where  $V = \min(D, D_s, M)$ .

Proof

Consider an arbitrary algorithm for the parallel evaluation of  $f(x)$  by using  $k$  processors. Let  $R_i$  be the set of rational expressions in  $F(x)$  which can be evaluated in time  $iV$  by the algorithm. We shall show by induction that there exists a common denominator  $D_i$  for the elements in  $R_i$  such that  $\deg D_i \leq (k+1)^i$  and such that if  $r \in R_i$  then  $r = \bar{r}/D_i$  for some  $\bar{r} \in F[x]$  with  $\deg \bar{r} \leq (k+1)^i$ . The induction statement clearly holds for  $i = 1$ . Assume that it holds for  $i \leq j$ . Let  $r_1, \dots, r_\ell$ ,  $\ell \leq k$ , be the results immediately following from the nonscalar multiplications or divisions of the algorithm, which occur in the time interval  $(jV, (j+1)V]$ .

Then

$$R_{j+1} = \left\{ \sum_1^\ell u_i r_i + ur \mid u_i, u \in F \text{ and } r \in R_j \right\}.$$

Assume that  $r_i = s_i \text{ op}_i t_i$  where  $s_i, t_i \in R_j$  and  $\text{op}_i \in \{x, /\}$ . By the induction hypotheses,  $s_i = \bar{s}_i/D_j$  and  $t_i = \bar{t}_i/D_j$  where  $\bar{s}_i, \bar{t}_i \in F[x]$  and both have degree  $\leq (k+1)^j$ . Hence  $r_i = \bar{s}_i \bar{t}_i / D_j^2$  when  $\text{op}_i = x$  and  $r_i = \bar{s}_i / \bar{t}_i$  when  $\text{op}_i = /$ . Without loss of generality, assume that  $\text{op}_i = /$  for  $i \leq h \leq \ell$  and  $\text{op}_i = x$  for  $i > h$ . Define

$$D_{j+1} = \begin{cases} \bar{t}_1 \dots \bar{t}_h D_j & \text{if } h = \ell, \\ \bar{t}_1 \dots \bar{t}_h D_j^2 & \text{if } h < \ell. \end{cases}$$

It is easy to check that  $D_{j+1}$  is a common denominator for the elements in  $R_{j+1}$ , and that  $\deg D_{j+1} \leq (k+1)^{j+1}$ , since  $\deg \bar{t}_i \leq (k+1)^j$  and  $\deg D_j \leq (k+1)^j$ . Also, it is easy to show that if  $r \in R_{j+1}$  then  $r = \bar{r}/D_{j+1}$  for some  $\bar{r} \in F[x]$  with  $\deg \bar{r} \leq (k+1)^{j+1}$ . Therefore the induction is complete and hence we have proven the theorem. ■

Corollary 4.2.

Suppose that we do not count the time needed for addition, subtraction and scalar multiplication. If  $k \leq n$ , then

$$\frac{\lceil \frac{\log n}{\log(k+1)} \rceil}{V} \leq T_k(x^n) \leq \frac{\lceil \frac{\log n}{\log k} \rceil}{(4D_s + 2M)},$$

where  $V = \min(D, D_s, M)$ . Hence the bounds are within a constant factor of the best possible.

Proof

The result follows from Corollary 3.2 and Theorem 4.2. ■

5. RESULTS ON NONLINEAR RECURRENCE PROBLEMS

It frequently occurs in applied mathematics that the solution to some problem is given by a recurrence relation. Hence we often have to compute  $y_n$  from  $y_0, y_{-1}, \dots, y_{-m}$  where  $y_n$  is defined by  $y_{i+1} = \phi(y_i, \dots, y_{i-m})$  for some function  $\phi(x_1, \dots, x_{m+1})$ . It is natural to try to use parallel computation to speed up the process of computing  $y_n$ . Karp, Miller and Winograd [8] studied some general aspects of parallelism and recurrence. Recent work in this area includes, for example, Heller [5], Kogge [9], Kogge and Stone [10], Maruyama [13], Munro and Paterson [14] and Stone [15]. These works concentrate essentially on linear recurrence problems. In particular, Kogge [9] has given a unified treatment for general linear recurrence problems and has shown that for a general class of linear recurrence problems we can have the  $n/\log n$  speed-up ratio, which can be shown to be, in some sense, optimal. Therefore the linear recurrence problem is essentially settled. However, we do not know how to construct efficient parallel algorithms for

even very simple nonlinear recurrence problems. (Note that nonlinear recurrence problems occur in practice very often.) For example, it seems very difficult to use parallelism for the following nonlinear recurrence:

$$(5.1) \quad y_{i+1} = \frac{1}{2} \left( y_i + \frac{a}{y_i} \right),$$

which is the well-known recurrence for approximating  $\sqrt{a}$ . (The question of using parallelism for the recurrence problem (5.1) was asked by H. S. Stone [17].) In this section we shall show that any parallel algorithm using any number of processors cannot be essentially faster than the obvious sequential algorithm, for any first order rational recurrence of degree  $> 1$  like (5.1), and for any nonlinear polynomial recurrence problem like

$$(5.2) \quad y_{i+1} = 2y_i^2 y_{i-1} + 3y_{i-2}.$$

Lemma 5.1.

If  $\varphi(x), \psi(x) \in F(x)$ , then  $\deg(\varphi \circ \psi) = (\deg \varphi)(\deg \psi)$ . (Note that "o" stands for composition.)

Proof

Write  $\varphi = \varphi_1/\varphi_2$ , where  $\varphi_1, \varphi_2$  are two relatively prime polynomials in  $F[x]$ . We may assume that the leading coefficient of  $\varphi_2$  is unity. We write  $\varphi_1(x) = a(x-a_1)^{m_1} \dots (x-a_h)^{m_h}$  and  $\varphi_2(x) = (x-b_1)^{n_1} \dots (x-b_\ell)^{n_\ell}$ , where the  $a$  is in  $F$ , the  $a_i$  are distinct elements in  $F$ , the  $b_i$  are distinct elements in  $F$  and the  $m_i, n_i$  are positive integers. Clearly,  $\deg \varphi_1 = \sum m_i$  and  $\deg \varphi_2 = \sum n_i$ . Since  $\varphi_1$  and  $\varphi_2$  are relatively prime, we have  $a_i \neq b_j, \forall i, j$ . Let  $\psi_1$  and  $\psi_2$  be two relatively prime polynomials such that  $\psi = \psi_1/\psi_2$ . Note that

$$\begin{aligned}
 (\varphi \circ \psi)(x) &= \frac{a(\psi(x)-a_1)^{m_1} \dots (\psi(x)-a_h)^{m_h}}{(\psi(x)-b_1)^{n_1} \dots (\psi(x)-b_\ell)^{n_\ell}} \\
 (5.3) &= \frac{a(\psi_1(x)-a_1\psi_2(x))^{m_1} \dots (\psi_1(x)-a_h\psi_2(x))^{m_h}}{(\psi_1(x)-b_1\psi_2(x))^{n_1} \dots (\psi_1(x)-b_\ell\psi_2(x))^{n_\ell}} \\
 &\quad \cdot \psi_2(x)^{\sum n_i - \sum m_i}.
 \end{aligned}$$

Claim that  $\psi_1(x) - a_i\psi_2(x)$  and  $\psi_1(x) - b_j\psi_2(x)$  are relatively prime for all  $i, j$ . We prove this by contradiction. Assume that there exists  $h(x) \in F[x]$  with  $\deg h \geq 1$  such that  $\psi_1 - a_i\psi_2 = h_1h$  and  $\psi_1 - b_j\psi_2 = h_2h$  where the  $h_1, h_2 \in F[x]$ . These imply that  $\psi_2 = [(h_1 - h_2)/(b_j - a_i)]h$  and  $\psi_1 = [h_1 + a_i(h_1 - h_2)/(b_j - a_i)]h$ . Hence  $h$  is a common divisor for  $\psi_1$  and  $\psi_2$ . This is a contradiction. Similarly, we can prove that there are no nontrivial common divisors between  $\psi_2(x)$  and  $\psi_1(x) - a_i\psi_2(x)$  and between  $\psi_2(x)$  and  $\psi_1(x) - b_j\psi_2(x)$ . Therefore, from (5.3), one can compute the degree of  $\varphi \circ \psi$  as follows: Assume that  $\deg \varphi = \deg \varphi_1 \geq \deg \varphi_2$  and  $\deg \psi = \deg \psi_1 \geq \deg \psi_2$ . (The proofs for the other cases are similar and will be omitted.) The numerator of (5.3) has degree  $(\sum m_i)\deg \psi_1 = (\deg \varphi)(\deg \psi)$ . The denominator of (5.3) has degree  $(\sum n_i)\deg \psi_1 + (\sum n_i - \sum m_i)\deg \psi_2 = (\deg \varphi_2)(\deg \psi_1) + (\deg \varphi_1 - \deg \varphi_2)(\deg \psi_2)$ , which is  $\leq (\deg \varphi)(\deg \psi)$ . Hence  $\deg(\varphi \circ \psi) = (\deg \varphi)(\deg \psi)$ . ■

Theorem 5.1.

Let  $y_n$  be defined by  $y_{i+1} = \varphi(y_i)$  where  $\varphi(x) \in F(x)$  with  $\deg \varphi = d$ . Then



$$(5.4) \quad \underline{T_k(y_n)} \geq \lceil n \log d \rceil U, \quad \forall k$$

where  $U = \min(A, M, D)$ .

Proof

Let  $y_0 = x$ . Then  $y_n = \phi^n(x)$  where  $\phi$  is the  $n$  times self-composition of  $\phi$ . Then by Lemma 5.1,  $\deg \phi^n = (\deg \phi)^n = d^n$ . The theorem follows from Theorem 4.1. ■

Under the assumptions of Theorem 5.1,  $y_n$  clearly can be computed sequentially in time  $n \cdot T_1(\phi(x))$ . If  $\deg \phi = d > 1$ , then by (5.4) we have

$$\frac{T_1(y_n)}{T_k(y_n)} \leq \frac{T_1(\phi(x))}{(\log d)U} = \text{constant}, \quad \forall n, \forall k.$$

Hence we have the following

Corollary 5.1.

By using parallelism the evaluation of an expression defined by any first order rational recurrence with degree  $> 1$  can be sped up at most by a constant factor.

Consider, for example, the recurrence problem (5.1). Assume that we work with real numbers and that every arithmetic operation takes the same time  $U$ . Then to evaluate  $y_n$  the obvious sequential algorithm takes time  $3nU$ , while by Theorem 5.1 any parallel algorithm takes time at least  $nU$ . Hence by using parallelism the evaluation of  $y_n$  can be sped up at most by a factor of 3, for all  $n$ . This is completely different from the evaluation of linear recurrence where  $n/\log n$  speed-ups can be obtained.

Now we consider higher order recurrences, i.e.  $y_{i+1} = \phi(y_i, y_{i-1}, \dots, y_{i-m})$  for  $m > 0$ . Suppose that  $\phi$  is a multivariate polynomial of degree  $> 1$ . Let

$y_0 = y_{-1} = \dots = y_{-m} = x$ . Then  $y_1, y_2, \dots, y_n$  are rational expressions in  $x$ . It is very easy to see that there exists a constant  $\theta > 1$  such that the degree of  $y_i$  in  $x$  is  $\geq \theta^i$  for all  $i$ . For example, consider the third order recurrence (5.2). Let  $a_i$  be a lower bound on the degree of  $y_i$  in  $x$ . Then by (5.2) we have  $a_{i+1} \geq 2a_i + a_{i-1}$ . By a standard technique on difference equations, we know  $a_i$  can be chosen as  $\theta^i$  where  $\theta^2 = 2\theta + 1$  and hence  $\theta > 1$ .

Since the degree of  $y_n$  in  $x$  is  $\geq \theta^n$ , by Theorem 5.1 we have

$$T_k(y_n) \geq [n \log \theta]U$$

where  $U = \min(A, M, D)$ . Let  $T_1(\varphi)$  denote the time for evaluating  $\varphi(x_1, x_2, \dots, x_{i+d+1})$  sequentially. Then  $T_1(y_n) \leq nT_1(\varphi)$  and hence

$$\frac{T_1(y_n)}{T_k(y_n)} \leq \frac{T_1(\varphi)}{(\log \theta)U} = \text{constant}, \forall n, \forall k.$$

Hence, we have the following

Corollary 5.2.

By using parallelism the evaluation of an expression defined by any non-linear polynomial recurrence can be sped up at most by a constant factor.

6. SUMMARY AND CONCLUSIONS

It is convenient to think that the paper consists of two parts. In the first part, we have given a general technique to construct parallel algorithms which minimize the number of multiplication or division steps. This technique is useful when multiplication or division is expensive. Some rather surprising algorithms are derived. For example, Algorithm 3.1 evaluates powers of  $x$  using additions instead of multiplications. This demonstrates the intrinsic difference between sequential and parallel computation.

In the second part of the paper, we have shown (Theorems 4.1 and 4.2) lower bounds on the time to evaluate rational expressions. The lower bounds are asymptotically close to the upper bounds established by the algorithms in the first part of the paper. Using the lower bound results, we have shown that by using parallelism the evaluation of an expression defined by any first order rational recurrence of degree  $> 1$  or any nonlinear polynomial recurrence can be sped up at most by a constant factor, no matter how many processors are used and how large the size of the problem is. This is probably the first and may be the only known example of a problem which cannot be essentially sped up.

#### ACKNOWLEDGMENTS

I want to thank Professor J. F. Traub for his helpful comments on this paper.

REFERENCES

(Note: References [6,7] are not cited in the text.)

- [1] Borodin, A. B. and Munro, I. Notes on Efficient and Optimal Algorithms, University of Toronto and University of Waterloo, 1972.
- [2] Brent, R.P. "On the Addition of Binary Numbers," IEEE Trans. Comput. C-19 (1970), 758-759.
- [3] Brent, R. P. "The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time," in Complexity of Sequential and Parallel Numerical Algorithms, (J. F. Traub ed.), Academic Press, New York, 1973, 83-102.
- [4] Flynn, M. J. "Very High-Speed Computing Systems," Proc. IEEE, 54 (1966), 1901-1909.
- [5] Heller, D. "A Determinant Theorem with Applications to Parallel Algorithms," SIAM J. Numer. Anal. 11 (1974), 559-568.
- [6] Hyafil, L. and Kung, H. T. "The Complexity of Parallel Evaluation of Linear Recurrences," Proc. 7th Annual ACM Symposium on Theory of Computing, 1975, 12-22.
- [7] Hyafil, L. and Kung, H. T. "Bounds on the Speed-Ups of Parallel Evaluation of Recurrences," Second USA-Japan Computer Conference Proceedings, 1975.
- [8] Karp, R., Miller, R. and Winograd, S. "The Organization of Computations for Uniform Recurrence Equations," J.ACM 14 (1967), 563-590.
- [9] Kogge, P. M. "Parallel Solution of Recurrence Problems," IBM J. Res. Develop. 18 (1974), 138-148.
- [10] Kogge, P. M. and Stone, H. S. "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Trans. Comput. C-22 (1973), 786-793.
- [11] Knuth, D. E. The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1969.
- [12] Kuck, D. J. "Multioperation Machine Computational Complexity, in Complexity of Sequential and Parallel Numerical Algorithms (J. F. Traub ed.), Academic Press, New York, 1973, 17-46.
- [13] Maruyama, K. "On the Parallel Evaluation of Polynomials," IEEE Trans. Comput. C-22 (1973), 2-5.
- [14] Munro, I. and Paterson, M. "Optimal Algorithms for Parallel Polynomial Evaluation," JCSS 7 (1973), 189-198.

- [15] Stone, H. S. "An Efficient Parallel Algorithms for the Solution of a Tridiagonal System of Equations," J.ACM 20 (1973), 27-38.
- [16] Stone, H. S. "Problems of Parallel Computation," in Complexity of Sequential and Parallel Numerical Algorithms (J. F. Traub ed.), Academic Press, New York, 1973, 1-16.
- [17] Stone, H. S. Private Communication, 1973.
- [18] Wilkinson, J. H. The Algebraic Eigenvalue Problem, Oxford University Press (Clarendon), London and New York, 1965.
- [19] Wulf, W. A. and Bell, C. G. "C.mmp -- A Multi-Mini-Processor," AFIPS Conference Proc., Vol. 41, Part II, FJCC 1972, 765-777.

## DOCUMENT CONTROL DATA - R &amp; D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Department of Computer Science; Carnegie-Mellon U. Pittsburgh, Pennsylvania 15213		2a. REPORT SECURITY CLASSIFICATION unclassified	
		2b. GROUP	
3. REPORT TITLE NEW ALGORITHMS AND LOWER BOUNDS FOR THE PARALLEL EVALUATION OF CERTAIN RATIONAL EXPRESSIONS			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates)			
5. AUTHOR(S) (First name, middle initial, last name) H. T. Kung			
6. REPORT DATE February, 1974		7a. TOTAL NO. OF PAGES 12	7b. NO. OF REFS 16
8a. CONTRACT OR GRANT NO. N00014-67-A-0314-0010, NR 044-422		9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.			
c.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.			
10. DISTRIBUTION STATEMENT approved for public release; distribution unlimited			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Mathematics Program Office of Naval Research Arlington, Virginia 22217	
13. ABSTRACT This paper presents new algorithms for the parallel evaluation of certain polynomial expressions. In particular, for the parallel evaluation of $x^n$ , we introduce an algorithm which takes two steps of <u>parallel division</u> and $\lceil \log_2 n \rceil$ steps of parallel addition, while the usual algorithm takes $\lceil \log_2 n \rceil$ steps of parallel multiplication. Hence our algorithm is faster than the usual algorithm when multiplication takes more time than addition. Similar algorithms for the evaluation of other polynomial expressions are also introduced. Lower bounds on the time needed for the parallel evaluation of rational expressions are given. All the algorithms presented in the paper are shown to be asymptotically optimal. Moreover, we prove that by using parallelism the evaluation of any first order rational recurrence, e. g., $y_{i+1} = \frac{1}{2}(y_i + \frac{a}{y_i})$ , and any non-linear polynomial recurrence can be sped up at most by a constant factor, no matter how many processors are used.			

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT