# NEW ALGORITHMS AND LOWER BOUNDS FOR THE PARALLEL EVALUATION OF CERTAIN RATIONAL EXPRESSIONS[*]

H. T. Kung
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

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 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. 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., $x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i})$, and any non-linear polynomial recurrence can be sped up at most by a constant factor, no matter how many processors are used.

## 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, +, -, ×, /, 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 [73] and Kuck [73].) 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 of course more expensive than addition. (Here we interpret arithmetic operations as matrix or multiple-precision number operations.) In 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 $x^n$, $\{x^2, x^3, \ldots, x^n\}$, $\prod_1^n (x+a_i)$, $\sum_0^n a_i x^i$, 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

[*]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.

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 (SISM) machines (Flynn [66]), which include ILLIAC IV, CDC STAR-100, Texas Instruments ASC, etc.

2) The algorithms require a very simple interconnection pattern. All we need is a binary tree network between processors. Hence, for most machine organizations, we should not expect any significant delay caused by communication between processors.

We also 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 [66]) such as C.mmp, the multi-mini-processor system currently under construction at Carnegie-Mellon University (Wulf and Bell [72]). It is clear that optimal algorithms with respect to MIMD machines must be also optimal with respect to SIMD machines. The lower bounds obtained in the paper imply that the algorithms presented in the paper are asymptotically optimal with respect to MIMD machines, although most of these algorithms can be run on SIMD machines, as noted above. Furthermore, these lower bounds imply that, by using parallelism, the evaluation of an expression defined by any first order rational recurrence or any <u>non-linear</u> 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,

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

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. Thus the theory for non-linear recurrence is completely different from the theory for linear recurrences, where good speed-ups have been obtained (for example, Heller [73], Kogge [72], Kogge and Stone [72], Maruyama [73], Munro and Paterson [73] and Stone [73a]).

Suppose that we have a problem for which multiplication is much more expensive than addition. We want to minimize the number of non-scalar multiplications and divisions. Lower bounds on the time needed for the multiplications and divisions are also derived.

In the next section, we give basic definitions and an abstract formulation of our problem. In Section 3 we derive algorithms for the parallel evaluation of various expressions. Lower bound results are given in Section 4. The final section deals with results on non-linear recurrences.

## 2. ABSTRACT FORMULATION AND DEFINITIONS

Let F be a commutative and algebraically closed field, e.g., F is the field $\mathbb{C}$ of complex numbers. Let F[x] and F(x) be the ring of polynomials and the field of rational expressions in x over F, respectively. Our task 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 $\{f_1(x),\ldots,f_m(x)\}$ we mean computing the values of $f_1(x),\ldots,f_m(x)$ from $F \cup \{x\}$, inside the field $F(x)$. 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 non-scalar multiplication. Scalar or non-scalar addition (subtraction) is similarly defined. A division whose dividend is a non-scalar is called a non-scalar division. Let $M$, $M_s$, $A$, $A_s$ denote the time needed for one non-scalar multiplication, scalar multiplication, non-scalar addition (subtraction), scalar addition (subtraction), respectively. Let $D, D_s$ denote the time needed for a division whose dividend is a non-scalar, scalar, respectively. Assume that $M > A$.

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 a given 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 $\{f_1(x),\ldots,f_m(x)\}$ is to be evaluated in parallel, while if $k$ is equal to one, we say $\{f_1(x),\ldots,f_m(x)\}$ is to be evaluated sequentially. We define $T_k(f_1(x),\ldots,f_m(x))$ to be the minimum time needed to evaluate $\{f_1(x),\ldots,f_m(x)\}$ 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 $x^n$, $\{x^2, x^3, \ldots, x^n\}$, $\prod_1^n (x+a_i)$, $\sum_0^n a_i x^i$, etc.

We first consider a well known problem, that of evaluating $x^n$. Knuth [69, § 4.6.3] gives a rather detailed survey of the sequential algorithms for 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, it is easy to show the following (see, for instance, Borodin and Munro [72]):

Fact 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 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 $\lceil \log n \rceil$ 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,

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

Proof

We establish the theorem by exhibiting an algorithm.

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

1) Compute $A_i = x - r_i$, $i=1,\ldots,n$, 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 $B_i = s_i/A_i$, $i=1,\ldots,n$, in parallel, where $s_i = [\prod_{j \neq i} (r_i - r_j)]^{-1}$;

3) Compute $C = \sum_1^n B_i$ in parallel;

4) Compute $D = 1/C$;

5) Compute $E = D + r$.

It is easy to check that $E = x^n$. Hence Algorithm 3.1 indeed evaluates $x^n$. Suppose that the number of processors $k \geq n$. Then clearly 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. Therefore 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,

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

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 non-singular 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) The algorithm raises x to the nth 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 the intrinsic difference between sequential computation and parallel computation (Stone [73b]).

Using these same ideas, we can immediately obtain the following

Theorem 3.2.

Let $a_1, \ldots, a_n$ be n distinct elements in F. If $k \geq n$, then $\prod_1^n (x + a_i)$ can be evaluated in two steps of parallel division and $\lceil \log n \rceil + 2$ steps of parallel addition. More precisely,

$$(3.2) \quad T_n\left(\prod_1^n (x + a_i)\right) \leq \lceil \log n \rceil A + 2(A_s + D_s).$$

Proof

We establish the lemma by exhibiting an algorithm.

Algorithm 3.2. [An algorithm for the parallel evaluation of $\prod_1^n (x + a_i)$.]

1) Compute $A_i = x + a_i$, $i=1,\ldots,n$, in parallel;

2) Compute $B_i = b_i/A_i$, $i=1,\ldots,n$, in parallel, where $b_i = [\prod_{j\neq i} (a_j+a_i)]^{-1}$;

3) Compute $C = \sum_1^n B_i$ in parallel;

4) Compute $D = 1/C$;

■

Corollary 3.1.

<u>If $P(x)$ is the nth degree Chebyshev polynomial with respect to some interval, then</u>

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

Proof

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

It is clear that after some obvious modifications of Algorithm 3.2, Theorem 3.2 can be extended to cover the general expression $\prod_1^n (x+a_i)^{m_i}$ where the $a_i$ are distinct and the $m_i$ are positive integers. Since it is straightforward, we will not give the details here.

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 [65, Chapter 9].) However, these applications do not fit the topic of this paper. They will be reported in another paper.

Lemma 3.1.

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

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

<u>provided $k \ge \frac{1}{2}n(n+1) - 1$.</u>

Proof

We establish the lemma by exhibiting an algorithm

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

1) Assign $i$ processors for the evaluation of $x^i$ for each $i=2,\ldots,n$. Use Algorithm 3.1 to evaluate $x^i$ for each $i$. Since $k \ge \frac{1}{2}n(n+1) - 1$, $x^2,\ldots,x^n$ can be evaluated simultaneously.

2) Step 4 of Algorithm 3.1 will not be performed for the evaluation of $x^2,\ldots,x^{n-1}$ until the time when step 4 of Algorithm 3.1 is ready to be performed for the evaluation of $x^n$.

Clearly, the lemma follows from Algorithm 3.3. ■

Theorem 3.3.

<u>If $k \ge n$, then the set $\{x^2,x^3,\ldots,x^n\}$ can be evaluated in five steps of parallel non-scalar multiplication or division and $\lceil\log n\rceil + 4$ steps of parallel addition.</u> <u>More precisely,</u>

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

Proof

We establish the theorem for the case $n \ge 9$

by exhibiting an algorithm. Using the same ideas of the algorithm, the theorem can be easily proven for $n \le 8$.

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

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

2) Compute $B_i = A_m^i$, $i=2,\ldots,m$ by Algorithm 3.3;

3) Compute $C_{i,j} = B_i \cdot A_j$, $i,j=1,\ldots,m-1$, in parallel, where $A_1 = x$ and $B_1 = A_m$.

It is easily seen that $C_{i,j} = x^{im+j}$ and that $\{x^2,\ldots,x^n\} \subset \{B_m\} \cup \{C_{i,j} | i,j=1,\ldots,m-1\}$. Hence Algorithm 3.4 indeed evaluates $\{x^2,\ldots,x^n\}$. Note that since $\frac{1}{2}m(m+1) - 1 \le n$ for $n \ge 9$, there are enough processors to perform Algorithm 3.3 at steps 1 and 2 . The total time needed for steps 1 and 2 is $2[\lceil\log m\rceil A + 2(A_s+D_s)]$. Since $(m-1)^2 \le n$, step 3 can be done in time $M$. Therefore Algorithm 3.4 takes time $\lceil\log n\rceil A + 4(A_s+D_s)+M$. ■

Corollary 3.2.

If $k < n$, then $x^n$ can be evaluated in $5\ell+1$ steps of parallel non-scalar multiplication or division and $(\lceil\log k\rceil + 4)\ell$ steps of parallel addition, where $\ell = \left\lceil\frac{\log n}{\log k}\right\rceil$. More precisely,

$$T_k(x^n) \le \ell[\lceil\log k\rceil A + 4(A_s+D_s) + M] + M,$$

for $k < n$.

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 $k<n$.]

1) For $i=0,\ldots,\ell-1$, let $y_i = x^{k^i}$ and evaluate $\{y_i^2, y_i^3, \ldots, y_i^k\}$ by Algorithm 3.4;

2) Compute $A = y_{\ell-1}^{a_{\ell-1}} y_{\ell-2}^{a_{\ell-2}} \cdots y_0^{a_0}$ where the $a_i$ are non negative integers such that $0 \le a_i < k$ and $n = \sum_0^{\ell-1} a_i k^i$. [Note that if $n = k^\ell$ then $x^n = y_{\ell-1}^k$ and hence step 2 need not be performed.] Clearly, $A = x^n$.

Observe that in the time when step 1 completes the task for $i = j$, $y_0^{a_0} \cdots y_j^{a_{j-1}}$ can also be computed, $j=1,\ldots,\ell-1$. ■

Corollary 3.3.

If $k \ge n$, then a general nth 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 $2\lceil\log n\rceil + 5$ steps of parallel addition. More precisely,

(3.6) $T_n(\sum_0^n a_i x^i) \le (2\lceil\log n\rceil+1)A+4(A_s+D_s)+M+M_s$.

Proof

The theorem is proven by an algorithm which computes $\{x^2,\ldots,x^n\}$ in time $\lceil\log n\rceil A + 4(A_s+D_s)+M$ by using Algorithm 3.4, then $\{a_0, a_1x, \ldots, a_nx^n\}$ in one step of scalar multiplication and finally combine these in a 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.

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

Proof

We apply a recursive evaluation procedure due to Maruyama [73] and (independently) Munro and Paterson [73, Algorithm A]. The procedure will not be described here. However, we note that the procedure requires $x^{2^i}$ at time iA + constant, for $i=1,\ldots,\lfloor \log n \rfloor$. 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 + constant, $x^{2^i}$ is always available. ■

## 4. LOWER BOUNDS

In this section we shall assume the same notation as in the previous sections, except that now x may also stand for a set of indeterminates $\{x_1, x_2, \ldots, x_r\}$ over F. Also recall that we allow different processors perform different operations at any time. Let f(x) be a rational expression in F(x). Define the degree of f(x) to be

$$\deg f = \max(\deg g, \deg h)$$

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

Lemma 4.1.

Let $f(x), g(x) \in F(x)$ and $h(x) = f(x) \text{ op } g(x)$ where op $\in \{+,-,\times,/\}$. Then if op is a non-scalar addition, multiplication or division then $\deg h \le (\deg f)(\deg g)$, otherwise $\deg h = \max(\deg f, \deg g)$.

Proof

Trivial. ■

Theorem 4.1.

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

$$T_k(f(x)) \ge \lceil \log n \rceil U, \ \forall k,$$

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

Proof

The proof follows from a growth argument. 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 $i \le j$. Let $r_1 \in R_{j+1}$. We want to prove $\deg r_1 \le 2^{j+1}$. If $r_1 \in R_j$ then $\deg r_1 \le 2^j < 2^{j+1}$. 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 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 the 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 a node is a non-scalar 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 non-scalar addition, multiplication or division is associated. Then by Lemma 4.1 and the induction hypothesis one can easily show that $\deg r_1 \le 2^{i+1}$. 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

$$\lceil \log n \rceil A \le \begin{cases} T_n(x^n) \le \lceil \log n \rceil A + 2(A_s+D_s) \\ T_n(\prod_1^n (x+a_i)) \le \lceil \log n \rceil A + 2(A_s+D_s) \\ T_n(x^2, x^3, \ldots, x^n) \le \lceil \log n \rceil A + 4(A_s+D_s)+M \\ T_{2n}(\sum_0^n a_i x^i) \le (\log n)A + O((\log n)^{\frac{1}{2}})M, \text{where} \\ \quad a_n \ne 0. \end{cases}$$

Hence the algorithms corresponding to the upper bounds are asymptotically optimal as $n \to \infty$.

Suppose that we have a problem for which $D \gg A$, $M \gg A$ and $D_s \gg A$. Hence we want to minimize the number of non-scalar multiplications and divisions. The following theorem gives a lower bound on the time needed for the non-scalar multiplications and divisions.

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, if $k \le n$,

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

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

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 $\deg D_i \le (k+1)^i$ and such that if $r \in R_i$ and $r = \bar{r}/D_i$ where $\bar{r} \in F[x]$, then $\deg \bar{r} \le (k+1)^i$. The induction statement clearly holds for $i = 1$. Assume that it holds for $i \le j$. Let $r_1, \ldots, r_\ell$, $\ell \le k$, be the results immediately following from the non-scalar multiplications or divisions of the algorithm, which occur in the time interval $(jV, (j+1)V]$. Then

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

Assume that $r_i = s_i \text{ op}_i\, t_i$ where $s_i, t_i \in R_j$ and $\text{op}_i \in \{\times, /\}$. 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 $\le (k+1)^j$. Hence $r_i = \bar{s}_i \bar{t}_i / D_j^2$ when $\text{op}_i = \times$ and $r_i = \bar{s}_i/\bar{t}_i$ when $\text{op}_i = /$. Without loss of generality, assume that $\text{op}_i = /$ for $i \le h \le \ell$ and $\text{op}_i = \times$ for $i > h$. Define

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

It is easy to see that $D_{j+1}$ is a common denominator for $R_{j+1}$ by (4.1), and that $\deg D_{j+1} \le (k+1)^{j+1}$, since $\deg t_i \le (k+1)^j$ and $\deg D_j \le (k+1)^j$. Also, it is easy to show that if $r \in R_{j+1}$ and $r = \bar{r}/D_{j+1}$ with $\bar{r} \in F[x]$ then $\deg \bar{r} \le (k+1)^{j+1}$. 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 $k \le n$, then

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

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

Proof

The proof follows from Corollary 3.2 and Theorem 4.2. ■

## 5. RESULTS ON NON-LINEAR 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 $y_0, y_{-1}, \ldots, y_{-d}$ where $y_n$ is defined by $y_{i+1} = \varphi(y_i, \ldots, y_{i-d})$ for some function $\varphi(x_1, \ldots, x_{d+1})$. It is natural to try to use parallel computation to speed up the process of computing $y_n$. Karp, Miller and Winograd [67] studied some general aspects of parallelism and recurrence. Recent work in this area includes, for example, Heller [73], Kogge [72], Kogge and Stone [72], Maruyama [73], Munro and Paterson [73] and Stone [73a]. These works concentrate essentially on linear recurrence problems. In particular, Kogge [72] has given a unified treatment for general linear recurrence problems and has shown for a very general class of linear recurrence problems that 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 non-linear recurrence problems. (Note that non-linear recurrence problems occur in practice very often.) For example, it seems very difficult to use parallelism for the following non-linear recurrence equations:

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

which is the well-known recurrence for approximating $\sqrt{a}$. (The question of using parallelism for the recurrence problem (5.1) was asked by Professor H. S. Stone [73c].) 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 problem like (2.1), and for any non-linear 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 $\deg(\varphi \circ \psi) = (\deg \varphi)\cdot(\deg \psi)$.

Proof

Write $\varphi = \varphi_1/\varphi_2$, where $\varphi_1$, $\varphi_2$ are two relatively prime polynomials in $F[x]$. Assume that the leading coefficient of $\varphi_2$ is unity. Then write $\varphi_1(x) = a(x-a_1)^{m_1} \ldots (x-a_h)^{m_h}$ and $\varphi_2(x) = (x-b_1)^{n_1} \ldots (x-b_\ell)^{n_\ell}$, 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 non-negative integers. Clearly, $\deg \varphi_1 = \Sigma m_i$ and $\deg \varphi_2 = \Sigma n_i$. Since $\varphi_1$ and $\varphi_2$ are relatively prime, we have $a_i \ne b_j$, $\forall i,j$. Let $\psi_1$ and $\psi_2$ be two relatively prime polynomials such that $\psi = \psi_1/\psi_2$. Note that

$$\varphi \circ \psi(x) = a\frac{(\psi(x)-a_1)^{m_1} \ldots (\psi(x)-a_h)^{m_h}}{(\psi(x)-b_1)^{n_1} \ldots (\psi(x)-b_\ell)^{n_\ell}}$$

(5.3) $$= a\cdot\frac{(\psi_1(x)-a_1\psi_2(x))^{m_1} \ldots (\psi_1(x)-a_h\psi_2(x))^{m_h}}{(\psi_1(x)-b_1\psi_2(x))^{n_1} \ldots (\psi_1(x)-b_\ell\psi_2(x))^{n_\ell}} \cdot \psi_2(x)^{\Sigma n_i - \Sigma m_i}.$$

We 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 non-trivial 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 easily check that $\deg(\varphi\circ\psi) = (\deg \varphi)\cdot(\deg \psi)$. ■

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

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

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

Proof

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

Under the assumptions of Theorem 5.1, $y_n$ clearly can be computed sequentially in time $nT_1(\varphi)$ . $\varphi$ is called a rational recurrence if $d > 1$. In this case, we have

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

Hence, we have the following

Corollary 5.1.

By using parallelism the evaluation of an expression defined by any first order rational recurrence 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. $y_{i+1} = \varphi(y_i,y_{i-1},\ldots,y_{i-m})$ for $m > 0$. Suppose that $\varphi$ is a multivariate polynomial of degree $> 1$. Let $y_0 = y_{-1} = \ldots = y_{-m} = x$. Then $y_1,y_2,\ldots,y_n$ 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

$$T_k(y_n) \geq \lceil n \log \theta \rceil U$$

where $U = \min(A,M,D)$. Let $T_1(\varphi)$ denote the time for evaluating $\varphi(x_1,x_2,\ldots,x_{i+d+1})$ 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 fact.

## ACKNOWLEDGMENTS

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

## REFERENCES

Borodin, A. B. and Munro, I. [72]. Notes on Efficient and Optimal Algorithms, University of Toronto and University of Waterloo.

Brent, R. P. [73]. The parallel evaluation of arithmetic expressions in logarithmic time, in Complexity of Sequential and Parallel Numerical Algorithms (J. F. Traub ed.), pp. 83-102. Academic Press, New York.

Flynn, M. J. [66]. Very high-speed computing systems, Proc. IEEE, Vol. 54, pp. 1901-1909.

Heller, D. [73]. A determinant theorem with applications to parallel algorithms. To appear in SIAM J. Numer. Anal. (Also available as a CMU Computer Science Department Report.)

Karp, R. Miller, R. and Winograd, S. [67]. The organization of computations for uniform recurrence equations, JACM 14, pp. 563-590.

Kogge, P. M. [72]. Parallel algorithms for the efficient solution of recurrence problem, Tech. Report 43, Digital System Laboratory, Stanford University.

Kogge, P. M. and Stone, H. S. [72]. A parallel algorithm for the efficient solution of a general class of recurrence equations, Tech. Report 25, Digital System Laboratory, Stanford University.

Knuth, D. E. [69]. The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, Mass.

Kuck, D. J. [73]. Multioperation machine computational complexity, in Complexity of Sequential and Parallel Numerical Algorithms (J. F. Traub ed.), pp. 17-46. Academic Press, New York.

Maruyama, K. [73]. On the parallel evaluation of polynomials, IEEE Trans. on Comp., C-22, pp. 2-5.

Munro, I. and Paterson, M. [73]. Optimal algorithms for parallel polynomial evaluation, JCSS 7, pp. 189-198.

Stone, H. S. [73a]. An efficient parallel algorithm for the solution of a tridiagonal system of equations, JACM, 20, pp. 27-38.

Stone, H. S. [73b]. Problems of parallel computation, in Complexity of Sequential and Parallel Numerical Algorithms (J. F. Traub ed.), pp. 1-16. Academic Press, New York.

Stone, H. S. [73c]. Private Communication.

Wilkinson, J. H. [65]. The Algebraic Eigenvalue Problem. Oxford University Press (Clarendon), London and New York.

Wulf, W. A. and Bell, C. G. [72]. C.mmp -- A Multi-Mini-Processor, AFIPS Conference Proc., Vol. 41, Part II, FJCC 1972, pp. 765-777.