NEAR OPTIMAL ALGORITHM FOR THE PARALLEL EVALUATION OF LINEAR RECURRENTS

AMNON BARAK and MARC SNIR
Department of Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel.

ABSTRACT

In this paper we develop near optimal algorithm for the parallel evaluation of linear recurrences $x_n$ of the form $x_i = x_{i-1}A_i + B_i$, $i > 1$ and $x_0 = y$, where $y$ is a given Horner expression.

It is shown that using $p$ independent processors, a linear recurrence, or equivalently, a Horner expression can be evaluated in $3n/(2p+1) + O(p)$ time units and that this result is $O(p)$ close to the lower bound for the problem.

1. INTRODUCTION

Consider a $p$-processor parallel computer such as the ILLIAC IV. In this paper we develop an algorithm for the solution of first order linear recurrences which utilize the multiprocessing capability of such a computer. Because of the importance of the evaluation of arithmetic expressions, the problem of parallel evaluation of these expressions has been investigated in recent years. Brent [1] was the first to investigate parallel algorithms using $p$ independent processors, where $p$ is a positive integer independent of $n$. He showed that a general arithmetic expression $E_n$ of $n$ variables and without division can be evaluated in at most $2n/p + O(\log n)$ time units. Winograd [5] improved Brent's results and showed that $E_n$ can be evaluated in at most $3n/2p + O(\log^2 n)$. He further conjectured that for $p > 1$, $3n/2p + \text{lower term in } n$, is the lower bound for the problem. Hyafil and Kung [2] proved that the lower bound for the computation of linear recurrences is at least $3n/(2p+1)$ for all values of $n$ and $p$.

In this paper we develop a parallel algorithm, which is easy for implementation, for the evaluation of alternating expressions $E_{2n}$ of $2n$ variables, denoted by

$$E_{2n} = (\ldots (a_0^n b_{i1})^0 a_1 \ldots )^0 b_n )^0 a_n,$$

where $n_i \in \{*, \ldots \}$ and $\theta_i \in \{+, -\}$ for $i \geq 1$.

We show that $E_n$ can be evaluated in at most $3n/(2p+1) + O(p)$ steps, by using $p$ independent processors in cases which do not include divisions, thus achieving a speed up of $2p/3 + 1/3$ over the serial computation with $p = 1$. If we compare the bound that we achieved to
the proven lower bound in [2] we conclude that our result is asymptotically optimal for this case.

We note that in [4] we developed algorithms for the parallel evaluation of general alternating expression, polynomial expression and general rational functions of n variables in $5n/(2p+3) + O(p)$, $3n/(2p+1) + O(p^2)$ and $5n/(2p+3) + O(p^2)$ steps respectively.

2. ALTERNATING EXPRESSIONS

In this section we develop algorithms for the parallel evaluation of alternating expressions of $2n$ variables using $p$ processors.

Let $E_{2n}$ be an alternating expression,

$$E_{2n} = \left(\ldots(a_0^n b_1^{\theta_1} a_1)\ldots\right)^n b_n^{\theta_n} a_n$$

where $n_i \in \{\ast\}$ and $\theta_i \in \{+,-\}$.

To compute $E_{2n}$ we define a set of linear recurrences relations:

$$(2.1) \quad y_0 = a_0$$

$$y_1 = (y_{i-1}^n b_1^{\theta_1} a_1), \quad i = 2,3,\ldots,n,$$

where

$$E_{2n} = y_n$$

The recurrence relation (2.1) consists of $n$ equations. Since only $p$ processors are available we combine successive relations into groups of relations such that we have $p$ recurrence relations.

Define

$$x_0 = \left(\ldots(a_0^n b_1^{\theta_1} a_1)\ldots\right)^{k_1} b_{k_1}^{\theta_{k_1}} a_{k_1}$$

$$x_1 = \left(\ldots(x_0^{k_1} b_{k_1}^{\theta_{k_1}} a_{k_1} + b_{k_1+1}^{\theta_{k_1+1}} a_{k_1+1})\ldots\right)^{k_2} b_{k_2}^{\theta_{k_2}} a_{k_2}$$

$$x_2 = x_1^* A_2 + B_2$$

$$(2.2) \quad x_{p-1} = x_{p-2}^* A_{p-1} + B_{p-1} = E_{2n}$$

We now introduce parallelism into the computation of (2.2) by computing the coefficients $A_i, B_i$ of $x_i$, $i = 1,2,\ldots,p-1$ simultaneously and independently in $p-1$ processors and, at the same time, we compute $x_0$. Our strategy is to choose $k_1,k_2,\ldots,k_p$ in such a way that when the computation of $x_0$ is done we also have $A_1$ and $B_1$. Therefore the time for the computation of $x_1$, is

$$t_1 = t_0 + 2.$$
Similarly we have
\[ t_i = t_{i-1} + 2 \]
\[ = t_0 + (i-1)2, \quad 1 \leq i \leq p-1. \]
The total time for all the computation is therefore
\[ t_{p-1} = t_0 + (p-1)2. \]

The analysis that follows is intended for the selection of \( k_1 \), in \( x_0 \) and for the number of operation needed to compute \( A_i \) and \( B_i \) of \( x_i \).

**Lemma 2.1:**

Let \( x_n = (\ldots(x_0^{n_1 b_1}a_1)\ldots)^{n_n b_n}a_n \)
where \( n_i \in \{\ast\} \) and \( a_i \in \{+,-\} \).

Then
\[ x_n = x_0^{n_1 b_1}a_1 \ldots \]
where \( A_n \) and \( B_n \) can be evaluated in \( \max(3n-3,0) \) operations.

**Proof:**

For \( i = 0,1 \) the assertion is evident.

Suppose that
\[ x_{n-1} = A_{n-1}^{n-1} x_0 + B_{n-1} \]
where \( A_{n-1} \), \( B_{n-1} \) can be evaluated in \( 3(n-1)-1 \) operations. Then we can write for \( x_n \)
\[ x_n = (A_{n-1}^{n_1 b_1} x_0 + (B_{n-1}^{n_1 b_1}a_1)^n a_n \).

Define
\[ A_n = A_{n-1}^{n_1 b_1} \]
and
\[ B_n = (B_{n-1}^{n_1 b_1}a_1)^n a_n \]
and the conditions of the lemma are satisfied.

We now prove a theorem which establishes a bound for the number of steps required to evaluate an alternating expression without division.

**Theorem 2.2:**

Let \( E_{2n} = (\ldots(a_0^{n_1 b_1}a_1)\ldots)^{n_n b_n}a_n \)
where
$\eta_i \in \{\ast\}$ and $\Theta_i \in \{+, -\}$.

$E_{2n}$ can be evaluated with $p$ processors in $\frac{6n}{2p+1} + p + 2$ steps.

**Proof:**

We first assume the case

$$n > \frac{1}{3} (p+2)(p-1).$$

Let $0 \leq k_1 < k_2 < \ldots < k_p = n$.

Define

$$x_i = (\ldots(a_0, b_1) \Theta \ldots) \eta_i b_i \Theta k_i k_i$$

for $i = 1, 2, \ldots, p$

where $x_p = E_{2n}$.

According to lemma 2.1

$$x_i = x_{i-1} * A_i \Theta B_i; \quad i = 2, 3, \ldots, p$$

where $A_i$ and $B_i$ can be computed in $\max(3p_i - 3, 0)$ operations where

$$p_i = k_i - k_{i-1}$$

for $2 \leq i \leq p$.

We shall now use processor $p_i$ for the computation of $x_i$, i.e. $A_i$ and $B_i$.

Let $T_i$ be the number of steps necessary to evaluate $x_i$, then it is clear that

$$T_i \leq 2p_1$$

where $P_1 = k_1$.

For $i \geq 2$, $x_{i-1}$ is already computed in $\max(T_{i-1}, 3p_i - 4)$, as is $A_i$, and we need at most one more step to complete the computation of $B_i$. Two more steps are required now for the computation of $x_i$. During the first step we compute $x_{i-1} * A_i$ in one processor and complete the computation of $B_i$ in the other. During the second step we evaluate $x_i$.

We have

$$T_i \leq \max(T_{i-1}, 3p_i - 4) + 2, \quad 2 \leq i \leq p,$$

therefore

$$T_p \leq \max(2P_1 + 2(p-1), \quad 3P_i - 2 + 2(p-1)), \quad 2 \leq i \leq p.$$
We will complete the proof of the theorem by showing a set of $P_i$ which satisfies all the requirements.

Let

$$p_i = \frac{3n-(p-1)(p+2)}{2p+1}$$

and

$$\tilde{p}_i = \frac{2}{3}(p_i + 1), \ i = 2, 3, \ldots, p.$$  

Then we have from (2.3)

$$\tilde{p}_i > 0$$

and

$$\sum_{i=1}^{p} p_i = n.$$  

We now have

$$2P_1 + 2(p-1) = 3\tilde{p}_1 - 2 + 2(p-1), \ 2 \leq i \leq p$$

so that

$$T_p = \max_{2 \leq i \leq p} (2P_1 + 2(p-1), \ 3\tilde{p}_1 - 2 + 2(p-1)) = 2P_1 + 2(p-1) = p - \frac{5}{2} + \frac{9}{4p+2} + \frac{6n}{2p+1}.$$  

Choose $P_i$ for $1 \leq i \leq p$ such that

$$|P_i - \tilde{P}_i| < 1,$$

$$P_i > 0$$

$$\sum_{i=1}^{p} p_i = n$$

and

$$k_1 = \sum_{j=1}^{i} P_j.$$

then the algorithms can evaluate $E_{2n}$ in $T_p$ steps where

$$T_p \leq T_p + 3 \leq \frac{6n}{2p+1} + p + 1/2 + \frac{9}{4p+2}$$

or

$$T_p \leq \left\lfloor \frac{6n}{2p+1} \right\rfloor + p + 2$$

and the theorem is proved for $n > \frac{1}{3}(p+2)(p-1)$. It is clear that the
theorem holds for \( p = 1 \) and for \( n \leq p + 2 \).

If \( p > 1 \) and \( p + 2 < n < \frac{1}{3} (p+2)(p-1) \) we choose \( q \) to be

\[
q = \max \{ r : n \geq \frac{1}{3}(r+2)(r-1) \}.
\]

We can compute \( E_{2n} \) using \( q \) processors in \( T_q \) steps where

\[
T_q \leq \frac{6n}{2p+1} + q + 1/2 + \frac{9}{4q+2},
\]

and we are still within the bounds of the theorem.

We note that it is evident that for small \( n \) better algorithms can be developed.

REFERENCES


