Reprinted from INFORMATION AND CONTROL Vol. 67, Nos. 1–3, October/November/December 1985

Printed in Belgium

Optimal Parallel Pattern Matching in Strings

UZI VISHKIN*

Courant Institute, New York University, and
Department of Computer Science, Tel Aviv University,
Tel Aviv 69 978, Israel

Given a text of length n and a pattern of length m, we present a parallel linear algorithm for finding all occurrences of the pattern in the text. The algorithm runs in $O(n/p)$ time using any number of $p \leqslant n/\log m$ processors on a concurrent-read concurrent-write parallel random-access-machine. © 1985 Academic Press, Inc.

I. INTRODUCTION

The family of models of computation used in this paper is the parallel random-access-machines (PRAMs). All members of this family employ p synchronous processors all having access to a common memory. The present papers refers to two member of the PRAM family. Our presentation focuses on the concurrent-read concurrent-write (CRCW) PRAM. This model allows simultaneous reading from the same memory location as well as simultaneous writing. In the latter case, the smallest serial numbered among the processors that attempt to write succeeds. At the end of the paper we show that a weaker concurrent-read concurrent-write PRAM model, where several processors may attempt to write at the same memory location only if they seek to write the same thing, actually suffices for the strongest results in this paper. There, we also show how to implement some of the results on a concurrent-read exclusive-write (CREW) PRAM, where simultaneous reading into the same memory location but not simultaneous writing is allowed. See Vishkin (1983a) for a recent survey of results concerning the PRAM family.

Let Seq(n) be the fastest known worst-case running time of a sequential algorithm, where n is the length of the input for the problem being considered. Obviously, the best upper bound on the parallel time achievable

* This research was supported by DOE Grant DE-AC02-76ER03077, by NSF Grants NSF-DCR-8413359, NSF-MCS79-21258, and NSF-DCR-8318874, and by ONR Grant N0014-85-K-0046.

0019-9958/85 $3.00

using p processors without improving the sequential result is of the form $O(\mathrm{Seq}(n)/p)$. A parallel algorithm that achieves this running time is said to have *optimal speed-up* or more simply to be *optimal*. A goal, in serial computation, is to design linear time algorithms ($O(n)$ time). Analogously, a goal, in parallel computation, is to design algorithms whose running time is proportional to n/p, where p is the number of processors used. In this case we say that a parallel algorithm achieves *parallel linear* running time.

There are not too many parallel algorithms which are optimal, in spite of the interest in them. We mention below a few such algorithms. We also include in this list a few algorithms which are very close to being optimal. (It was an arbitrary decision to include only algorithms which are away from optimal by a factor of $O(\log n)$, where n is the size of the problem being considered.) See Chin, Lam, and Chen (1981) and Vishkin (1984a) (among many others) for computation of partial (prefix) "sums" of n variables, where the word "sum" stands for any associative binary operation, Shiloach and Vishkin (1981) for finding the maximum among n elements (Borodin and Hopcroft, 1982; Kruskal, 1982; Shiloach and Vishkin, 1981) for merging two sorted lists (Akl, 1984; Reischuk, 1981 (a randomized algorithm) and Vishkin, 1983b) for finding the k smallest out of n elements (Ajtai, Komlos, and Szemeredi, 1983; Reif and Valiant, 1983 (a randomized algorithm) and Shiloach and Vishkin, 1981) for sorting, (Paul, Vishkin, and Wagener, 1983) for various operations on 2-3 tree, (Kruskal, Rudolf, and Snir, 1985; Vishkin, 1984b (a randomized algorithm), and Vishkin, 1985b) for ranking a linked list, (Galil, 1984) for string matching (where the symbols are taken from an alphabet whose size is bounded), (Aggarwal *et al.*, 1985; Nath *et al.*, 1981) for computing convex hulls (Aggarwal *et al.*, 1985, also considers other problems from computational geometry), (Bar-On and Vishkin, 1985) for generation of a computation tree form of an arithmetic expression and for finding matches in a sequence of parentheses, (Tarjan and Vishkin, 1983; Vishkin, 1985a) for computing preorder, postorder, lowest common ancestors and other tree functions, (Awerbuch and Shiloach, 1983; Chin, Lam, and Chen, 1981; Hirschberg, Chandra, and Sarwate, 1979; Savage and Ja'Ja', 1981; Shiloach and Vishkin, 1982a; Vishkin, 1984a; Wyllie, 1979) for computing connected components and minimal spanning forests of graphs, (Tsin and Chin, 1984; Tarjan and Vishkin, 1983) for computing biconnected components of graphs, (Vishkin, 1985a) for finding strongly connected orientation of the edges in undirected graphs, (Awerbuch *et al.*, 1984; Atallah and Vishkin, 1984) for finding Euler tours in directed and undirected graphs, and (Shiloach and Vishkin, 1982b) for finding maximum flow in a network. See (Heller, 1978) for a survey of numerical parallel algorithms.

The *string matching* problem is defined as follows. *Input.* Two arrays PATTERN and TEXT whose lengths are m and n, respectively. *Output.* A

Boolean array MATCH of length n. MATCH(i), $1 \leqslant i \leqslant n$, indicates if an occurrence of PATTERN starts at TEXT(i).

The main contribution of this paper is in presenting an original parallel linear algorithm for the general case of this problem which runs in $O(\log m)$ time on a CRCW PRAM. The text analysis part of the algorithm is parallel linear and runs in $O(\log n)$ time on a CREW PRAM. The algorithm can also be implemented as a parallel linear algorithm which runs in time $O(\log^2 n)$ on a CREW PRAM.

There are two known linear time serial algorithms for this extensively studied problem, due to (Boyer and Moore, 1977; Knuth, Morris, and Pratt, 1977). Recall that every parallel linear algorithms is, in particular, a linear time serial algorithm. The present result is stronger than theirs in the sense that it gives a parallel linear algorithm while theirs serial algorithms do not seem to imply satisfactory parallel linear algorithms. Moreover, our algorithm is not more complicated than theirs. Some parts of it (particularly, the analysis of the text) are even considerably simpler.

The string matching algorithm of (Galil, 1984) runs in $O(\log n)$ time using $n/\log n$ processors but requires the size of the alphabet to be fixed. However, it needs n processors in order to obtain $O(\log n)$ time for the general case considered here, and simulating it by a single processors takes $O(n \log n)$ time. Unlike his algorithm, ours does not use the "Four Russians Trick" (Aho, Hopcroft, and Ullman, 1974). There, $O(\log n)$ bits are packed into a single register and then each instruction concerning this register is counted as one operation. We use a few ideas from Galil's paper but are able to improve his result due to the following:

(1) Novel algorithmic ideas for the string matching problem. We sketch briefly one such notable idea. A formal presentation of this idea is given in Section 3. The pattern is preanalyzed and the following table is constructed. Consider the following proposition: "The suffix starting at position i of the pattern is a prefix of the pattern." For each i, $1 \leqslant i < m$, the table will either indicate that the proposition is true, or point to a single character following i that provides a *counter example* to the proposition.

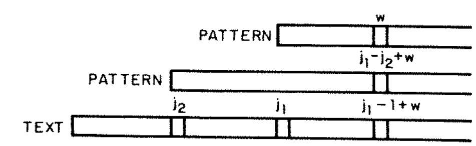

FIG. 1. A duel between positions j_1 and j_2 of the text: PATTERN(w) $\neq$ PATTERN($j_1 - j_2 + w$); if TEXT($j_1 - 1 + w$) $\neq$ PATTERN(w) then there is no occurrence of the pattern at j_1; if TEXT($j_1 - 1 + w$) $\neq$ PATTERN($j_1 - j_2 + w$) then there is no occurrence of the pattern at j_2.

Let $j_1 > j_2$ be two locations of the *text* such that $j_1 - j_2 < m$ and the suffix starting at position $j_1 - j_2 + 1$ of the pattern is not a prefix of the pattern. Following the analysis of the pattern, position $j_1 - j_2 + 1$ of the table points to a counter example, say w. That is, PATTERN$[j_1 - j_2 + w] \neq$ PATTERN$[w]$. *The duel idea.* (See also Fig. 1). It is impossible that occurrences of the pattern start both at location j_1 and j_2 of the text. Moreover, the $j_1 - 1 + w$ position of the text can be either the w position of the pattern, the $j_1 - j_2 + w$ position of the pattern or neither (but not both). The idea of a duel between j_1 and j_2 is to compare this position of the text with each of these positions of the pattern. Thereby, we can eliminate the possibility that an occurrence of the pattern starts in at least one of j_1 or j_2.

Now, consider the set of locations of the text such that at time t of an algorithm the possibility that an occurrence of the pattern starts at each of them has not (yet) been ruled out. Applying duels between successive pairs of these locations enable us to decrease by a factor of two a bound on the cardinality ("density") of this set.

(2) A careful assignment of processors to their jobs (using Brent's theorem).

The text analysis part of the algorithm is described in Section 3 and the pattern analysis part in Section 4.

II. Preliminaries

The following notation is used in this paper: Let x be a real number. $\lceil x \rceil$ is the smallest integer which is $\geqslant x$. $[x]$ is the largest integer which is $\leqslant x$. Let u be a string; $|u|$ is the number of characters in the string.

Most of this section is devoted to definitions and known facts regarding periodicity in strings.

Let u, w be two strings. u is a *period* of w if w is a prefix of u^k for some k, or equivalently if w is a prefix of uw. We call the shortest period of a string w *the period* of w. w has *period size* P if the length of the period of w is P. If w is at least twice longer than its period we say that w is *periodic*. We will use some simple facts about periodicities.

Proposition 1. *Let v be a periodic string and let w, $|w| \leqslant |v|/2$, be a period of v. Suppose w itself is periodic and u is a period of w such that $w = u^k$, $k > 1$. Then u is a period of v.*

Proof. v is a prefix of w^s for some $s > 1$. Hence, v is a prefix of u^{ks}.

Proposition 2 (The periodicity Lemma (Lyndon and Schutzenberger, 1962). *If w has two periods of size P and Q and $|w| \geqslant P + Q$, then w has a period of size* $\gcd(P, Q)$.

In the rest of this section an *occurrence* of some pattern *at j* will mean that the pattern is a substring beginning at position j of a given fixed string z. For proofs of Propositions 3–6 below, see Galil (1984).

Proposition 3. *If v occurs at j and $j + P$, for any $P \leqslant |v|/2$, then (1) v is periodic with a period of length P, and (2) v occurs at $j + P$, where P is the period size of v.*

In the rest of this section we consider a periodic string $v = u^k u'$, $k > 1$, u the period of v, u' a proper prefix of u, and $|u| = P$.

Proposition 4. *If v occurs at j and $j + mP$, $m \leqslant k$, then $u^{k+m} u'$ occurs at j.*

Proposition 5. *If v occurs at j and $j + \Delta$, $\Delta \leqslant |v| - P$, then Δ is a multiple of P.*

We call an occurrence of v at j *important* if v *does not* occur at $j + P$.

Proposition 6. *If there are two important occurrences of v at r and s, $r > s$, then $r - s > |v| - P$.*

Theorem (Brent). *Any synchronous parallel algorithm of time t that consists of a total of x elementary operations can be implemented by p processors within a time of $\lceil x/p \rceil + t$.*

Proof of Brent's Theorem. Let x_i denote the number of operations performed by the algorithm in time i ($\sum_1^t x_i = x$). We now use the p processors to "simulate" the algorithm. Since all the operations in time i can be executed simultaneously, they can be computed by the p processors in $\lceil x_i/p \rceil$ units of time. Thus, the whole algorithm can be implemented by p processors in time of

$$\sum_1^t \lceil x_i/p \rceil \leqslant \sum_1^t (x_i/p + 1) \leqslant \lceil x/p \rceil + t.$$

Remark. The proof of Brent's theorem poses two implementation problems. The first is to evaluate x_i at the beginning of time i in the algorithm. The second is to assign the processors to their jobs.

III. Analysis of the Text

The algorithm has three steps. In the first step an analysis of the pattern is performed. This analysis is used in the second step to find a sparse set of "suspicious" indices of the text. By suspicious indices we mean indices of the text in which occurrences of the pattern may start. The last step applies a character by character check to find in which of the suspicious indices an occurrence of the pattern really starts. In this section we describe the last two steps. The first step is described in the next section.

Definition. Suppose that PATTERN$[j, j+1,..., m]$ is not a prefix of PATTERN$[1,..., m]$ for some j, $2 \leqslant j \leqslant m$. That is, there exists an integer w, $1 \leqslant w \leqslant m-j+1$, such that PATTERN$(w) \neq$ PATTERN$((j-1)+w)$. We say that w is a *witness* to this mismatch. Observe that w is a witness against the existence of a period of size $j-1$ in PATTERN.

Output of Step 1. For each j, $2 \leqslant j \leqslant [m/2]+1$, Step 1 determines whether PATTERN has a period of size $j-1$ (WITNESS(j) will be 0) and computes a witness if not (it assigns such a witness to WITNESS(j)).

Steps 2 *and* 3. For $k \geqslant 0$, the *set of k-blocks* is $\{$TEXT$[1,..., 2^k]$,..., TEXT$[l2^k+1,..., (l+1)2^k]$,...$\}$. Steps 2 and 3 depend considerably on whether the pattern is periodic.

Case 1. The pattern is not periodic.

Step 2. *Initialize. For all i*, $1 \leqslant i \leqslant n-m+1$ *pardo*

$$\text{MATCH}(i) := T.$$

Recall that the goal of our algorithm is that MATCH$(i) = T$ if and only if an occurrence of the pattern starts at i, for any i, $1 \leqslant i \leqslant n-m+1$.

Let us define the *k-sparsity property*: For each k-block at most one value of MATCH is T. Namely, each of MATCH$[1,..., 2^k\}$,..., MATCH$[l2^k+1,..., (l+1)2^k]$,... contains at most one T.

The goal of Step 2 is to satisfy ($[\log m]-1$)-sparsity. However, at the end of Step 2 it will still be possible that MATCH(i) is T while there is no occurrence of PATTERN that starts at TEXT(i).

LEFT(k, a) contains the entry of the leftmost T in TEXT of k-block number a, $1 \leqslant a \leqslant |(n-m+1)/2^k|$, or an indication that there is no such T.

Let us describe stage k of Step 2. (The input to stage k satisfies $(k-1)$-sparsity.)

Stage k, $1 \leqslant k \leqslant [\log m]-1$: Satisfy k-sparsity.

The procedure given below is performed in parallel for all k-blocks. Let a be an integer satisfying $1 \leqslant a \leqslant |(n-m+1)/2^k|$. We describe the procedure for k-block a; k-block a is the union of two $(k-1)$-blocks: $2a$ and $2a-1$.

if LEFT$(k-1, 2a)$ = "null"
then LEFT(k, a) := LEFT$(k-1, 2a-1)$
else if LEFT$(k-1, 2a-1)$ = "null"
  then LEFT(k, a) := LEFT$(k-1, 2a)$
  else see below.

$(k-1)$-sparsity implies that following stage $k-1$ there is at most one index j_1 in $(k-1)$-block $2a$ and at most one index j_2 in $(k-1)$-block $2a-1$ such that MATCH(j_1) = MATCH$(j_2) = T$. The remaining case is where both indices j_1 and j_2 exist. We use the concept of a *duel* (which was described informally in the introduction) to eliminate one of these $T-s$ using information that exists in WITNESS following Step 1.

Let w be WITNESS(j_1-j_2+1). Let x = PATTERN(w), y = PATTERN(j_1-j_2+w) and z = TEXT(j_1-1+w). Since j_1 and j_2 belong to the same k-block, $j_1-j_2<2^k$. For $k<[\log m]-1$, this implies $w \neq 0$. w is a witness that PATTERN$[j_1-j_2+1,..., m]$ is not a prefix of PATTERN. Namely, $x \neq y$:

If an occurrence of PATTERN starts at j_1 then $x = z$.

If an occurrence of PATTERN starts at j_2 then $y = z$.

$x \neq y$ implies that only one of the later two equalities can be satisfied and therefore at most one of these two occurrences may hold. We use z to eliminate the possibility of (at least) one of these occurrences:

if $z \neq y$
then MATCH(j_2) := F
if $z \neq x$
then MATCH(j_1) := F

Finally,

if MATCH$(j_2) = T$
then LEFT$(k, a) := j_2$
else if MATCH$(j_1) = T$
  then LEFT$(k, a) := j_1$
  else LEFT(k, a) := "null"

Complexity. Stage k of Step 2 needs $O(n/2^k)$ operations and $O(1)$ time. Therefore, Step 2 needs a total of $O(n)$ operations and $O(\log m)$ time.

Step 3. For each α, $1 \leqslant \alpha \leqslant n-m+1$, such that MATCH$(\alpha) = T$ check, character by character, if an occurrence of the pattern starts at α:

for all j, $1 \leqslant j \leqslant |(n-m+1)/2^{[\log m]-1}|$, **pardo**
  for all i, $1 \leqslant i \leqslant m$, **pardo**
    (Denote $t(j)$ = LEFT$([\log m]-1, j)$)

if $t(j) \neq$ "null"
then if $\mathrm{TEXT}(t(j)+i-1) \neq \mathrm{PATTERN}(i)$
then $\mathrm{MATCH}(t(j)) := F$ (simultaneous writes are possible)

This results in $\mathrm{MATCH}(i) = T$ (for any i, $1 \leqslant i \leqslant n-m+1$) if and only if an occurrence of the pattern starts at location i of the text as we wanted.

Complexity. $O(mn/2^{[\log m]-1}) = O(n)$ operations and $O(1)$ time.

Case 2. The pattern is periodic.

Say that the pattern is $u^s v$, where u is the period of the pattern, $|u| = P$, and $|v| < P$ and let $Q = |v| = m - sP$ $(<P)$.

Step 2.1. Rerun Step 1 for PATTERN$[1,..., 2P]$ instead of the whole pattern. (*Remark.* Actually, there is a way to avoid rerunning Step 1. The goal of Step 2.1 is to revise the values of WITNESS, such that for every i, $2 \leqslant i \leqslant P$, $\mathrm{WITNESS}(i) \leqslant 2P - i$. Suppose that following the pattern analysis there is some i, $2 \leqslant i \leqslant P$, for which $\mathrm{WITNESS}(i) > 2P - i$. Denote $w = \mathrm{WITNESS}(i)$. The fact that the pattern has a period of length P implies that $w - P$ is also a witness for i and it is right to assign $w - P$ into WITNESS(i). Let $c > 0$ be any integer such that $w - cP > 0$. Similarly, we can assign $w - cP$ into WITNESS(i). So, we select $c = |(w - 2P + i)/P|$, and assign $w - cP$ into WITNESS(i). This results in $\mathrm{WITNESS}(i) \leqslant 2P - i$ as desired.)

Step 2.2. Perform $[\log P]$ rounds of duels with respect to the text (similar to Step 2 of Case 1 above). As a result each $[\log P]$-block of the text will have at most one index, where an occurrence of the period u may start (to be called a *suspicious index*, as before). Observe that since the information in WITNESS is based now only on u^2, every index of the text in which an occurrence of u^2 starts is suspicious.

Step 3.1. For every suspicious index check, character by character, if an occurrence of $u^2 v$ starts at it (similar to Step 3 of Case 1 above).

Steps 2.1, 2.2, and 3.1 result in the following: for every i, $1 \leqslant i \leqslant n - 2P - Q + 1$, $\mathrm{MATCH}(i) = T$ if and only if there is an occurrence of $u^2 v$ at i. These steps need a total of $O(n)$ operations and $O(\log m)$ time.

Step 3.2. Our present goal is to find for each such i the maximum k such that an occurrence of $u^k v$ starts at i. Then, if $k \geqslant s$ we conclude that an occurrence of the pattern starts at i. For completeness of the presentation we bring below the slightly tedious implementation of Step 3.2.

We will use a standard balanced binary tree with $n - 2P - Q + 1$ leaves to guide the computation. Denote $\beta = n - 2P - Q + 1$. Each node of the tree is a pair (x, y), where x is the level of the node in the tree and y is its serial number among other nodes of the same level. The leaves of the tree are $(0, 1),..., (0, 2^{|\log \beta|})$. A node (x, y) of the tree is the father of two nodes:

$(x-1, 2y-1)$, its left son, and $(x-1, 2y)$, its right son. For each i, $1 \leqslant i \leqslant \beta$, such that $\mathrm{MATCH}(i) = T$, we compute below into LARGEST(i) the largest index l such that $\mathrm{MATCH}(l) = T$ and $\mathrm{PATTERN}[i,..., l-1] = u^{(l-i)/P}$, where $(l-i)/P$ is an integer. An addition of two to $(l-i)/P$ will yield the maximum k as required.

Serially, this l can be easily computed in linear time by scanning the text from right to left. Our parallel implementation uses auxiliary arrays $A[i, j]$ and $B[i, j]$ whose entries correspond to nodes of the binary tree.

Initialization.

for all i, $1 \leqslant i \leqslant 2^{|\log \beta|}$, **pardo**
if $\mathrm{MATCH}(i) = T$ and $\mathrm{MATCH}(i+P) = F$
(**Comment.** In case the **if** condition is satisfied the maximum l for i is i itself.)
then $A(0, i) := i$
else $A(0, i) := \infty$

The computation has $2\,|\log \beta|$ stages. (*Remark.* The last paragraph of this section explains why only $2\,|\log m|$ stages suffice.) Each of the first $|\log \beta|$ stages consists of moving one level up the tree, starting from the leaves and ending at the root. They result in each $A(x, y)$ having the minimum $A(0, i)$ over its leaf-descendents:

Stage r, r := 1 to |log β|.
for all i, $1 \leqslant i \leqslant 2^{|\log \beta| - r}$ **pardo**
$A(r, i) := \min(A(r-1, 2i-1), A(r-1, 2i))$

Each of the last $|\log \beta|$ stages consists of moving one level down the tree, starting at the root and ending at the leaves. The goal in these stages is to compute into $B(0, i)$, $1 \leqslant i \leqslant \beta$, the smallest j for which the *if* condition above is satisfied and $j > i$. It can be readily verified by decreasing induction on the level r that $B(r, i)$ has the smallest j for which the if condition is satisfied such that $j > i2^r$.

Set, $B(|\log \beta|, 1) := \infty$.

Stage 2 |log β| + 1 − r, r := |log β| downto 1.
for all i, $1 \leqslant i \leqslant 2^{|\log \beta| - r}$ **pardo**
$B(r-1, 2i) := B(r, i)$
if $A(r-1, 2i) < \infty$
then $B(r-1, 2i-1) := A(r-1, 2i)$
else $B(r-1, 2i-1) := B(r, i)$

In order to complete the computation of LARGEST perform:

for all i, $1 \leqslant i \leqslant n$, **pardo**
if $\mathrm{MATCH}(i) = T$ and $\mathrm{MATCH}(i+P) = F$
then $\mathrm{LARGEST}(i) := i$
else $\mathrm{LARGEST}(i) := B(0, i)$

It is straightforward to see that for every i, $1 \leqslant i \leqslant \beta$, such that MATCH($i$) = T, LARGEST(i) has the desired value. Finally,

for all i, $1 \leqslant i \leqslant \beta$, such that MATCH($i$) = T **pardo**
  $k := (\text{LARGEST}(i) - i)/P + 2$
  if $k < s$
  then MATCH(i) := F

Complexity. The number of operations required by Step 3.2 is proportional to the number of nodes in the binary tree ($O(\beta)$) and the time is proportional to its height ($O(\log \beta)$).

(*Remark.* There is a way to modify Step 3.2 such that the number of operations will remain $O(n)$ but the time will be reduced to $O(\log m)$. The modified Step 3.2 will find for each i, such that MATCH(i) = T following Step 3.1, whether there is k, such that $k \geqslant s$ and $u^k v$ starts at i. Observe, that now we are not looking for the largest such k. Essentially, the modified Step 3.2 consists of the first $|\log m|$ stages and the last $|\log m|$ stages given above. In the first $|\log m|$ stages we move $|\log m|$ stages up the tree. Since each node of the form $[|\log m|, y]$ has $2^{|\log m|}$ leaf descendents, it is straightforward to adapt the last $|\log m|$ stages (above) to compute for each i, whether there is $k \geqslant s$ such that $u^k v$ starts at i.)

So, Steps 2 and 3 of Case 2 also need a total of $O(n)$ operations and $O(\log m)$ time. Apply Brent's theorem to get a bound of $O(\log m)$ time using $n/\log m$ processors for both Cases 1 and 2. The reader is invited to verify that here and throughout the rest of the algorithm the implementation problems in the remark following Brent's theorem can be readily overcome.

IV. Step 1—Analysis of the Pattern

The pattern is the input for Step 1. Step 1 consists of manipulating the array WITNESS, whose length is m. Recall that in the previous section we already specified what WITNESS must include following Step 1. It is initialized as follows:

for all j, $1 \leqslant j \leqslant m$ **pardo**
  WITNESS(j) := 0 (*Interpretation.* PATTERN[$j, j+1, \ldots, m$] is "suspected" to be a prefix of the pattern.)

In this section, the *set of k-blocks* refers to the pattern. It is $\{\text{PATTERN}[1, \ldots, 2^k], \ldots, \text{PATTERN}[l2^k + 1, \ldots, (l+1)2^k], \ldots\}$.

Step 1 consists of $[\log m] - 2$ or $[\log m] - 3$ iterations (called *stages*) and a terminal stage. Later in this section we describe the terminal stage and how to determine the exact number of iterations to be performed. Following stage k, the following three properties are satisfied:

(1) *The k-certainty property.* For j, $1 \leqslant j \leqslant 2^k$, WITNESS($j$) = 0 if and only if an occurrence of PATTERN[$1, \ldots, 2^{k+1} - j + 1$] starts at PATTERN($j$). That is, for $1 \leqslant j \leqslant 2^k$, WITNESS($j$) = 0 indicates that we are certain that there is such an occurrence at j. The k-certainty property can alternatively be presented as follows:

Imagine that PATTERN[$1, \ldots, 2^{k+1}$] has the whole pattern. Then, WITNESS[$1, \ldots, 2^k$] *has its final values as required by the output definition of Step* 1. Obviously, WITNESS(1) must be always zero.

(2) *The k-sparsity property.* (In this section it will apply to the pattern.) If WITNESS[$2, \ldots, 2^k$] does not have any zero then WITNESS of each k-block has at most one zero. (That is, each of WITNESS[$1, \ldots, 2^k$], ..., WITNESS[$l2^k + 1, \ldots, (l+1)2^{k+1}$], ... contains at most one zero.)

(3) *The k-lookahead property.* WITNESS(i) $\leqslant 2^{k+1}$ for every index i of the pattern.

Satisfying the k-certainty and the k-sparsity properties is a fairly intuitive goal, while satisfying the k-lookahead property may seem counter-intuitive. Particularly, since satisfying it implied in several places not using available information which seemed as if it will speed up the algorithm. Therefore, our presentation focuses on satisfying the first two properties. We prove later that the k-lookahead property is satisfied as well (in Lemma 1).

We describe now stage $k+1$ of Step 1. We follow closely the illustrative description which is given in Fig. 2. After stage k we must be at either the arrow leading to Box 2 or at the arrow leading to Box 4. In either case "k-certainty" is satisfied.

"k-sparsity" is satisfied when we enter Box 2; k-sparsity need not be satisfied at a periodic mode (i.e., if we were at Box 4 in stage k and proceeded to stage $k+1$ at Box 4).

LEFT(k, a) relates in this section to the pattern. It contains the entry of the leftmost zero in WITNESS of k-block number a, $1 \leqslant a \leqslant |m/2^k|$, or an indication that there is no such zero. If PATTERN[$1, \ldots, 2^{k+1}$] has a period of size $\leqslant 2^k - 1$ then PERIODICITY(k) contains the period size.

Let us specify the instructions in each of the boxes. The instructions for boxes 2–5 assume that they are activated in stage $k+1$:

Box 1 (Start).
for all j, $1 \leqslant j \leqslant m$ **pardo**
  WITNESS(j) := 0
if PATTERN(1) $\neq$ PATTERN(2)
then start stage 1 at Box 2
else start stage 1 at Box 4 (enter a periodic mode)

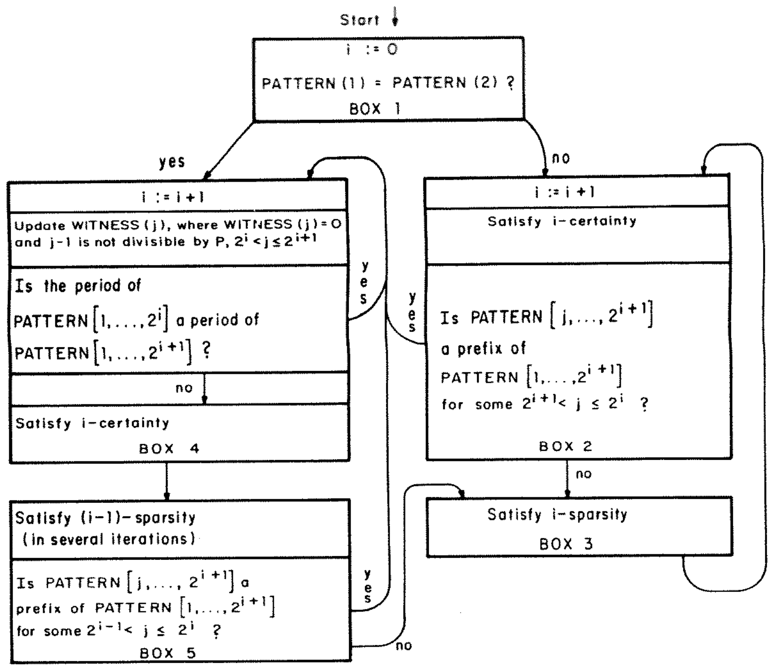

FIG. 2. Step 1.

Box 2. (Upon entering Box 2, k-sparsity and k-certainty are satisfied. WITNESS$[2,..., 2^k]$ has no zeros.) If suspected periodicity has been found start stage $k+2$ at a periodic mode (Box 4). Otherwise, progress to Box 3. Specifically,

if LEFT$(k, 2) \neq$ "null" (i.e., does WITNESS of k-block number 2 has a zero? Note that k-sparsity implies that there is at most one such zero)
then (let $x = \text{LEFT}(k, 2)$)
  for all j, $1 \leqslant j \leqslant 2^{k+2} - x + 1$ **pardo**
    if PATTERN$(j) \neq$ PATTERN$(x-1+j)$
    then WITNESS$(x) := j$ (Note that the **if** statement condition may hold for several j's. This would result in simultaneous writes into WITNESS(x))
  if WITNESS$(x) = 0$ (i.e., the condition did not hold for any j)
  then PERIODICITY$(k+1) := x - 1$; Start stage $k+2$ at Box 4
Proceed to Box 3 (the situation is that WITNESS$[2,..., 2^{k+1}]$ has no zeros)

Box 3. (k-sparsity and $(k+1)$-certainty are satisfied. For every $2 \leqslant j \leqslant 2^{k+1}$, WITNESS$(j) \neq 0$.) Satisfy $(k+1)$-sparsity. The procedure given below is performed in parallel for all $(k+1)$-blocks. Let a be an integer satisfying $2 \leqslant a \leqslant \lfloor m/2^{k+1} \rfloor$. We describe the instructions for $(k+1)$-block a. $(k+1)$-block a is the union of two k-blocks: $2a$ and $2a-1$:

if LEFT$(k, 2a) =$ "null"
then LEFT$(k+1, a) :=$ LEFT$(k, 2a-1)$
else if LEFT$(k, 2a-1) =$ "null"
  then LEFT$(k+1, a) :=$ LEFT$(k, 2a)$
  else see below.

k-sparsity implies that there is at most one index j_1 in k-block $2a$ and at most one index j_2 in k-block $2a-1$ such that WITNESS$(j_1) =$ WITNESS$(j_2) = 0$. The remaining case is that indices j_1 and j_2 exist. Here enters again the concept of a duel. We perform a duel between these indices in which one of these zeros will be eliminated, similar to the previous section. Let $w = \text{WITNESS}(j_1 - j_2 + 1)$, $x = \text{PATTERN}(w)$, $y = \text{PATTERN}(j_1 - j_2 + w)$ and $z = \text{PATTERN}(j_1 - 1 + w)$.

(*Implementation Remark* 1. In the present description, we ignore the case where $j_1 - 1 + w > m$ (or when there is reference to an index of the pattern which is $> m$). The algorithm proceeds as if PATTERN$(j_1 - 1 + w)$ matches any possible character. However, the k-lookahead property prevents this case from affecting the correctness of the algorithm as explained in the presentation of the terminal stage of Step 1 later.)

We use z to eliminate (at least) one of the zeros at j_1 and j_2.

if $z \neq y$
then WITNESS$(j_2) := j_1 - j_2 + w$
if $z \neq x$
then WITNESS$(j_1) := w$

Finally,

if WITNESS$(j_2) = 0$
then LEFT$(k+1, a) := j_2$
else if WITNESS$(j_1) = 0$
  then LEFT$(k+1, a) := j_1$
  else LEFT$(k+1, a) :=$ "null"

Box 4. *Periodic mode.* (Recall that we are presently describing stage $k+1$.) (Say that the last transition from Box 2 or 5 occurred at stage k_1+1. k-certainty and k_1-sparsity are satisfied. Say the period size of PATTERN$[1,..., 2^{k+1}]$ (the *suspected periodicity*) is P).

We pick indices j of the first $(k+1)$-block such that WITNESS$(j) = 0$ and $j-1$ is not divisible by P. The fact that k-certainty was satisfied upon entering Box 4, implies that each such j must belong to the second k-block. For each such index j, we select the index i, such that $j-P<i<j$ and $i-1$ is divisible by P and perform a "one way" duel between i and j in which only an assignment into WITNESS(j) can be performed. Explicitly $i=[(j-1)/P]\,P+1$. As we see below, it is very useful that $j-i<P$:

for all j, $2^k<j\leqslant 2^{k+1}$ **pardo**
  if WITNESS(j) $=0$ and j mod $P\neq 1$
  then (Let $w=$ WITNESS(j mod P)
    if PATTERN($j-1+w$) $\neq$ PATTERN(w)
    then WITNESS(j) := w

(*Explanation.* We prove later (Lemma 1) that after stage t, WITNESS(l) $\leqslant 2^{t+1}$, for every index l. Therefore, if P is the period of PATTERN[1,..., 2^{k+2}] then for all the j indices that satisfy the first **if** condition above, the second **if** condition must be satisfied as well. Hence, this instruction will result in WITNESS(j) $\neq 0$.)

CLAIM 1. *Suppose P is not a period of* PATTERN[1,..., 2^{k+2}]. *Then for at most four indices j that satisfied the first* **if** *condition* WITNESS(j) *remains* 0. To show this we need the following.

CLAIM 2. WITNESS(i) $\leqslant 2^{[\log P]+2}$, *for each* $2\leqslant i<P$.

Proof. Apply Lemma 1 and the fact that all these indices of WITNESS were updated before we entered Box 4 at stage $[\log P]+1$ when P became the suspected periodicity. We can conclude from the proof the following.

COROLLARY 1. ($[\log P]$)-*sparsity is satisfied.* From Claim 2 we can conclude that,

COROLLARY 2. *For each index* $j\leqslant 2^{k+1}-2^{[\log P]+2}$, *that satisfied the first* **if** *condition,* WITNESS(j) $\neq 0$.

Proof of Claim 1. By Corollary 2, only indices j, $2^{k+1}-2^{[\log P]+2}<j\leqslant 2^{k+1}$, that satisfied the first **if** condition may have WITNESS(j) $=0$. These indices may be included in at most four ($[\log P]$)-blocks. (Corollary 1 implies, that WITNESS of at most one index in each of these blocks has a zero.)

Check whether the periodicity continues until index 2^{k+2} of the pattern. If yes, start stage $k+2$ at Box 4. (Observe that $(k+1)$-certainty is satisfied.) Suppose the periodicity does not continue until index 2^{k+2}. Consider the possibilities that any multiple of P, which is $<2^{k+1}$, is a period of PATTERN[1,..., 2^{k+2}]. The character of the pattern which caused the assignment into WITNESS($P+1$) is also a counter example to any of these possibilities. Update this into WITNESS. As a result WITNESS[2,..., 2^{k+1}] will have at most four zeros whose indices are $>2^k$ (Claim 1). Check, character by character, if any of these zeros represents a period of PATTERN[1,..., 2^{k+2}] and update WITNESS appropriately. Obviously at most one of this zeros represents a period of PATTERN[1,..., 2^{k+1}] (Proposition 2). Proceed to Box 5. (Observe that $(k+1)$-certainty is satisfied):

for all j, $2^{k+1}<j\leqslant 2^{k+2}$ **pardo**
  if PATTERN(j) $\neq$ PATTERN(j mod P)
  then WITNESS($P+1$) := $j-P$ (simultaneous writes are possible and the lowest $j-P$ will be assigned)
if WITNESS($P+1$) $=0$
(In words: Is P still the suspected periodicity?)
then start stage $k+2$ at Box 4
else for all j, $2\leqslant j\leqslant(2^{k+1}-1)/P$ **pardo**
  WITNESS($jP+1$) := WITNESS($P+1$) $-(j-1)\,P$

(*Explanation.* Observe that for each of these j's, $jP+$ WITNESS($jP+1$) is the same. As we said before, this means that the same character of the pattern contradicts periods of sizes jP, for each of these j's)

for each i, $2^k<i\leqslant 2^{k+1}$, such that WITNESS(i) $=0$ **do**
  (there are at most four such i's)
  for all j, $1\leqslant j\leqslant 2^{k+2}-i+1$ **pardo**
    if PATTERN(j) $\neq$ PATTERN($i-1+j$)
    then WITNESS(i) := j (simultaneous writes are possible)
  if WITNESS(i) $=0$ (i.e., the condition did not hold for any j)
  then PERIODICITY($k+1$) := $i-1$;
proceed to Box 5

Box 5. (($k+1$)-certainty is satisfied. WITNESS[2,..., 2^k] has no zeros. k_1-sparsity is satisfied.) Satisfy k-sparsity. This is done in $k-k_1$ iterations. In iteration t, $1\leqslant t\leqslant k-k_1$, (k_1+t)-sparsity is satisfied. Each iteration is similar to the way in which $(k+1)$-sparsity is satisfied in Box 3. The details are left to the reader—no new ideas are required. If WITNESS[2^k+1,..., 2^{k+1}] has a zero (PERIODICITY($k+1$) $\neq$ "null") then start stage $k+2$ at Box 4. Otherwise, proceed to satisfy $(k+1)$-sparsity at Box 3.

Next, we give a complexity analysis of the stages described above. Later, the *terminal stage* of Step 1 is presented. For reasons of clarity the main points required for a correctness proof of Step 1 will be combined into the presentation of the terminal stage.

Complexity analysis. Stage k. Each of Boxes 2, 3, and 4 is visited at most once in each stage. Box 2 needs $O(2^k)$ operations and $O(1)$ time. Box 3 needs $O(1)$ operations and $O(1)$ time per each of the $\leqslant|m/2^k|$

k-blocks in order to satisfy k-sparsity. Box 4 needs $O(2^k)$ operations and $O(1)$ time. Since k increases from 1 to $\leqslant [\log m]-2$, we have so far $O(m)$ operations and $O(\log m)$ time. *Box* 5: For each i, we satisfy i-sparsity at most once during these stages. As in Box 3 satisfying i-sparsity needs $O(m/2^i)$ operations and $O(1)$ time, and the same total bound of $O(m)$ operations and $O(\log m)$ time applies. Apply Brent's theorem to get $O(\log m)$ parallel running time using $m/\log m$ processors.

The terminal stage and correctness of Step 1.

Recall that our goal is to determine WITNESS(j), $2 \leqslant j \leqslant [m/2]+1$. The only problem that may arise in arguing that Step 1 achieves this goal relates to Implementation Remark 1 in Box 3. There, we describe a situation where the information in WITNESS implies a comparison with a character of the pattern whose index is $>m$. By Implementation Remark 1 the outcome of such a comparison would not affect the values in WITNESS. In order to be able to proceed in this discussion we need the following lemma.

LEMMA 1 (the k-lookahead property). *Following stage k, WITNESS(i) $\leqslant 2^{k+1}$ for every index i of the pattern.*

Proof. By induction on k. For $k=0$ (before stage 1) the lemma readily holds. We assume the lemma holds for k and show it holds for $k+1$. Let us check all instructions of stage $k+1$ in which an assignment into WITNESS(i) can be performed. The order in which boxes are visited in stage $k+1$ is first Boxes 2 or 4 and then Boxes 5 and 3 (several of the boxes may not be visited at all during this stage).

Observation. All assignments into WITNESS(i) in Boxes 2 and 4 satisfy $i-1+\text{WITNESS}(i) \leqslant 2^{k+2}$. Let us prove this observation. In Box 2 there is only one instruction in which assignment into WITNESS(i) may be performed. In this assignment, $i \leqslant 2^{k+1}$, and the number assigned is $\leqslant 2^{k+2}-i+1$. There are three instructions in Box 4 in which assignments into WITNESS(i) may be performed. In the first assignment $i \leqslant 2^{k+1}$ and WITNESS(i) is assigned a value already in WITNESS(j) for some index j, which was computed in a previous stage. By the inductive hypothesis, this assignment is $\leqslant 2^{k+1}$ and therefore, $i-1+\text{WITNESS}(i) \leqslant 2^{k+2}$. In the second and third assignments, $i \leqslant 2^{k+1}$ and the numbers being assigned are $\leqslant 2^{k+2}-i+1$. This completes the proof of the observation. In each iteration of Box 5 and in Box 3, there are two assignments into WITNESS(i), where $i > 2^{k+1}$. One is of the form $j-1+\text{WITNESS}(j)$ and the other is of the form WITNESS(j). In both assignments $j < 2^{k+1}$. The fact that the ranges of i and j above do not overlap implies that an assignment into WITNESS(i) in Box 3 or in any iteration of Box 5 cannot be affected by an assignment into WITNESS(j) in a previous iteration of Box 5 at

stage $k+1$. Let us take a closer look at the more potentially problematic assignment in Boxes 3 and 5. Namely the one of the form $j-1+\text{WITNESS}(j)$. We have to show that $j-1+\text{WITNESS}(j) \leqslant 2^{k+2}$. If WITNESS($j$) received its value before stage $k+1$ then this is implied by the inductive hypothesis. If WITNESS(j) received its value in Boxes 2 or 4 of stage $k+1$ then this is implied by the observation. We conclude that following stage $k+1$, WITNESS(i) must be $\leqslant 2^{k+2}$ for every index i.

How does the algorithm determine whether to perform $[\log m]-2$ or $[\log m]-3$ stages? Recall that we are interested only in entries of WITNESS, which are $\leqslant [m/2]+1$. Let i be an index of the pattern, such that $1 \leqslant i \leqslant [m/2]+1$. Lemma 1 implies that in the first k stages, there is no reference to an index of the pattern which is $>i+2^{k+1}$. The idea will be to run the algorithm as long as there is a $(k+1)$-block (which is of size 2^{k+1}) that can serve as a "buffer" between $[m/2]+1$ and the end of the pattern. Specifically, we will be looking for the maximum k for which there exists a $(k+1)$-block such that all its entries are $\leqslant m$ and $>[m/2]+1$. The situation is illustrated in Fig. 3.

Case 1. $m \geqslant 3.2^{[\log m]-1}$. Here $[2^{[\log m]}+1, \ldots, 3.2^{[\log m]-1}]$ (i.e., $([\log m]-1)$-block number (3) is the buffer. We can perform $[\log m]-2$ stages with this buffer protecting us from referencing any index $>m$ from the first four $([\log m]-2)$-blocks (which include $[m/2]+1$).

Case 2. $m < 3.2^{[\log m]-1}$. Here $[3.2^{[\log m]-2}+1, \ldots, 2^{[\log m]}]$ (i.e., $([\log m]-2)$-block number (4) is the buffer. We can perform $[\log m]-3$ stages with this buffer protecting us from referencing any index $>m$ from the first six $([\log m]-3)$-blocks (which include $[m/2]+1$).

Let us describe the *terminal stage* for Case 1. In a few of the subcases considered the terminal stage will determine WITNESS(i) for the first four $([\log m]-2)$-blocks. In other subcases WITNESS(i) will be determined

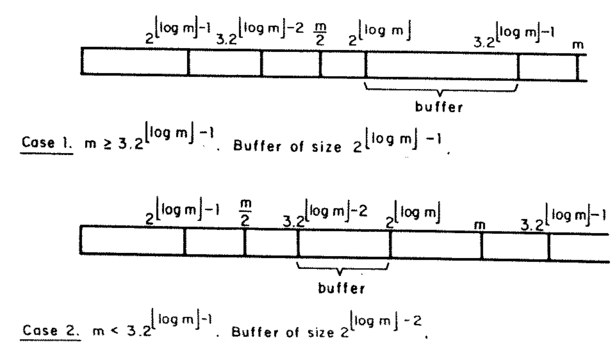

FIG. 3. The buffers for the terminal stage of Step 1.

for $i \leqslant [m/2]+1$ only. After stage $[\log m]-2$, $([\log m]-2)$-certainty is satisfied. (It is easy to prove this by induction on the number of stages using Lemma 1.) Case 1 breaks into two subcases.

Case 1.1. PATTERN$[1,..., 2^{[\log m]-1}]$ does not have a period of size $\leqslant 2^{[\log m]-2}-1$. (That is, if there had been stage $[\log m]-1$, it would have started at Box 2.) WITNESS$[2,..., 2^{[\log m]-2}]$ does not have any zero. Each one of $([\log m]-2)$-blocks 2, 3, and 4 may have at most a single zero. (Lemma 1 and induction on the number of stages are, again, all that are required to prove this. This was referred to as $([\log m]-2)$-sparsity earlier).

The terminal stage. For each of these three possible zeros, check in character by character fashion if they stand for a period of the pattern. If not, update WITNESS using (possibly) simultaneous writes into the same memory locations.

Case 1.2. PATTERN$[1,..., 2^{[\log m]-1}]$ has a period whose size is $P \leqslant 2^{[\log m]-2}-1$. (That is, if there had been stage $[\log m]-1$, it would have started at Box 4.)

The terminal stage:

for all j, $2^{[\log m]-2} < j \leqslant m$ **pardo**
(*Comment.* Check, character by character, if P is the
periodicity of the whole pattern (similar to Box 4).
  if PATTERN$(j) \neq$ PATTERN$(j \bmod p)$
  then WITNESS$(P+1) := j-P$ (Recall that we use the convention that if several processors attempt to write, then the one with the smallest j succeeds)

if WITNESS$(P+1) \neq 0$
(In words: Did we stop considering P to be the suspected periodicity?)
then for all j, $2 \leqslant j \leqslant (P+\text{WITNESS}(P+1))/P$ **pardo**
  WITNESS$(jP+1) :=$ WITNESS$(P+1)-(j-1)P$
  (*Explanation.* As in Box 4, Proposition 1 precludes the possibility that these multiples of P are sizes of periods of the pattern. It is not difficult to see that the same character of the pattern (at location WITNESS$(P+1)+P$) witnesses against each of these periods. We use later the fact that if WITNESS$(P+1)+P > [m/2]+1$, then these assignments result in WITNESS$(i) \dagger 0$ for all i, $2 \leqslant i \leqslant [m/2]+1$, of the form $jP+1$.)
  if WITNESS$(P+1)+P \leqslant [m/2]+1$
  then satisfy $([\log m]-2)$-sparsity in $[\log m]-2-k_1$ iterations (similar to Box 5. k_1+1 is the stage in which the transition into the present periodic mode was performed. Unlike Box 5, we operate here on $([\log m]-2)$-block number 2, as well.)

Explanation. The last **if** statement treats the case where P fails to be a period of PATTERN$[1,..., [m/2]+1]$. The satisfaction of the **if** condition implies that the assignments into WITNESS$(jP+1)$ earlier in the terminal stage satisfy WITNESS$(jP+1)+jP \leqslant [m/2]+1$. Similar considerations to the proof of Lemma 1, imply that, as a result of the iterations of the last instruction, each of WITNESS$[2^{[\log m]-2}+1,..., 2.2^{[\log m]-2}]$, WITNESS$[2.2^{[\log m]-2}+1,..., 3.2^{[\log m]-2}]$ $(([\log m]-2)$-blocks 2 and 3), and WITNESS$[3.2^{[\log m]-2}+1,..., [m/2]+1]$ has at most one zero. Finally, check in a character by character fashion whether these zeros should remain and, if not, update WITNESS.

Next, we deal with the two remaining cases: P is a period of the whole pattern or P is a period of PATTERN$[1,..., [m/2]+1]$ but not of the whole pattern:

if WITNESS$(P+1)=0$ or WITNESS$(P+1)+P > [m/2]+1$
then for all j, $2^{[\log m]-2} < j \leqslant [m/2]+1$ **pardo**
  if WITNESS$(j)=0$ and $j \bmod P \neq 1$ and
    PATTERN(WITNESS$((j-1) \bmod P)) \neq$
    PATTERN$(j-1+$WITNESS$((j-1) \bmod P))$
  then WITNESS$(j) :=$ WITNESS$((j-1) \bmod P)$

(*Explanation.* If P is the period of the whole pattern (WITNESS$(P+1)=0$) then this instruction would guarantee that if WITNESS$(i)=0$, $1 \leqslant i \leqslant [m/2]+1$, then it is of the form $jP+1$.

If WITNESS$(P+1)+P > [m/2]+1$, we already argued that WITNESS$(i) \neq 0$, for all i, $2 \leqslant i \leqslant [m/2]+1$, of the form $jP+1$. The last instruction updates indices i which are not of this form and results in the following.

CLAIM 3. *For at most five indices i, $2 \leqslant i \leqslant [m/2]+1$, WITNESS$(i)=0$. Similar to the proof of Claim* 1 *we use Claim* 2. Claim 2 will have the following corollaries.

COROLLARY 3. *$([\log P])$-sparsity is satisfied for the blocks that cover indices $\leqslant [m/2]+1$.*

COROLLARY 4. *For all indices $i \leqslant [m/2]+1-2^{[\log P]+2}$,* WITNESS$(i) \neq 0$.

Proof of Claim 3. By Corollary 4, only indices i, $[m/2]+1-2^{[\log P]+2} < i \leqslant [m/2]+1$, may have WITNESS$(i)=0$. These indices may be included in at most five $([\log P])$-blocks. (Corollary 3 implies, that WITNESS of at most one index in each of these blocks has a zero.) Finally, check in a character by character fashion whether these zeros should remain and, if not, update WITNESS.

It is easy to verify that Case 1 of the terminal stage needs $O(m)$ operations and $O(\log m)$ time.

Case 2. No new ideas are required to resolve this case within the same complexity efficiencies.

Complexity of Step 1. Step 1 requires $O(m/p)$ time using $p \leqslant m/\log m$ processors.

How Important is the Model of Parallel Computation?

The strongest concurrent-write model of parallel computation considered in this paper uses the following convention. Suppose that several processors attempt to write simultaneously at the same memory location. Then the lowest serial numbered among the trying processors succeeds. In a weaker concurrent-write model of parallel computation several processors may attempt to write at the same memory location only if they are seeking to write the same value. This results in this value being written into the memory location. Observe that we use the stronger model only in Step 1. We need the following problem for our discussion.

Input. A vector of p bits. Find the minimal index of the vector whose bit is 1 using p processors. Fich, Ragde, and Wigderson (1983) proposed the following $O(1)$ time algorithm for the problem in the weaker concurrent-write model of computation: Partition the input vector into $\lceil\sqrt{p}\rceil$ successive sub-vectors each of length $\lceil\sqrt{p}\rceil$ (or $\lfloor\sqrt{p}\rfloor$). For each such sub-vector, find, in $O(1)$ time using $O(\sqrt{p})$ processors, if it has a one. Apply the $O(1)$ time algorithm of (Shiloach and Vishkin, 1981) for finding the minimum among these $\lceil\sqrt{p}\rceil$ sub-vectors using p processors in the weaker model of computation. Reapply this algorithm for finding the index of minimum one in this sub-vector. Using this algorithm we can simulate the string matching algorithm, which was given in the stronger concurrent-write model, by the weaker concurrent-write model within the same bounds for time and number of processors.

Consider another problem. *Input.* A vector of l bits. Compute the OR of these bits in a concurrent-read exclusive-write PRAM. We use a balanced binary tree with l leaves to guide the computation. The number of operations of this trivial algorithm is proportional to the number of nodes in the tree and its time is proportional to its height. That is, $O(l)$ operations and $O(\log l)$ time. Apply Brent's theorem to get the $O(l/p)$ time using any number of $p \leqslant l/\log l$ processors. Using this algorithm we can run our algorithm on a concurrent-read exclusive-write PRAM in time $O(n/p)$ using any number of $p \leqslant n/\log^2 n$ processors. Using this algorithm we can run the text analysis part of our algorithm on a concurrent-read exclusive-write PRAM in time $O(n/p)$ using any number of $p \leqslant n/\log n$ processors.

Conclusion

We presented a new linear time serial algorithm for the string matching problem in which the analysis of the text is particularly simple. The algorithm is parallel linear for a very wide range for the number of processors. The exact range depends on the model of computation being used.

A natural extension of the string matching problem allows erroneous input. That is, a few characters may be omitted, a few may be replaced by others and a few superfluous characters may be added. Sankoff and Kruskal (1983) describes several applications in which we are required to solve this problem rather than our exact string matching problem. Landau and Vishkin (1985) proposed recently efficient serial algorithms for such problems. It will be interesting to try and design efficient parallel algorithms for this purpose. Baker (1978) gave a serial algorithm for two-dimensional string matching. It will also be interesting if an efficient parallel algorithm can be designed for this problem.

Acknowledgment

I am grateful to Zvi Galil for encouraging me to continue improving the results in this paper and for quite a few insights through both discussions and his paper. Helpful comments by Amir Ben-Amram, Gadi Landau, Baruch Schieber, and Dennis Shasha are also gratefully acknowledged.

RECEIVED October 2, 1984; ACCEPTED July 24, 1985

References

AGGARWAL, A., CHAZELLE, B., GUIBAS, L., O'DUNLAING, C., AND YAP, C. (1985), Parallel computational geometry, *in* "Proc. 26th Annual IEEE Symposium on Foundations of Computer Science."

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. (1974), "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass.

AWERBUCH, B., ISRAELI, A., AND SHILOACH, Y. (1984), Finding Euler circuits in logarithmic parallel time, *in* "Proc. 16th ACM Sympos. on Theory of Computing," pp. 249–257.

AKL, S. G. (1984), An optimal algorithm for parallel selection, *Inform. Process. Lett.* **19**, No. 1, 47–50.

AJTAI, M., KOMLOS, J., AND SZEMEREDI, E. (1983), An $O(n \log n)$ sorting network, *Combinatorica* **3**, No. 1, 1–19.

AWERBUCH, B., AND SHILOACH, Y. (1983), New connectivity and MSF algorithms for Ultracomputer and PRAM, *in* "Proc. 1983 International Conf. on Parallel Processing."

ATALLAH, M. J., AND VISHKIN, U. (1984), Finding Euler tours in parallel, *J. Comput. System Sci.* **29**, No. 3, 330–337.

Baker, T. P. (1978), A technique for extending rapid exact-match string matching to arrays of more than one dimension, *SIAM J. Comput.* **7**, No. 4, 533–541.

Borodin, A., and Hopcroft, J. E. (1982), Routing, merging and sorting on parallel model of computation, *in* "Proc. 14th ACM Symp. on Theory of Computing," pp. 338–344.

Boyer, R. S., and Moore, J. S. (1977), A fast string searching algorithm, *Comm. ACM* **20**, 762–772.

Bar-On, I., and Vishkin, U. (1985), Optimal parallel generation of a computation tree form, *ACM Trans. Programm. Lang. Systems* **7**, No. 2, 348–357.

Chin, F. Y., Lam, J., and Chen, I. (1981), Optimal parallel algorithms for the connected component problems, *in* "Proc. 1981 Internat. Conf. on Parallel Processing," pp. 170–175.

Fich, F. E., Ragde, R. L., and Wigderson, A. (1983), Relations between concurrent-write models of parallel computation, preprint, Div. of Computer Science, Univ. of Calif., Berkeley.

Galil, Z. (1984), Optimal parallel algorithms for string matching, *in* "Proc. 16th ACM Symp. on Theory of Computing," pp. 240–248.

Hirschberg, D. S., Chandra, A. K., and Sarwate, D. V. (1979), Computing connected components on parallel computers, *Comm. ACM* **22**, No. 8, 461–464.

Heller, D. (1978), A survey of parallel algorithms in numerical linear algebra, *SIAM Rev.* **20**, No. 4, 740–777.

Knuth, D. E., Morris, J. H., and Pratt, V. R. (1977), Fast pattern matching in strings, *SIAM J. Comput.* **6**, 322–350.

Kruskal, C. P. (1982), "Searching, Merging and Sorting on Parallel Models of Computation," Dept. of Computer Science, Univ. of Illinois, Urbana, Illinois.

Kruskal, C. P., Rudolph, L., and Snir, M. preprint, 1985.

Lyndon, R. C., and Schutzenberger, M. P. (1962), The equation $a^M = b^N c^P$ in a free group, *Michigan Math. J.* **9**, 289–298.

Landau, G. M., and Vishkin, U. (1985), Efficient string matching in the presence of errors, *in* "Proc. 26th Annual IEEE Symposium on Foundations of Computer Science."

Nath, D., Maheshwari, S. N., and Bhatt, P. C. P. (1981), Parallel algorithms for the convex hull problem in two dimensions, *in* "Proc. CONPAR 81," Lecture Notes in Computer Science Vol. 111, pp. 358–372, Springer-Verlag, Berlin/New York.

Paul, W., Vishkin, U., and Wagener, H. (1983), Parallel dictionaries on 2-3 trees, *in* "Proc. 10th Internat. Colloq. Automata. Lang. and Program.," Lecture Notes in Computer Science Vol. 154, pp. 597–609, Springer-Verlag, Berlin/New York.

Reischuk, R. (1982), A fast probabilistic parallel sorting algorithm, *in* "Proc. 22th Annual IEEE Symposium on Foundations of Computer Science," pp. 212–219.

Reif, J. H., and Valiant, L. G. (1983), A logarithmic time sort for linear size networks, *in* "Proc. 15th Annual ACM Symp. on Theory of Computing," pp. 10–16.

Savage, C., and Ja'Ja', J. (1981), Fast, efficient parallel algorithms for some graph problems, *SIAM J. Comput.* **10**, No. 4, 682–691.

Sankoff, D., and Kruskal, J. B. (Eds.) (1983), "Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison," Addison-Wesley, Reading, Mass.

Shiloach, Y., and Vishkin, U. (1981), Finding the maximum merging, and sorting in a parallel computation model, *J. Algorithms* **2**, 88–102.

Shiloach, Y., and Vishkin, U. (1982a), An $O(\log n)$ parallel connectivity algorithm, *J. Algorithms* **3**, No. 1, 57–67.

Shiloach, Y., and Vishkin, U. (1982b), An $O(n^2 \log n)$ parallel max-flow algorithm, *J. Algorithms* **3**, No. 2, 128–146.

Tsin, Y. H., and Chin, F. Y. (1984), Efficient parallel algorithms for a class of graph theoretic problems, *SIAM J. Comput.* **13**, 580–599.

Tarjan, R. E., and Vishkin, U. (1983), "An Efficient Parallel Biconnectivity Algorithm," TR 69, Dept. of Computer Science, Courant Institute, NYU; *SIAM J. Comput.*, to appear.

Vishkin, U. (1983a), "Synchronous Parallel Computation—A Survey," TR 71, Dept. of Computer Science, Courant Institute, NYU.

Vishkin, U. (1983b), An optimal parallel algorithm for selection, preprint.

Vishkin, U. (1984a), An optimal parallel connectivity algorithm, *Discrete Appl. Math.* **9**, 197–207.

Vishkin, U. (1984b), Randomized speed-ups in parallel computation, *in* "Proc. 16th Annual ACM Sympos. on Theory of Computing," pp. 230–239.

Vishkin, U. (1985a), On efficient parallel strong orientation, *Inform. Process. Lett.* **20**, 235–240.

Vishkin, U. (1985b), A deterministic parallel symmetry breaking technique with applications to list ranking, preprint.

Wyllie, J. C. (1979), "The Complexity of Parallel Computation," TR-79-387, Dept. of Computer Science, Cornell Univ., Ithaca, New York.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium