A Constant-Time Optimal Parallel String-Matching Algorithm

ZVI GALIL

Columbia University, New York, New York, and Tel-Aviv University, Tel-Aviv, Israel

In memory of Renato M Capocelli, May 3, 1940–April 8, 1992

Abstract. Given a pattern string, we describe a way to preprocess it. We design a CRCW-PRAM constant-time optimal parallel algorithm for finding all occurrences of the (preprocessed) pattern in any given text.

Categories and Subject Descriptors: F2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Constant time, CRCW-PRAM, lion hunting, optimal parallel algorithm, period, string matching

1. *Introduction*

All algorithms considered in this paper are for the parallel random access machine (PRAM) computation model. This model consists of some processors with access to a shared memory. There are several versions of this model, which differ in their simultaneous access to a memory location. The weakest is the exclusive-read exclusive-write (EREW-PRAM) model in which simultaneous read operations and write operations at the same memory location are not allowed. A more powerful model is the concurrent-read exclusive-write (CREW-PRAM) model in which only simultaneous read operations are allowed. The most powerful model is the concurrent-read concurrent-write (CRCW-PRAM) model in which read and write operations can be simultaneously executed.

In the case of the CRCW-PRAM model, there are several ways to resolve write conflicts. The weakest model, called the *common* CRCW-PRAM, assumes that when several processors attempt to write to a certain memory location simultaneously, they all write the same value. The new algorithm in

The work of Z. Galil was supported in part by National Science Foundation (NSF) grant CCR 90-14605 and CISE Institutional Infrastructure Grant CDA 90-24735.

Author's address: Columbia University, Department of Computer Science, 450 Computer Science Building, New York, NY 10027, e-mail: galil@cs.columbia.edu.

this paper is designed on the common CRCW-PRAM model. In fact, the same constant value of 1 is always used in case of concurrent writes.

An *optimal parallel algorithm* is one that requires linear work. When designing parallel algorithms, the best one can hope for is an optimal algorithm that in addition has the smallest (parallel) running time. In the case of the CREW- and EREW-PRAM, that time must be $\Omega(\log n)$ for some very simple problems [Cook et al. 1986]. But for the stronger CRCW-PRAM, the time can be constant. For example, a constant-time optimal parallel algorithm can be easily derived for computing the Boolean n-variable OR and AND functions. A less immediate constant-time parallel algorithm is known for computing the maximum of n numbers in $[1 \cdot\cdot n]$ [Fich et al. 1984]. On the other hand, it is known that no constant-time optimal parallel string-matching algorithm is possible [Breslauer and Galil 1991] (see below). But it has been an open problem whether such an algorithm exists if the cost of preprocessing is not included. In this paper, we answer this problem positively by designing such an algorithm.

The input to the string-matching problem consists of two strings: the *pattern* of size m and the *text* of size n. The output should list all occurrences of the pattern in the text. The symbols in the strings are chosen from some set Σ, which is called an *alphabet*. We assume that the input strings are stored in memory and the required output is a Boolean array of size n, which will have a "true" value if an occurrence starts at the corresponding position and "false" value, otherwise.

A naive algorithm for solving the string-matching problem can proceed as follows: consider the first $n - m + 1$ positions of the text string. Occurrences of the pattern can start only at these positions. The algorithm checks each of these positions for an occurrence of the pattern. This can be done in constant time by a common CRCW-PRAM, but requires $O(nm)$ processors.

An optimal parallel algorithm discovered by Galil [1985] solves the problem in $O(\log m)$ time. This algorithm works for fixed alphabets and was later improved by Vishkin [1985] whose algorithm works for general alphabets. Optimal algorithms by Karp and Rabin [1984] and Kedem et al. [1989] (the latter based on Karp et al.'s method [1972]) also work in $O(\log m)$ time for fixed alphabets. Breslauer and Galil [1990] obtained an optimal $O(\log \log m)$ time algorithm for general alphabets. Vishkin [1991] developed an optimal $O(\log^* m)$[1] time algorithm. Unlike the case of the other algorithms this time bound does not account for the preprocessing of the pattern. The preprocessing in Vishkin's algorithm takes $O(\log^2 \mathrm{m}/\log \log m)$ time. Vishkin's super fast algorithm raised the question whether an optimal constant-time algorithm is possible. Recently, Breslauer and Galil [1991] proved an $\Omega(\log \log m)$ lower bound for parallel string-matching over general alphabets. This lower bound implies that a slower preprocessing is crucial for Vishkin's algorithm.

In this paper, we improve Vishkin's algorithm obtaining a constant-time optimal parallel string-matching algorithm for general alphabets. As in Vishkin's algorithm, this performance does not include preprocessing. The cost of the preprocessing in our case is of the same order of magnitude as in Vishkin's algorithm.

[1] $\log^* m$ is the smallest k such that $\log^{(k)} m \leq 2$, where $\log^{(1)} m = \log m$ and $\log^{(i+1)} m = \log \log^{(i)} m$.

In many places in the paper, we treat quantities like $\log m$, $\log\log m$, $\sqrt{m}$, and $m/2$ as integers. It is easy to check that any way of rounding them to integers suffices.

This paper is organized as follows: In Section 2, we describe the problem as well as its solution in terms of hunting lions in the desert. In Section 3, we describe the algorithm. In Section 4, we list some open problems and describe some recent progress concerning some of them.

2. *Hunting Lions in the Desert*

Assume we have a desert of size n and we are given a lion of size m. We want to hunt all lions in the desert that are identical to our given lion. A lion is hunted whenever m hunters make sure that it is identical to the given lion. In our desert, lions can overlap. We show that, after preprocessing our lion, we can hunt all lions in constant time with only n hunters. A constant-time algorithm is easy if we have nm hunters. We next sketch our hunting algorithm.

We solve our problem by transforming it to other hunting problems with different, usually smaller lions. In all cases, we can assume that the hunted lion appears in our desert for a very simple reason: If it does not appear, we will not find it.

We develop a process called a *diet*. Whenever we input a big lion to the diet, it outputs a much smaller lion. As a result, we now have to hunt only all small lions in the desert. We can apply the diet a constant number of times and shrink our lion considerably. This is still not enough to get a constant-time algorithm as the size of the lion is not constant.

After making our target lion sufficiently small, we make our hunting problem even easier. Actually, there will be many different types of small lions and we will have the freedom to choose just one of them. We will have to hunt only *one* small lion of the chosen type. We will chose the small lion that is the most frequent. We will be assured that it appears many times. So hunting one of them will not be difficult.

A randomized hunt will be immediate. We do not have enough hunters to try and look for our lion at all places. Instead, our hunters will randomly try in enough places and the probability to miss all the small lions will be very small. A deterministic hunt is also possible since our pride of lion cubs has some structure. We exploit it to generate a hitting set (or an ambush set) during the preprocessing. This set has the property that if our little lions are in the desert, at least one position in the set will hit one of the lions. The set will not be large, so our algorithm will execute all the ambushes in parallel.

3. *The Algorithm*

We denote by $SM(n, m)$ the problem of string-matching with text of length n and pattern of length m. We use the term *optimally* for "in constant time with n processors." In such a case, we associate a processor with each position of the text.

Goal 0. Solving $SM(n, m)$ optimally, not including preprocessing.

We will define new goals below. After defining a new goal, show that achieving it can be used to achieve an earlier goal. After defining four more goals, we show how to achieve Goal 4, consequently achieving Goal 0. Like earlier algorithms, initially all positions in the text are candidates (for being a start of an occurrence). Subsequently, some will die and others will survive.

We now recall some definitions and facts on periodicity in strings. All strings are over the alphabet Σ. A string u is called a *period* of string w if w is a prefix of u^k for some positive integer k or equivalently if w is a prefix of uw. The shortest period of a string w is called *the period* of w. In addition, p is the length of a period of the string if and only if all the positions i, $i > p$ of the string satisfy $w_i = w_{i-p}$. For example, the period of the string *dobidobido* is *dobi*; *dobidobi* is also a period (but not the shortest one). A string is *periodic* if its period repeats at least twice. Given k, a k-block is any set of positions of the form $(kh \cdot\cdot k(h+1)]$ for some integer h. The following facts about periodicities are either well known or can easily be derived.

PERIODICITY LEMMA [LYNDON AND SCHUTZENBERGER 1962]. *If w has periods of sizes p and q and $|w| \geq p + q$, then w has a period of size $gcd(p, q)$.*

W denote by z^- the string z without its last symbol. Assume w is periodic and u is its period. The two corollaries follow from the periodicity lemma.

COROLLARY 3.1. *uu^- is not periodic.*

COROLLARY 3.2. *Assume $a \in \Sigma$, u is not the period of wa and w' is any suffix of wa of length at least $2|u|$. Then w' is not periodic.*

In both cases, if the corollary does not hold, there is a period shorter than u, and by the periodicity lemma u cannot be the period of w.

Fact 1. Without loss of generality, the pattern is not periodic.

For a periodic pattern w, we find all occurrences of (nonperiodic) uu^- and then from them all occurrences of w. (See Breslauer and Galil [1990].) From now on, *SM* will always have a nonperiodic pattern.

Fact 2. There is at most one occurrence of a nonperiodic pattern in any $m/2$-block.

Goal 1. Optimally kill all candidates except possibly one (or two) in each $m/2$-block (or $m/4$-block or $m/8$-block).

Once Goal 1 is achieved, all $O(n/m)$ survivors are checked naively, achieving Goal 0.

VISHKIN'S LEMMA [VISHKIN 1991]. *There is a set DS of at most $\log m$ positions in the pattern such that if for position i in the text we verify $pattern[j] = text[i + j]$ for all $j \in DS$, then there is $k \leq (m/2)$ such that among all $m/2$ positions in $[i - k + 1, \ldots, i + (m/2) - k]$, the pattern can only occur starting at i.*

PROOF. Consider $m/2$ copies of the pattern placed under each other, each shifted ahead by one position with respect to the previous one. Thus copy number k is aligned at position k of copy number one. Call the symbols of all copies aligned over position number i of the first copy *column i*. Since we assume that the pattern is nonperiodic and there are $m/2$ copies, for any two copies there is a column in which they differ.

Take the first and last copies and a column in which they have two different symbols. Choose one of the two symbols that appears in the column in at most half of the copies. Keep only the copies that have this symbol in that column to get a set of half the number of original copies, at most. This step can be repeated at most $\log m$ times until there is a single copy left, say copy number k. Note that all columns chosen hit copy number k. The set DS consists of the indices in copy number k of the columns considered. There are at most $\log m$ such columns. If DS is verified for position i of a text string, then no other occurrence starting at positions $i - k + 1$ to $i - k + m/2$ is possible. □

We gave the proof above for the sake of completeness. We will only use the proof of the following corollary.

COROLLARY 3.3 (VISHKIN). *If the number of processors is $n \log m$, then $SM(n, m)$ can be done in constant time.*

PROOF. We show how to achieve Goal 1 in constant time with $n \log m$ processors. Compare DS for each possible start. A position survives (initially) if all positions in DS match. The processors of each $m/2$-block optimally find the largest and the smallest survivors in the block [Fich et al. 1984]. All the others die. This is justified by Vishkin's Lemma. □

Goal 2

(a) Optimally kill all candidates, except possibly one or two for each $\log m$-block (or $\log m/4$-block or $\log m/8$-block).
(b) Optimally kill all candidates in an $m/2$-block, except at most $m/\log m$ and assign the survivors to the processors associated with the first positions of the block, one survivor per processor.

If we achieve Goal 2(a) (or 2(b)), we achieve Goal 1 as in the proof of Corollary 3.3. The only difference is that we compare DS only for each of the $\leq m/\log m$ survivors in an $m/2$-block. Thus, n processors suffice. Note that Goal 2(b) is stronger than Goal 2(a). But we will mostly use the latter.

Goal 3. Solving $SM(n, \log m)$ optimally.

We now show that if we achieve Goal 3, we can achieve Goal 2(a). Let w be the prefix of the pattern of size $\log m$.

Case (a). *w is nonperiodic.* First we optimally find all occurrences of w (Goal 3). By Fact 2, we obtain at most one survivor per $\log m/2$-block. This achieves Goal 2(a).

Case (b). *w is periodic.* This periodicity must terminate since the pattern is nonperiodic. Let uv be the shortest prefix of the pattern longer than w that does not have the same period as w and $|v| = \log m$. By Corollary 3.2, v is not periodic. First, we optimally find all occurrences of v (Goal 3). By Fact 2, there is at most one for each $\log m/2$-block. A text position i survives if and only if v occurs starting at $i + |u|$. This achieves Goal 2(a).

The following corollary derives Vishkin's algorithm. We do not need it here.

COROLLARY 3.4. *$SM(n, m)$ can be solved in $O(\log^* m)$ time with n processors.*

The time follows from the recursion $t(n, m) \leq t(n, \log m) + c$. However, the immediate algorithm is not optimal. Like Vishkin's algorithm, it can be made optimal by slightly complicating it.

We can repeat the argument and reduce our goal to:

Goal 3*. Solving $SM(n, \log\log m)$ (or even $SM(n, \log\log\log m)$) optimally. In other words, we can assume without loss of generality that the pattern is nonperiodic and of size $\log m$ (or $\log\log m$ or $\log\log\log m$). Reducing Goal 0 to Goal 3 is our advertised diet. We can also substitute $\log m$ for m in Goal 2:

Goal 2*(a). Optimally kill all candidates except possibly one or two for each $\log\log m$-block.

To achieve Goal 2 (or 2*), it is enough to consider a text of size $1.25\,m$ (having $1.25\,m$ processors). So initially the candidates are all positions in the first quarter (out of the five quarters) of the text. For longer texts, each $m/4$-block is handled separately (in parallel). We now assume that the pattern occurs in the text and reduce Goal 2(a) to Goal 4 below. We next show how to achieve Goal 4 under this assumption. If we fail to achieve Goal 4, we can conclude that the pattern does not appear in the text, in which case we can achieve Goal 2(a) (and Goal 1) by killing all candidates in the first quarter of the text. The description we just gave does not explain the need for Goal 3 and Goal 2(b), which will be made clear below.

Note that, since the pattern is nonperiodic, it cannot occur in the text more than once. Consider a substring z of the pattern. As part of the occurrence of the pattern in the text, we have also a unique corresponding occurrence of z in the text. We call these two occurrences of z (one in the pattern and one in the text) *dual* occurrences.

We return for a moment to the lions metaphor. As a result of the diet, we can look for a given small lion. We now reduce our goal so we can choose a specific small lion and we have to find only one such lion in the desert.

Goal 4. Choose z, a substring of size $\log m/4$ in the second quarter of the pattern, and optimally find one occurrence of z in the second or third quarters of the text.

Recall that we assume that the pattern occurs in the text. It follows that z satisfies the following two properties:

—z occurs in the second or third quarter of the text; and
—each occurrence of z in the second or third quarter of the text has a unique dual occurrence of z in one of the first three quarters of the pattern.

We now show that if we achieve Goal 4, we can achieve Goal 2(a). We first optimally find one occurrence of z in the second or third quarter of the text (Goal 4). We refer to it as the *found* z.

Case (a). *z is nonperiodic*. By Fact 2, z can occur at most once in each $\log m/8$-block. In the preprocessing, we find all occurrences of z in the first three quarters of the pattern and store them in the first processors, one per processor. One of these must be the dual of the found z. Each of these occurrences will give rise to one survivor whose position is the position of the found z minus the position of the occurrence. Consequently, we will have at most one survivor in each $\log m/8$-block, achieving Goal 2(a).

Case (b). *z is periodic*. In this case, the same approach may not work, because there can be too many occurrences of z in the pattern and each may be the dual of the found z. We now need the following definition and fact.

Definition 3.5. Given z, a periodic substring of w, a *segment* is a maximal substring of w containing z having the same period size as z. The *first* (*last*) z of the segment is the leftmost (rightmost) occurrence of z in the segment.

Fact 4. Given an occurrence of a periodic string z in w and the size p of the period of z, one can optimally find the first (last) z in its segment.

PROOF. Assume the given occurrence starts at position j. We first find k, the first position of the segment. The processors generate an array A. $A[0] = 0$. Processor i writes $A[i] = 1$ if $w[i] = w[i + p]$ and it writes $A[i] = 0$, otherwise. Note that k is the largest integer smaller than j with $A[k - 1] = 0$. The first z occurs starting at the smallest position $r \geq k$ such that $r = j \pmod{p}$. Both, k and r, can be found optimally since the minimum and the maximum of integers between 1 and $|w|$ can be found optimally [Fich et al. 1984]. The last z can be found similarly. □

We now return to Case (b). Let α be the segment of the found z and let β be the segment in the pattern that corresponds to α. α and β need not match because α may continue beyond the occurrence of the pattern (to the left or to the right). But the following fact will be sufficient for our purposes:

Fact 5. Either the first z's of α and β are dual or the last z's of α and β are dual.

PROOF. Since the pattern is not periodic, either β starts after the beginning of the pattern or it ends before the end of the pattern. In the former case, the first z's of α and β are dual and in the latter the last z's of α and β are dual. □

In the preprocessing, we find all the segments containing z in the pattern. For each segment, we find its first and last z's and store them in a processor; different processors for different segments. Consider two segments. They can overlap only by less than $|z|/2$ (otherwise, the periodicity continues and they must be part of the same segment). So, at most, one first z (last z) can start in each $\log m/8$-block.

After locating the found z, we find the first and last z in its segment. By Fact 4, this can be done optimally. As in Case (a), we compute from the position of the first z of α and the positions of the first z's of the segments of the pattern the corresponding survivors, at most one per $\log m/8$-block. Similarly, using the last z's we get another set of survivors, at most one per $\log m/8$-block. So we end up with at most two survivors per $\log m/8$-block, achieving Goal 2(a).

We now show how to achieve Goal 4 in case the pattern occurs in the text. Let $\Sigma' \subseteq \Sigma$ refer to the set of symbols in the second quarter of the pattern. We distinguish between three cases depending on the size of Σ'.

Case 1. $|\Sigma'| = 2$. Take z to be the most frequent substring of length $\log m/4$ in the second quarter of the pattern. It occurs at least

$$\frac{m}{4 \cdot m^{0.25}} > m^{0.7}$$

times in the second or third quarters of the text. We show how to find one such occurrence of z.

First, we give an easy randomized solution: In the processing, each of the first $m/\log m$ processors generates a random integer (= position) in $(m/4 \cdots 3m/4]$. In the search, each one of these processors with a group of $\log m/4$ processors looks for z in its chosen position. If there is an occurrence of the pattern in the text, the probability to hit an occurrence of z is at least

$$\frac{m^{0.7}}{m} = \frac{1}{m^{0.3}}$$

and the probability to miss one is smaller than

$$1 - \frac{1}{m^{0.3}}.$$

The probability to miss in all tries is thus smaller than

$$\left(1 - \frac{1}{m^{0.3}}\right)^{m/\log m} < \exp(-m^{0.6}).$$

The deterministic solution constructs a *hitting set*. This is a set of positions in the second and third quarters of the text such that if there is an occurrence of the pattern in the text, at least one position in the set must be a position of an occurrence of z. The size of the set will be $k \ll m \log m$, so each of the first k processors will have a position in the set and $\log m/4$ processors to look for z in constant time. Thus, in case there is an occurrence of the pattern in the text the algorithm will find an occurrence of z and achieve Goal 4. If there is no occurrence, it is still possible that an occurrence of z will be found. But if the algorithm fails to find an occurrence of z, we can conclude that the pattern does not occur in the text and the algorithm can kill all candidates in the first quarter of the text and achieve Goals 2(a) and 1.

In the preprocessing, we construct the set using a greedy approach. Consider a matrix defined as follows: It has $m/2$ columns corresponding to the second and third quarters of the text and $m/4$ rows corresponding to the first quarter. Row i corresponds to a potential occurrence starting at the ith position. Assuming that there is such an occurrence, the matrix has 1 for the dual of each occurrence of z in the second quarter of the pattern; namely, if the occurrences of z start at positions $j_1, j_2, \ldots,$ the matrix has 1's in positions $i + j_1 - 1, i + j_2 - 1, \ldots$ of row i.

The algorithm will choose the column hitting the maximal number of 1's and throw away all rows that are hit. The hitting set will consist of the chosen columns. Let tot_i be the number of 1's left after choosing i columns. We choose the $(i + 1)$-st column to be one with at least

$$\frac{tot_i}{(m/2) - i} \geq \frac{tot_i}{m}$$

hits. Since we eliminate a row when it is hit, each remaining row has all its 1's. So

$$tot_{i+1} \leq tot_i - \frac{tot_i}{m} \times m^{0.7}$$

as each 1 hit eliminates at least $m^{0.7}$ 1's of its row, and hence

$$tot_{i+1} \leq tot_i \times \left(1 - \frac{1}{m^{0.3}}\right).$$

Since $tot_0 < m^2$, the size of the hitting set is smaller than $k = m^{0.3} \log(m^2)$ because after choosing k columns we have

$$tot_k \leq tot_0 \times \left(1 - \frac{1}{m^{0.3}}\right)^{m^{0.3}\log(m^2)} < 1.^2$$

Case 2. $|\Sigma'| > \log m$. There is a symbol $\hat{a} \in \Sigma'$ that appears less than $m/\log m$ in the pattern. In the preprocessing, we find $\hat{a}$ and all of its occurrences in the pattern and store them with the first processors. Note that if there is an occurrence of the pattern in the text, then $\hat{a}$ must satisfy the two properties that z satisfied. In the search, we look for one occurrence of $\hat{a}$ in the second or third quarter of the text. It gives at most $m/\log m$ survivors in the first processors, achieving Goal 2(b).

Case 3. $2 < |\Sigma'| \leq \log m$. Here we achieve Goal 4*; namely, Goal 4 with a substring z of size $\log\log m$. Essentially, the same reduction to Goal 2* works as in Case 1. The number of different substrings of length $\log\log m$ is at most $(\log m)^{\log\log m} < m^{\epsilon}$ and there is one that repeats more than $m^{1-\epsilon} > m^{0.7}$ times. So the probability to miss z (in the randomized version) or the size of the hitting set are even smaller.

We now consider the cost of the preprocessing. We need Vishkin's preprocessing. We also need $SM(m, \log m)$ (that can be done by an $O(\log\log m)$ optimal parallel algorithm [Breslauer and Galil 1991]) and some other operations that can be done in constant time with m processors. In addition, we need to choose z and find its occurrences, to compute the hitting set, the size of the alphabet and possibly the frequency of the alphabet symbols. We described a sequential $O(m^2)$ algorithm for computing the hitting set that can be augmented to compute the rest in the same time bound. But, without loss of generality, the size of the pattern is $\log\log m$ and the additional preprocessing can be done in time $O((\log\log m)^2)$ by one processor. So our preprocessing is no more expensive than Vishkin's.

4. *Conclusion*

We have optimally solved parallel string-matching in case the preprocessing cost is not included. This cost is the same as in Vishkin's algorithm; namely, an optimal $O(\log^2 m/\log\log m)$ parallel algorithm preprocesses the pattern. Our work raises a number of questions, some of which have been answered recently.

An immediate question is whether we can improve the cost of preprocessing. By the lower bound of Breslauer and Galil [1991], the best one can hope for is an optimal $O(\log\log m)$ algorithm. This problem was solved recently in Cole et al. [1993], where an optimal $O(\log\log m)$ parallel algorithm for preprocessing the pattern was designed.

[2]Baruch Schieber has pointed out to us that the upper bound on the size of the hitting set generated by the greedy approach can be derived from Lovasz' work on the ratio of optimal integral and fractional covers [Lovasz 1975].

Another problem suggested by our result is whether we can do better than the $\Omega(\log\log m)$ lower bound if we allow randomization. This problem was also solved in Cole et al. [1993], where an optimal constant expected time Las Vegas algorithm for string matching (including preprocessing) is described.

In an even more recent work [Czumaj et al. 1995], the following notion is defined: An algorithm is called *work-time-optimal* if it performs the minimal possible work (which in our case is linear) and among algorithms with best work has the minimal possible time. The result of this paper can be stated as a CRCW-PRAM work-time-optimal algorithm for the text search of string matching. The result in Cole et al. [1993] extended work-time-optimality also to the first stage of preprocessing the pattern. Work-time-optimal algorithms on the weaker versions of PRAM do not follow from the one for CRCW-PRAM. The known simulations do not preserve work optimality and in some cases also lose time optimality. In Czumaj et al. [1995], work-time-optimal algorithms are given for both the pattern preprocessing and the text search for CREW-PRAMs, EREW-PRAMs, and hypercubes.

It is still open whether we can do better deterministically in case of a fixed alphabet. The lower bound of Breslauer and Galil [1991] assumes that the input strings are drawn from a general alphabet and the only access to them is by comparisons. The lower and upper bounds for the string-matching problem over a general alphabet are identical to those for a comparison-based maximum finding algorithm obtained by Valiant [1975]. Similarly an optimal constant expected-time algorithm is known for computing the maximum. A constant-time deterministic optimal algorithm can find the maximum of integers in a restricted range [Fich et al. 1984], which suggests the possibility of a faster string-matching algorithm in case of a fixed alphabet.

ACKNOWLEDGMENTS. I am indebted to Renato Capocelli for inviting me to the SEQUENCES91 workshop held in Positano, Italy in June 1991. It was only in that paradise, being disconnected from e-mail for six days, that I could solve this problem.

Thanks to Yehuda Afek, Yossi Azar, Dany Breslauer, Amos Fiat, Pino Italiano, Oded Margalit, Yossi Matias, Krishna Palem, Kunsoo Park, Baruch Schieber, Yiftach Ravid, and Joel Seiferas for helpful comments.

Yossi Matias has pointed out to me that the lions metaphor is not new. In Mosinzon [1952], two important algorithms were described: (1) how to hunt lions in the desert—later known as binary search, and (2) how to hunt crocodiles with a magnifying glass and a match box. Thanasis Tsantilas has led me to other references concerning lion hunting. It turns out, that there is quite an extensive mathematical literature on the subject, some dating back to 1938. For example see Petard [1938] and Morphy [1968].

REFERENCES

BRENT, R. P. 1974. The parallel evaluation of general arithmetic expressions. *J. ACM 21*, 2 (Apr.), 201–206.

BRESLAUER, D., AND GALIL, Z. 1990. An optimal $O(\log\log n)$ parallel string matching algorithm. *SIAM J. Comput. 19*, 6, 1051–1058.

BRESLAUER, D., AND GALIL, Z. 1991. A lower bound for parallel string matching. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (New Orleans, La., May 6–8). ACM, New York, pp. 439–443.

COLE, R., CROCHEMORE, M. A., GALIL, Z., GASIENIEC, L., HARIHARAN, R., MUTHUKRISHNAN, S., PARK, K., AND RYTTER, W. 1993. Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*. IEEE, New York, pp. 248–258.

COOK, S. A., DWORK, C., AND REISCHUK, R. 1986. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput. 15*, 1, 87–97.

CZUMAJ, A., GALIL, Z., GASIENIEC, L., PARK, K., AND PLANDOWSKI, W. 1995. Work-time-optimal parallel algorithms for string problems. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*. ACM, New York, to appear.

FICH, F. E., RAGDE, R. L., AND WIGDERSON, A. 1984. Relations between concurrent-write models of parallel computation. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, B.C., Canada, Aug. 27–29). ACM, New York, pp. 179–189.

GALIL, Z. 1985. Optimal parallel algorithms for string matching. *Inf. Cont. 67*, 144–157.

KARP, R. M., MILLER, R. E., AND ROSENBERG, A. L. 1972. Rapid identification of repeated patterns in strings, trees, and arrays. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing* (Denver, Colo., May 1–3). ACM, New York, pp. 125–136.

KARP, R. M., AND RABIN, M. O. 1987. Efficient randomized pattern matching algorithms. *IBM J. Res. Develop. 31*, 2, 249–260.

KEDEM, Z., LANDAU, G., AND PALEM, K. 1989. Optimal parallel suffix-prefix matching algorithm and applications. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures* (Santa Fe, N.M., June 18–21). ACM, New York, pp. 388–398.

LOVASZ, L. 1975. On the ratio of optimal integral and fractional covers. *Disc. Math. 13*, 383–390.

LYNDON, R. C., AND SCHUTZENBERGER, M. P. 1962. The equation $a^M = b^N c^P$ in a free group. *Mich. Math J. 9*, 289–298.

MORPHY, O. 1968. A contribution to the mathematical methods in the theory of lion hunting. *Am Math. Monthly 75*, 185–187.

MOSINZON, Y. 1952. *Hassamba*. Hebrew children's book.

PETARD, H. 1938. A contribution to the mathematical theory of big game hunting. *Am. Math. Monthly 45*.

VALIANT, L. G. 1975. Parallelism in comparison models. *SIAM J. Comput. 4*, 348–355.

VISHKIN, U. 1985. Optimal parallel pattern matching in strings. *Inf. Cont. 67*, 91–113.

VISHKIN, U. 1991. Deterministic sampling—A new technique for fast pattern matching. *SIAM J. Comput. 20*, 1, 22–40.

RECEIVED AUGUST 1992; REVISED APRIL 1995; ACCEPTED APRIL 1995