Algorithmica (1995) 14: 355–366

Algorithmica
© 1995 Springer-Verlag New York Inc.

Finding All Periods and Initial Palindromes of a String in Parallel[1]

D. Breslauer[2] and Z. Galil[3]

Abstract. An optimal $O(\log \log n)$-time CRCW-PRAM algorithm for computing all period lengths of a string is presented. Previous parallel algorithms compute the period only if it is shorter than half of the length of the string. The algorithm can be used to find all initial palindromes of a string in the same time and processor bounds. Both algorithms are the fastest possible over a general alphabet. We derive a lower bound for finding initial palindromes by modifying a known lower bound for finding the period length of a string [9]. When p processors are available the bounds become $\Theta(\lceil n/p \rceil + \log \log_{\lceil 1+p/n \rceil} 2p)$.

Key Words. Parallel algorithms, Lower bounds, Comparison model, Strings, Periods, Palindromes.

1. Introduction. A string $\mathcal{S}[0..n]$ has a *period* $\mathcal{S}[0..p-1]$ of length p if $\mathcal{S}[i] = \mathcal{S}[i+p]$ for $i = 0 \cdots n-p$. *The period* of $\mathcal{S}[0..n]$ is defined as its shortest period. Periodicity properties of strings have been studied extensively [18] and are practically used in almost all efficient sequential and parallel string-matching algorithms.

A *palindrome* is a string that reads the same forward and backward. Formally, a string $\mathcal{S}[0..k]$ is a *palindrome* if $\mathcal{S}[i] = \mathcal{S}[k-i]$ for $i = 0 \cdots k$. A string $\mathcal{S}[0..n]$ is said to have an *initial palindrome* of length k if the prefix $\mathcal{S}[0..k-1]$ is a palindrome. Palindromes have been studied for centuries as word puzzles [3] and more recently have some uses in complexity theory [14].

A parallel algorithm is said to be *optimal* if its time-processor product, that is, the total number of operations performed, is equal to that of the fastest sequential algorithm for the same problem. Note that simple parallel algorithms can compute all periods and all initial palindromes of a string in constant time using an n^2-processor CRCW-PRAM. These algorithms are not optimal since both problems have linear-time sequential algorithms [17], [20]. Our goal in this paper is to design fast optimal parallel algorithms.

The period length of a string is computed in linear time in a step of Knuth *et al.*'s [17] sequential string-matching algorithm and in optimal $O(\log \log n)$ time

[1] This work was partially supported by NSF Grant CCR-90-14605. D. Breslauer was partially supported by an IBM Graduate Fellowship while studying at Columbia University and by a European Research Consortium for Informatics and Mathematics postdoctoral fellowship.

[2] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands.

[3] Computer Science Department, Columbia University, New York, NY 10027, USA, and Tel-Aviv University, Ramat Aviv 69 978, Israel.

Received May 5, 1992; revised December 4, 1993. Communicated by F. T. Leighton.

on a CRCW-PRAM in a step of Breslauer and Galil's [8] parallel string-matching algorithm. A recent lower bound that was discovered by Breslauer and Galil [9] for finding the period length of a string shows that the $O(\log \log n)$ bound is the best possible over a general alphabet, where the only access the algorithm has to the input string is by pairwise symbol comparisons. However, Breslauer and Galil's [8] parallel string-matching algorithm as well as an $O(\log n)$-time optimal string-matching algorithm that was discovered by Vishkin [22] compute the period length p only if $p < \lceil n/2 \rceil$; knowing the fact that $p \geq \lceil n/2 \rceil$ is sufficient to obtain efficient string-matching algorithms. An earlier string-matching algorithm that was designed by Galil [13] can find all periods and all initial palindromes of a string in $O(\log n)$ time on an n-processor CRCW-PRAM. This algorithm can be made optimal by reducing the number of processors to $n/\log n$, if the input symbols are drawn from a constant-size alphabet. Other parallel string-matching algorithms that are based on the Karp–Miller–Rosenberg [15] sequential string-matching algorithm [10], [16] can also be adapted for these problems but require $O(\log n)$ time, n processors ([16] requires only $n/\log n$ processors), superlinear space, and a restricted alphabet.

In this paper we show that given an optimal parallel string-matching algorithm, all periods, including those which are longer than half of the length of the input string, can be computed in the same processor and time bounds of the string-matching algorithm. In particular, Breslauer and Galil's [8] algorithm can be used to obtain an optimal $O(\log \log n)$-time CRCW-PRAM algorithm that computes the period length of a string exactly, even if it is long. This reduction establishes that the task of computing the period length of a string in parallel is not harder than string matching.

To find the initial palindromes, we use a known reduction from the sequential setting [12] to show how the algorithm that finds all periods of a string can find all initial palindromes in the same time and processor bounds. We also prove a matching lower bound for this problem under the assumption of a general alphabet.

The paper is organized as follows: In Section 2 we overview the algorithms for finding all periods and initial palindromes. Section 3 contains the details of these algorithms and in Section 4 we prove the lower bound for finding the initial palindromes.

2. Finding the Periods.

We describe an algorithm that computes all period lengths of a given string $\mathcal{S}[0..n]$. The output of the algorithm is a Boolean array $P[1..n]$ such that $P[i] = \mathit{true}$ *iff* i is a period length of $\mathcal{S}[0..n]$.

One of the major issues in the design of PRAM algorithms is the assignment of processors to their tasks. We ignore this issue in this paper and use a general theorem that states that the assignment can be done.

THEOREM 2.1 [4]. *Any synchronous parallel algorithm of time t that consists of a total of x elementary operations can be implemented on p processors in $\lceil x/p \rceil + t$ time.*

This theorem can be used for example to slow down a constant-time p-processor algorithm to work in time t using p/t processors. We describe an $O(\log\log n)$-time algorithm using $n/\log\log n$ processors. Some of the steps in our algorithm are described as constant-time steps using n processors.

We prove the following theorem:

THEOREM 2.2. *An algorithm exists that computes* $P[1..n]$ *and takes* $O(\log\log n)$ *time using* $n/\log\log n$ *processors. If* p *processors are available the algorithm takes* $O(\lceil n/p \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ *time.*

COROLLARY 2.3. *The exact period length of a string* $\mathscr{S}[0..n]$ *can be computed in the same time and processor bounds.*

PROOF. The period length of $\mathscr{S}[0..n]$ is the smallest i such that $P[i]$ is *true*. We use a technique of Fich *et al.* [11] to compute the minimum of n integers in the range $1..n$ in constant time using an n-processor CRCW-PRAM. (By Theorem 2.1 this step can be slowed down to work in optimal $O(\log\log n)$ time or in $O(n/p)$ time on p processors.) □

COROLLARY 2.4. *All initial palindromes of a string* $\mathscr{S}[0..n]$ *can be computed in the same time and processor bounds.*

PROOF. Suppose we want to compute all initial palindromes of a string w that does not contain the symbol \$. We present $w\$w^R$ (where w^R is the string w reversed) as input to the algorithm that computes all periods of a string. Each period of this string corresponds to an initial palindrome of w. Two copies of the string $w\$w^R$ are aligned with each other shifted by some offset and the overlapping parts are identical if and only if the overlapping part is an initial palindrome of w. This reduction was used by Fischer and Paterson [12]. □

EXAMPLE The string *abaab* has an initial palindrome *aba*. This initial palindrome corresponds to the period *abaab\$ba* of the string *abaab\$baaba*.

PROOF OF THEOREM 2.2. The algorithm proceeds in independent stages which are all computed simultaneously and are described in the next section. In stage number η, $0 \le \eta < m$, the algorithm computes only $P[n - l_\eta + 1..n - l_{\eta+1}]$; where the sequence $\{l_\eta\}$ is a decreasing sequence defined as $l_0 = n$, $l_{\eta+1} = \lfloor \frac{2}{3} l_\eta \rfloor$ and m is the smallest integer for which $l_m = 0$. Note that each stage is assigned to compute a disjoint part of the output array P and the entire array is covered.

By breaking the output array into segments that are handled separately, we are able to use periodicity properties of strings [18] in each segment. These properties let us represent and manipulate the output of some string-matching problems efficiently. These ideas were successfully applied in several other parallel algorithms for string problems [1], [2], [7], [5], [6].

We denote by T_η the time it takes to compute stage number η using P_η processors. The number of operations at stage η are denoted by $O_\eta = T_\eta P_\eta$. We show later how to implement stage number η in $T_\eta = O(\log\log l_\eta)$ time and $O_\eta = l_\eta$ operations using Breslauer and Galil's [8] parallel string-matching algorithm.

Since all stages of our algorithm are executed in parallel the total number of operations performed in all stages is $\sum_\eta O_\eta \le \sum_\eta (\frac{2}{3})^\eta n = O(n)$ and the time is $\max T_\eta = O(\log\log n)$. By Theorem 2.1 the algorithm can be implemented using $n/\log\log n$ processors in $O(\log\log n)$ time.

It remains to show that if the number of available processors is p the algorithm takes $O(\lceil n/p \rceil + \log\log_{\lfloor 1+p/n \rfloor} 2p)$ time. If $p < n/\log\log n$, then by Theorem 2.1 the algorithm can be slowed down to work in $O(n/p)$ time. If $n/\log\log n \le p \le n$, then the bound above is still $O(\log\log n)$. If $p > n$, then stage number η can be implemented in $T_\eta = O(\log\log_{\lceil 1+p/n \rceil}(2p/n)l_\eta)$ time using $(p/n)l_\eta$ processors. The total number of processors used for all stages is $\sum_\eta (p/n)l_\eta \le \sum_\eta (\frac{2}{3})^\eta n = O(p)$ and the time is $\max T_\eta = O(\log\log_{\lceil 1+p/n \rceil} 2p)$. □

3. A Single Stage. In this section we describe a single stage η, $0 \le \eta < m$, that computes $P[n - l_\eta + 1..n - l_{\eta+1}]$ in optimal $O(\log\log l_\eta)$ time. Note that since a period of length p implies that $\mathscr{S}[0..n-p] = \mathscr{S}[p..n]$, there must be occurrences of $\mathscr{S}[0..l_{\eta+1}]$ starting at each position p which is a period length of $\mathscr{S}[0..n]$ and is in the range computed by this stage.

Stage η starts with a call to a string-matching algorithm to find all occurrences of $\mathscr{S}[0..l_{\eta+1}]$ in $\mathscr{S}[n - l_\eta + 1..n]$. Let q_i, $i = 1 \cdots r$, denote the indices of all these occurrences (all indices are in the string $\mathscr{S}[0..n]$, thus $n - l_\eta < q_i \le n - l_{\eta+1}$).

If no occurrences were found, the string $\mathscr{S}[0..n]$ has no period length in the range computed by this stage and all entries of $P[n - l_\eta + 1..n - l_{\eta+1}]$ can be set to *false*. Otherwise, we continue with another call to a string-matching algorithm to find all occurrences of $\mathscr{S}[0..l_{\eta+1}]$ in $\mathscr{S}[0..l_\eta - 1]$. Let p_i, $i = 1 \cdots k$, denote the indices of all these occurrences (note that $p_1 = 0$).

If there was only one occurrence of $\mathscr{S}[0..l_{\eta+1}]$ in $\mathscr{S}[n - l_\eta + 1..n]$, this occurrence can be verified to be a period length in $O(l_\eta)$ operations. However, if there are $r > 1$ occurrences, $O(rl_\eta)$ operations may be needed to verify all of them. Luckily the sequences $\{p_i\}$ and $\{q_i\}$ have a "nice" structure as we show in the following lemmas. This structure enables us to proceed efficiently to test which of the q_i's is actually a period length of $\mathscr{S}[0..n]$.

LEMMA 3.1 [19]. *If a string of length m has two periods of length p and q and $p + q \le m$, then it has also a period of length* $\gcd(p, q)$.

LEMMA 3.2. *If a string $A[1..l]$ has period length p and occurs only at positions $p_1 < p_2 \cdots < p_k$ of a string $B[1..\lceil \frac{3}{2}l \rceil]$, then the p_i's form an arithmetic progression with difference p.*

me $k \geq 2$. We prove that $p = p_{i+1} - p_i$ for $i = 1 \cdots k - 1$. The string
riod of length p and $q = p_{i+1} - p_i$. Since $p \leq q \leq \lceil l/2 \rceil$, by Lemma
s a period of length $\gcd(p, q)$. However, p is the length of the
so $p = \gcd(p, q)$ and p must divide q. The string $B[p_i .. p_{i+1} + l - 1]$
igth p. If $q > p$, then there must be another occurrence of A at
of B; a contradiction. □

*The sequences $\{p_i\}$ and $\{q_i\}$ form an arithmetic progression with
here $\mathscr{P}$ is the period length of $\mathscr{S}[0..l_{\eta+1}]$.*

sequences p_i and q_i are indices of occurrences of a string of length
ngs of length l_η. Recall that $l_{\eta+1} = \lfloor \frac{2}{3} l_\eta \rfloor$. By Lemma 3.2 the p_i's and
rithmetic progression with a difference $\mathscr{P}$, the period length of
□

es $\{p_i\}$ and $\{q_i\}$ can be represented using three integers (each): the
equence, the difference, and the length of each sequence. This
can be easily obtained from the output of the string-matching
onstant time and l_η processors.

q_i's can be ruled out of being period lengths of $\mathscr{S}[0..n]$ immediately,
the following lemma.

If $k < r$, then q_i is not a period length of $\mathscr{S}[0..n]$ for $1 \leq i \leq r - k$.

me that q_i is a period length of $\mathscr{S}$ and $1 \leq i \leq r - k$. In this case
$[0..n - q_i]$. The string $\mathscr{S}[q_i..n]$ has $r - i + 1 > k$ occurrences of
hich are $q_i, \ldots, q_r$. However, $\mathscr{S}[0..n - q_i]$ is of the same length and
urrences of $\mathscr{S}[0..l_{\eta+1}]$; a contradiction. □

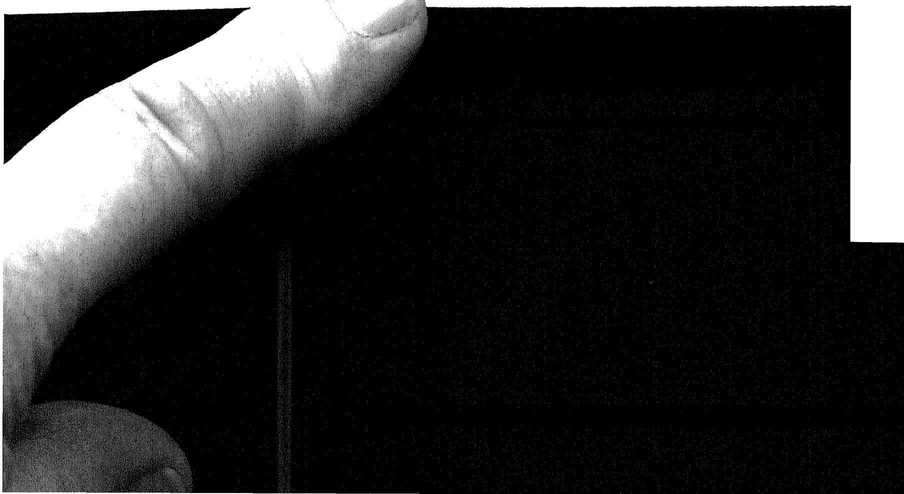
be two reasons why $q_r + \mathscr{P}$ is not included in the $\{q_i\}$ sequence:

$..\mathscr{N}] \neq \mathscr{S}[0..\mathscr{N} - q_r - \mathscr{P}]$, and $\mathscr{N} = \min(n, q_r + \mathscr{P} + l_{\eta+1})$ we call
h.

no mismatch, then the only reason that $q_r + \mathscr{P}$ is not in the $\{q_i\}$
that $q_r + \mathscr{P} + l_{\eta+1} > n$. We call this case *an overflow*.

A Mismatch). *If $\mathscr{S}[q_r + \mathscr{P}..\mathscr{N}] \neq \mathscr{S}[0..\mathscr{N} - q_r - \mathscr{P}]$, then
t most one period whose length is in the range computed by this stage.
ible period length may exist if $k \leq r$, and it is q_{r-k+1}.*

Lemma 3.4 all q_i, $1 \leq i < r - k + 1$, are not period lengths.
a period length and $i > r - k + 1$, then $\mathscr{S}[q_i..n] = \mathscr{S}[0..n - q_i]$.
$-i + 2 \leq k$ and $p_j = (j-1)\mathscr{P}$, $\mathscr{S}[q_r + \mathscr{P}..\mathscr{N}] = \mathscr{S}[p_{r-i+2}..\mathscr{N} -$
ssumption of a mismatch $\mathscr{S}[q_r + \mathscr{P}..\mathscr{N}] \neq \mathscr{S}[0..\mathscr{N} - q_r - \mathscr{P}]$.
$..\mathscr{N} - q_i] \neq \mathscr{S}[0..\mathscr{N} - q_r - \mathscr{P}]$. However $\mathscr{S}[p_{r-i+2}..p_{r-i+2} +$
$l_{\eta+1}]$ and also $\mathscr{N} - q_r - \mathscr{P} \leq l_{\eta+1}$; a contradiction. □



LEMMA 3.6 (An Overflow). *If* $\mathcal{S}[q_r + \mathcal{P}..n] = \mathcal{S}[0..n - q_r - \mathcal{P}]$, *then*:

(a) *If* $k > r$, *then* $q_1, \ldots, q_r$ *are period lengths of* $\mathcal{S}[0..n]$.
(b) *If* $k \leq r$, *then* $q_{r-k+2}, \ldots, q_r$ *are period lengths of* $\mathcal{S}[0..n]$. *In this case* q_{r-k+1} *may also be a period length of* $\mathcal{S}[0..n]$.

PROOF. Assume $\mathcal{S}[q_r + \mathcal{P}..n] = \mathcal{S}[0..n - q_r - \mathcal{P}]$. It is enough to show that q_i is a period length of $\mathcal{S}$ for $\max(r - k + 2, 1) \leq i \leq r$.

By the definition of the $\{q_i\}$ and $\{p_i\}$ sequences

$$\mathcal{S}[0..p_{r-i+1} + l_{\eta+1}] = \mathcal{S}[q_i..q_r + l_{\eta+1}], \tag{1}$$

since both substrings are covered by $r - i + 1$ occurrences of $\mathcal{S}[0..l_{\eta+1}]$. Also, since $r - i + 2 \leq k$,

$$\mathcal{S}[p_{r-i+2}..p_{r-i+2} + l_{\eta+1}] = \mathcal{S}[0..1_{\eta+1}]. \tag{2}$$

However, $n - q_r - \mathcal{P} < l_{\eta+1}$ and $\mathcal{S}[q_r + \mathcal{P}..n] = \mathcal{S}[0..n - q_r - \mathcal{P}]$. By taking prefixes of (2)

$$\mathcal{S}[q_r + \mathcal{P}..n] = \mathcal{S}[p_{r-i+2}..p_{r-i+2} + n - q_r - \mathcal{P}]. \tag{3}$$

By combining equalities (1) and (3), we get that $\mathcal{S}[0..n - q_i] = \mathcal{S}[q_i..n]$. □

The computation in stage η can be summarized as follows:

1. Compute the $\{q_i\}$ and $\{p_i\}$ sequences.
2. If $k \leq r$, check if q_{r-k+1} is a period length of $\mathcal{S}[0..n]$.
3. If $\mathcal{S}[q_r + \mathcal{P}..\mathcal{N}] = \mathcal{S}[0..\mathcal{N} - q_r - \mathcal{P}]$, then:
   (a) If $k > r$, then $q_1, \ldots, q_r$ are all period lengths of $\mathcal{S}[0..n]$.
   (b) If $k \leq r$, then $q_{r-k+2}, \ldots, q_r$ are all period lengths of $\mathcal{S}[0..n]$.

LEMMA 3.7. *Stage number* η *correctly computes* $\mathcal{P}[n - l_\eta + 1..n - l_{\eta+1}]$. *It takes* $O(\log\log l_\eta)$ *time and uses* $O(l_\eta)$ *operations.*

PROOF. Correctness of the algorithm follows from Lemmas 3.4–3.6. The two calls to a string-matching algorithm to compute the $\{q_i\}$ and $\{p_i\}$ sequences take $O(\log\log l_\eta)$ time and $O(l_\eta)$ operations if we use Breslauer and Galil's [8] string-matching algorithm. The sequences $\{q_i\}$ and $\{p_i\}$ can be represented by three integers which can be computed from the output of the string-matching algorithm (which is assumed to be a Boolean vector representing all occurrences) in constant time and $O(l_\eta)$ operations. Steps 2 and 3 can also be done in constant time and $O(l_\eta)$ operations. □

LEMMA 3.8. *Stage number η can be implemented in $O(\log\log_{\lceil 1+p/n\rceil}(2p/n)l_\eta)$ time on $(p/n)l_\eta$ processors if $p \geq n$.*

PROOF. The calls to Breslauer and Galil's [8], [9] string-matching algorithm take $O(\log\log_{\lceil 1+p/n\rceil}(2p/n)l_\eta)$ time if $p \geq n$ and $(p/n)l_\eta$ processors are available for stage number η. The rest of the work can be done in constant time since the number of processors is larger than l_η. □

4. A Lower Bound. Given a string $\mathcal{S}[0..n]$, we say that it has an initial palindrome of length k if $\mathcal{S}[i] = \mathcal{S}[k-i-1]$ for $i = 0, \ldots, k-1$. We modify the lower bound of [9] to a lower bound for determining whether a string $\mathcal{S}[0..n]$ has an initial palindrome whose length is larger than $n/2$. This lower bound holds even for deciding if the string $\mathcal{S}[0..n]$ has any initial palindrome other than the trivial initial palindrome of length one. Since there are some modifications in the details of the lower bound we repeat most steps of the proof. The missing proofs can be found in the original paper.

The model for which the lower bound is proved is similar to Valiant's parallel comparison tree model [21]. We assume the only access the algorithm has to the input string is by comparisons that check whether two symbols are equal or not. The algorithm is allowed p comparisons in each round, after which it can proceed to the next round or terminate with the answer. We give a lower bound on the minimum number of rounds necessary in the worst case. This lower bound holds even if an algorithm is allowed to perform order comparisons that can result in *less than*, *equal*, or *greater than* answers [9]. In the case of a general alphabet a CRCW-PRAM must use comparisons to solve any string problem and our lower bound holds.

We show a strategy for an adversary to answer $\frac{1}{4}\log\log n$ rounds of comparisons after which it still has the choice of fixing the input string $\mathcal{S}$ in two ways: in one, the resulting string has an initial palindrome whose length is larger than $n/2$, and in the other it does not have any such initial palindrome. This implies that any algorithm that claims to compute all initial palindromes in fewer rounds can be fooled.

We say that an integer k is a possible period length of $\mathcal{S}[0..n]$ if we can fix $\mathcal{S}$ consistently with answers to comparisons made in earlier rounds in such a way that k is a period length of $\mathcal{S}$. For such k to be a period length we need each residue class modulo k to be fixed to the same symbol, thus if $l \equiv j \bmod k$, then $\mathcal{S}[l] = \mathcal{S}[j]$.

We say that an integer k is a possible initial palindrome of $\mathcal{S}[0..n]$ if we can fix $\mathcal{S}$ consistently with answers to comparisons made in earlier rounds in such a way that $\mathcal{S}$ has an initial palindrome of length k. For such k to be an initial palindrome length we need that if $l \equiv k - j - 1$, then $\mathcal{S}[l] = \mathcal{S}[j]$.

For an integer k to be a period length and an initial palindrome length we need both conditions to hold. That is, if $l \equiv j \bmod k$ or if $l \equiv -j-1 \bmod k$, then $\mathcal{S}[l] = \mathcal{S}[j]$. We call such k a *palindromic-period length.*

Let $l = qk + r$ such that $0 \le r < k$. That is, $r = l \bmod k$. Define $\phi_k(l)$ as:

$$\phi_k(l) = \begin{cases} r & \text{if} \quad r < \left\lceil \frac{k}{2} \right\rceil, \\ k - r - 1 & \text{otherwise.} \end{cases}$$

Using this notation, k is a palindromic-period length of $\mathcal{S}$ if, for any two indices l and j that satisfy $\phi_k(l) = \phi_k(j)$, $\mathcal{S}[l] = \mathcal{S}[j]$. If $l \equiv j \bmod k$, we say that l and j are in the *same residue class* modulo k. If $l \equiv -j - 1 \bmod k$, we say that l and j are in *symmetric residue classes* modulo k. The function ϕ_k maps integers which are in the same residue class or in symmetric residue classes modulo k to the same value. We say that such integers are in the same *extended residue class* modulo k (this is an equivalence relation on the integers).

At the beginning of round i the adversary will maintain an integer k_i which is a possible palindromic-period length. The adversary answers the comparisons of round i in such a way that some k_{i+1} is a possible palindromic-period length and few symbols of $\mathcal{S}$ are fixed. Let $K_i = n^{1-4^{-(i-1)}}$. The adversary will maintain the following invariants which hold at the beginning of round number i:

1. k_i satisfies $\frac{1}{2}K_i \le k_i \le K_i$.
2. If $\mathcal{S}[l]$ was fixed, then, for every j such that $\phi_{k_i}(l) = \phi_{k_i}(j)$, $\mathcal{S}[j]$ was fixed to the same symbol. In other words, the entire extended residue class of l modulo k_i was fixed to the same symbol.
3. If a comparison was answered as equal, then both symbols compared were fixed to the same value.
4. If a comparison between positions l and j was answered as unequal, then:
   (a) l and j are in different extended residue classes modulo k_i. That is $\phi_{k_i}(l) \ne \phi_{k_i}(j)$.
   (b) If the symbols $\mathcal{S}[l]$ and $\mathcal{S}[j]$ were fixed, then they were fixed to different values.
5. The number of fixed symbols f_i satisfies $f_i \le K_i$.

Note that invariants 3 and 4 imply consistency of the answers given so far. Invariants 2–4 imply that k_i is a possible palindromic-length: if we fix all symbols in each unfixed extended residue class modulo k_i to a new value, using the same value within an extended residue class but different values for unrelated residue classes, we obtain a string which is consistent with the comparisons answered so far and has a palindromic-period length k_i. Such a string will have initial palindromes of all lengths which are integral multiples of k_i.

We start at round number *one* with $k_1 = K_1 = 1$. It is easy to see that the invariants hold initially. We show how to answer the comparisons of round i and how to choose k_{i+1} so that the invariants still hold. All multiples of k_i in the range $\frac{1}{2}K_{i+1} \cdots K_{i+1}$ are candidates for the new k_{i+1}. A comparison $\mathcal{S}[l] = \mathcal{S}[j]$ must be answered as equal if l and j are in the same extended residue class modulo k_{i+1}; that is, if $\phi_{k_{i+1}}(l) = \phi_{i+1}(j)$. We say that k_{i+1} *forces* this comparison.

LEMMA 4.1. *If p, q, $r \geq \sqrt{2n/k_i}$ and are relatively prime, then a comparison $\mathcal{S}[s] = \mathcal{S}[t]$ is forced by at most two of pk_i, qk_i, and rk_i.*

PROOF. A comparison can be forced by some pk_i because the indices of the compared symbols are in the same residue class or because they are in symmetric residue classes.

Assume s and t are in the same residue classes modulo pk_i and qk_i, thus $s \equiv t \bmod pk_i$ and $s \equiv t \bmod qk_i$. Then $s \equiv t \bmod pqk$. However, $pqk_i > n$ and $0 \leq s, t \leq n$ which implies that $s = t$; a contradiction.

If s and t are in symmetric residue classes modulo pk_i and qk_i, then $s \equiv -t - 1 \bmod pk_i$ and $s \equiv -t - 1 \bmod qk_i$. Then $s + t + 1 \equiv 0 \bmod pqk_i$. However, $pqk_i > 2n$ and $0 \leq s, t \leq n$; a contradiction.

The only remaining case is when s and t are in the same residue class modulo one of pk_i or qk_i and in symmetric residue classes modulo the other. In this case we go back to the third candidate rk_i and consider the pairs rk_i and pk_i, and rk_i and qk_i. One of these pairs is in one of the categories above; a contradiction to the existence of the third candidate. □

LEMMA 4.2. *The number of candidates for k_{i+1} which are prime multiples of k_i and satisfy $\frac{1}{2}K_{i+1} \leq k_{i+1} \leq K_{i+1}$ is greater than $K_{i+1}/(4K_i \log n)$. Each such candidate satisfies the condition of Lemma* 4.1.

LEMMA 4.3. *A candidate for k_{i+1} in the range $\frac{1}{2}K_{i+1} \cdots K_{i+1}$ that forces at most $(8nK_i \log n)/K_{i+1}$ comparisons.*

PROOF. By Lemma 4.2 there are at least $K_{i+1}/(4K_i \log n)$ such candidates which are prime multiples of k_i and satisfy the condition of Lemma 4.1. By Lemma 4.1 each of the n comparisons is forced by at most two of them. So the total number of comparisons forced by all these candidates is at most $2n$ (at most two comparisons forced by each candidate). Thus, there is a candidate that forces at most $(8nK_i \log n)/K_{i+1}$ comparisons. □

LEMMA 4.4. *For large enough n and $i \leq \frac{1}{4} \log \log n$, $1 + n^{2 \cdot 4^{-i}} 64 \log n \leq n^{3 \cdot 4^{-i}}$.*

LEMMA 4.5. *Assume the invariants hold at the beginning of round i and the adversary chooses k_{i+1} to be a candidate which forces at most $(8nK_i \log n)/K_{i+1}$ comparison. Then the adversary can answer the comparisons in round i so that the invariants also hold at the beginning of round $i + 1$.*

PROOF. By Lemma 4.3 such k_{i+1} exists. For each comparison that is forced by k_{i+1}, and is of the form $\mathcal{S}[l] = \mathcal{S}[j]$ where $\phi_{k_{i+1}}(l) = \phi_{k_{i+1}}(j)$, the adversary fixes the symbols in the residue class modulo k_{i+1} and its symmetric residue class (the extended residue class) to the same new value (a different value for different extended residue classes). The adversary answers comparisons between fixed symbols based on the values they are fixed to. All other comparisons involve

symbols that are not in the same extended residue class modulo k_{i+1} (and at least one unfixed symbol) and are always answered as unequal.

The extended residue classes form a partition of the set of integers between 0 and n. This partition is refined when we move from extended residue classes modulo k_i to extended residue classes modulo k_{i+1}. Since k_{i+1} is a multiple of k_i, the extended residue classes modulo k_i split. This means that if two indices are in different extended residue classes modulo k_i, then they are also in different extended residue classes modulo k_{i+1}; and if two indices are in the same extended residue class modulo k_{i+1}, then they are also in the same extended residue class modulo k_i.

We show that the invariants still hold.

1. The candidate we chose for k_{i+1} was in the required range.
2. Extended residue classes which were fixed in earlier rounds split into several extended residue classes, all are fixed. Any symbols that is fixed at this round causes its entire extended residue class modulo k_{i+1} to be fixed to the same value.
3. Equal answers of earlier rounds are not affected since the symbols involved were fixed to the same value by the invariants held before. Equal answers of this round are either between symbols which were fixed before this round to the same value or are within the same extended residue class modulo k_{i+1} and the entire extended residue class if fixed to the same value.
4. (a) Unequal answers of earlier rounds are between different extended residue classes modulo k_{i+1}, since extended residue classes modulo k_i split. Unequal answers of this round are between different extended residue classes, because comparisons within the same extended residue class modulo k_{i+1} are always answered as equal.
   (b) Unequal answers to comparisons that involve symbols which were fixed in earlier rounds are answered according to the symbol values and, therefore, these symbols must have been fixed to different values. Unequal answers to comparisons that involve symbols which are fixed at the end of this round and at least one fixed at this round are consistent since a new value is used for the symbols in each extended residue classes that is fixed.
5. We prove inductively that $f_{i+1} \le K_{i+1}$. We fix at most $(16nK_i \log n)/K_{i+1}$ residue classes modulo k_{i+1}. There are k_{i+1} such classes and each class has at most $\lceil n/k_{i+1} \rceil \le 2n/k_{i+1}$ elements. By Lemma 4.4 and simple algebra the number of fixed elements satisfies

$$
\begin{aligned}
f_{i+1} &\le f_i + \frac{2n}{k_{i+1}} \frac{16nK_i \log n}{K_{i+1}} \\
&\le K_i \left[1 + \left(\frac{n}{K_{i+1}} \right)^2 64 \log n \right] \\
&\le n^{1-4^{-(i-1)}} (1 + n^{2 \cdot 4^{-i}} 64 \log n) \\
&\le n^{1-4^{-i}} = K_{i+1}.
\end{aligned}
$$

□

THEOREM 4.6. *Any comparison-based parallel algorithm for finding the initial palindromes of a string $\mathscr{S}[0..n]$, using n comparisons in each round, requires $\frac{1}{4}\log\log n$ rounds.*

PROOF. Fix an algorithm which finds the initial palindromes of $\mathscr{S}$ and let the adversary described above answer the comparisons. After $i = \frac{1}{4}\log\log n$ rounds $f_{i+1}, k_{i+1} \leq n^{1-4^{-(1/4)\log\log n}} = n/2^{\sqrt{\log n}} \leq n/2$. The adversary can still fix $\mathscr{S}$ to have a palindromic-period length k_{i+1} by fixing the symbols in each remaining residue class modulo k_{i+1} and its symmetric residue class to the same value, and different values for each class. In this case any integral multiple of k_{i+1} is also an initial palindrome. Alternatively, the adversary can fix all unfixed symbols to different values. Note that this choice is consistent with all the comparisons answered so far by invariants 3 and 4, and the string does not have any initial palindrome of length larger than $n/2$. In fact, in the latter case, the string will not have any initial palindrome except the trivial initial palindrome of length one. Consequently, any algorithm which terminates in less than $\frac{1}{4}\log\log n$ rounds can be fooled.

This proof also gives a lower bound for computing the period length of a string. □

THEOREM 4.7. *Any comparison-based parallel algorithm for finding the initial palindromes of a string $\mathscr{S}[0..n]$ using p comparisons in each round requires at least $\Omega(\lceil n/p \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ rounds.*

5. Discussion. The algorithm described in this paper uses a string-matching procedure as a "black-box" that has a specific input–output functionality, without going into its implementation details. By using Breslauer and Galil's [8] string-matching algorithm, we obtained an optimal $O(\log\log n)$-time algorithm which is the best possible in the case of a general alphabet, as implied by a lower bound of Breslauer and Galil [9]. It is unknown if faster optimal string-matching algorithms exist in the case of a fixed alphabet. If such an algorithm exists it would immediately imply a faster algorithm for finding the periods. Note that a fast CRCW-PRAM implementation requires the computation of certain functions, such as the log function and powers of $\frac{2}{3}$ within the time and processor bounds.

Acknowledgments. We thank the referee for reading this paper carefully and providing many suggestions. We also thank Alberto Apostolico, Roberto Grossi, Jörg Keller, Kunsoo Park, and Laura Toniolo for comments on early versions of this paper.

References

[1] A. Apostolico and D. Breslauer. An optimal $O(\log \log n)$ time parallel algorithm for detecting all squares in a string. *SIAM J. Comput.*, to appear.

[2] A. Apostolico, D. Breslauer, and Z. Galil. Optimal parallel algorithms for periods, palindromes and squares. *Proc. 19th Internat. Colloq. on Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 623. Springer-Verlag, Berlin, 1992, pages 296–307.

[3] H. W. Bergerson. *Palindromes and Anagrams.* Dover. New York, 1973.

[4] R. P. Brent. Evaluation of general arithmetic expressions. *J. Assoc. Comput. Mach.*, 21:201–206, 1974.

[5] D. Breslauer. Efficient String Algorithmics. Ph.D. thesis, Department of Computer Science, Columbia University, New York, 1992.

[6] D. Breslauer. Fast parallel string prefix-matching. *Theoret. Comput. Sci.*, 137(2):269–278, 1995.

[7] D. Breslauer. Testing string superprimitivity in parallel. *Inform. Process. Lett.*, 49:5 235–241, 1994.

[8] D. Breslauer and Z. Galil. An optimal $O(\log \log n)$ time parallel string matching algorithm. *SIAM J. Comput.*, 19(6):1051–1058, 1990.

[9] D. Breslauer and Z. Galil. A lower bound for parallel string matching. *SIAM J. Comput.*, 21(5):856–862. 1992.

[10] M. Crochemore and W. Rytter. Usefulness of the Karp–Miller–Rosenberg algorithm in parallel computations on strings and arrays. *Theoret. Comput. Sci.*, 88:59–82, 1991.

[11] F. E. Fich, R. L. Radge, and A. Wigderson. Relations between concurrent-write models of parallel computation. *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, 1984, pages 179–189.

[12] M. J. Fischer and M. S. Paterson. Sring matching and other produces. In R. M. Karp, editor, *Complexity of Computation.* American Mathematical Society, Providence, RI, 1974, pages 113–125.

[13] G. Galil. Optimal parallel algorithms for string matching. *Inform. and Control.* 67:144–157, 1985.

[14] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.

[15] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. *Proc. 4th ACM Symp. on Theory of Computing*, 1972, pages 125–136.

[16] Z. Kedem, G. M. Landau, and K. Palem. Optimal parallel suffix–prefix matching algorithm and applications. *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, 1989, pages 388–398.

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

[18] M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA, 1983.

[19] R. C. Lyndon and M. P. Schutzenberger. The equation $a^m = b^n c^p$ in a free group. *Michigan Math. J.*, 9:289–298, 1962.

[20] G. Manacher. A new linear-time "On-line" algorithm for finding the smallest initial palindrome of a string. *J. Assoc. Comput. Mach.*, 22:346–351, 1975.

[21] L. G. Valiant. Parallelism in comparison models. *SIAM J. Comput.*, 4:348–355, 1975.

[22] U. Vishkin. Optimal parallel pattern matching in strings. *Inform. and Control*, 67:91–113, 1985.