Finding all periods and initial palindromes of a string in parallel*

Dany Breslauer† Columbia University

Zvi Galil Columbia University and Tel-Aviv University

CUCS-017-92

Abstract

An optimal $O(\log \log n)$ time CRCW-PRAM algorithm for computing all periods of a string is presented. Previous parallel algorithms compute the period only if it is shorter than half of the length of the string. This 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 palindromes by a modification of a previously known lower bound for finding the period of a string [3]. When p processors are available the bounds become $\Theta(\lceil \frac{n}{p} \rceil + \log \log_{\lceil 1+p/n \rceil} 2p)$.

1 Introduction

A string $S[0..n]$ has *a period* p if $S[i] = S[i+p]$ for $i = 0 \cdots n-p$. *The period* of S is defined as its shortest period. The period of a string is computed in linear time as a step in Knuth, Morris and Pratt's sequential string matching algorithm [6] and in optimal $O(\log \log n)$ parallel time on a CRCW-PRAM as a step in Breslauer and Galil's string matching algorithm [2]. By optimal

*Work partially supported by NSF Grant CCR-90-14605.

†Partially supported by an IBM Graduate Fellowship.

we mean that the time-processor product is linear in the length of the string. Note that a simple algorithm can compute all periods of S in constant time if n^2 processors are available; our goal is to design a fast optimal parallel algorithm. A recent lower bound by Breslauer and Galil [3] shows that the $O(\log\log n)$ bound is the best possible over a general alphabet, where only comparisons between symbols are allowed. However, Breslauer and Galil's [2] algorithm as well as an algorithm discovered by Vishkin [10] compute the period p only if $p \leq \lceil\frac{n}{2}\rceil$; knowing the fact that $p > \lceil\frac{n}{2}\rceil$ is sufficient to obtain good string matching algorithms. We show that given an optimal parallel algorithm for string matching, one can compute all periods, including those which are longer than half of the length of the input string, in the same processor and time bounds of the string matching algorithm. In particular Breslauer and Galil's [2] algorithm can be used to obtain an optimal $O(\log\log n)$ time CRCW-PRAM algorithm that computes the exact period of a string.

A *palindrome* is a string that reads the same forward and backward. Formaly, a string $S[1..k]$ is a *palindrome* if $S[i] = S[k+1-i]$ for $i = 1..k$. A string $S[1..n]$ is said to have an *initial palindrome* of length k if the prefix $S[1..k]$ is a palindrome. We show how the algorithm that finds all periods of a string can be used also to find all initial palindromes in the same time and processor bounds. We prove also a matching lower bound for this problem under the assumption of a general alphabet. The model for which the lower bound is proved is similar to Valiant's parallel comparison tree model [9]. 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. We show also that our bound holds even if the algorithm is allowed to perform order comparison that can result in a *less than*, *equal* or *greater than* answers. In the case of a general alphabet a CRCW-PRAM must use comparisons to solve any string problem and our lower bound holds.

The paper is organized as follows: In Section 2 we describe 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 given a string $S[0..n]$ will compute all periods of $S[0..n]$. The output of the algorithm will be a Boolean array $P[1..n]$ such that $P[i] = \text{true}$ *iff* i is a period of S.

One of the major issues in the design of a 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 (Brent): 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 will describe an $O(\log\log n)$ time algorithm using $\frac{n}{\log\log n}$ processors. Some of the steps in our algorithm are described as constant time steps using n processors.

We will prove the following theorem:

Theorem 2.2: There exists an algorithm to compute $P[1..n]$ that takes $O(\log\log n)$ time using $\frac{n}{\log\log n}$ processors. If p processors are available the algorithm takes $O(\lceil \frac{n}{p} \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ time.

Corollary 2.3: The exact period of a string S can be computed in the same time and processor bounds.

Proof: The period of S is the smallest i such that $P[i]$ is true. We use a technique of Fich, Ragde and Weigderson [4] 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 $O(\frac{n}{p})$ time on p processors.) $\square$

Corollary 2.4: All initial palindromes of a string S 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 [5]. $\square$

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 will proceed in independent stages which are all computed simultaneously and described in the next section. In stage number η, $0 \leq \eta < m$, we will compute 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.

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 will be 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 [2] parallel string matching algorithm.

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

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

3 A single stage

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

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

If there were no occurrences found, the string S has no period 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 $S[0..l_{\eta+1}]$ in $S[0..l_\eta - 1]$. Let p_i, $i = 1..k$, denote the indices of all these occurrences (note that $p_1 = 0$).

If there was only one occurrence of $S[0..l_{\eta+1}]$ in $S[n - l_\eta + 1..n]$, it can be verified to be a period 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 of S.

Lemma 3.1 (Lyndon and Schutzenberger [7]): If a string of length m has two periods of length p and q and $p + q \leq m$, then it has also a period of length $\gcd(p, q)$.

Lemma 3.2: If a string $A[1..l]$ has period 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.

Proof: Assume $k \geq 2$. We prove that $p = p_{i+1} - p_i$ for $i = 1 \cdots k - 1$. The string $A[1..l]$ has periods p and $q = p_{i+1} - p_i$. Since $p \leq q \leq \lceil \frac{l}{2} \rceil$, by Lemma 3.1 it has also a period of length $gcd(p, q)$. But p is the shortest period so $p = gcd(p, q)$ and p must divide q. The string $B[p_i..p_{i+1} + l - 1]$ has period p. If $q > p$ then there must be another occurrence of A at position $p_i + p$ of B; a contradiction. $\Box$

Lemma 3.3: The sequences $\{p_i\}$ and $\{q_i\}$ form an arithmetic progression with difference $\mathcal{P}$, where $\mathcal{P}$ is the period of $S[0..l_{\eta+1}]$.

Proof: The sequences p_i and q_i are indices of occurrences of a string of length $l_{\eta+1} + 1$ in strings 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 q_i's form an arithmetic progression with a difference $\mathcal{P}$, the period of $S[0..l_{\eta+1}]$. $\Box$

The sequences $\{p_i\}$ and $\{q_i\}$ can be represented using three integers (each): the start of the sequence, the difference, and the length of each sequence. This representation can be easily obtained from the output of the string matching algorithm in constant time and l_η processors.

Some of the q_i's can be ruled out of being periods of S immediately as we show in the following lemma.

Lemma 3.4: If $k < r$ then q_i is not a period of $S[0..n]$ for $1 \leq i \leq r - k$.
Proof: Assume q_i is a period of S and $1 \leq i \leq r-k$. In this case $S[q_i..n] = S[0..n - q_i]$. The string $S[q_i..n]$ has $r - i + 1 > k$ occurrences of $S[0..l_{\eta+1}]$, which are $q_i \cdots q_r$. But $S[0..n - q_i]$ has only k occurrences of $S[0..l_{\eta+1}]$; contradiction. $\square$

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

1. If $S[q_r + \mathcal{P}..\mathcal{N}] \neq S[0..\mathcal{N} - q_r - \mathcal{P}]$, and $\mathcal{N} = \min(n, q_r + \mathcal{P} + l_{\eta+1})$ we call it **a mismatch**.

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

Lemma 3.5 (a mismatch): If $S[q_r + \mathcal{P}..\mathcal{N}] \neq S[0..\mathcal{N} - q_r - \mathcal{P}]$ then, S has at most one period in the range computed by this stage. This only possible period may exist if $k \leq r$ and it is q_{r-k+1}.
Proof: By Lemma 3.4 all q_i, $1 \leq i < r-k+1$, are not periods. Assume q_i is a period and $i > r-k+1$, then $S[q_i..n] = S[0..n-q_i]$. Since $r-i+2 \leq k$ and $p_j = (j-1)\mathcal{P}$, also $S[q_r + \mathcal{P}..\mathcal{N}] = S[p_{r-i+2}..\mathcal{N} - q_i]$. By the assumption of a mismatch $S[q_r + \mathcal{P}..\mathcal{N}] \neq S[0..\mathcal{N} - q_r - \mathcal{P}]$. So $S[p_{r-i+2}..\mathcal{N} - q_i] \neq S[0..\mathcal{N} - q_r - \mathcal{P}]$. But $S[p_{r-i+2}..p_{r-i+2} + l_{\eta+1}] = S[0..l_{\eta+1}]$ and also $\mathcal{N} - q_r - \mathcal{P} \leq l_{\eta+1}$; contradiction. $\square$

Lemma 3.6 (an overflow): If $S[q_r + \mathcal{P}..n] = S[0..n - q_r - \mathcal{P}]$ then:

a. If $r < k$ then $q_1, .., q_r$ are periods of S.

b. If $r \geq k$ then $q_{r-k+2}, .., q_r$ are periods of S. In this case q_{r-k+1} can also be a period of S.

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

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

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

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

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

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

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

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

The computation in stage η can be summarized as follows:

1. Compute the $\{q_i\}$ and $\{p_i\}$ sequences.
2. If $k \le r$, check if q_{r-k+1} is a period of S.
3. If $S[q_r+\mathcal{P}..\mathcal{N}] = S[0..\mathcal{N}-q_r-\mathcal{P}]$ then,

   a. If $r < k$, then $q_1, .., q_r$ are all periods of S.

   b. If $r \ge k$, then $q_{r-k+2}, .., q_r$ are all periods of S.

Lemma 3.7: Stage number η is correct and it takes $O(\log\log l_\eta)$ time and $O(l_\eta)$ operations.

Proof: Correctness of the algorithm follows from Lemmas 3.4, 3.5 and 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 [2] 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 assume to be a Boolean vector representing all occurrences) in constant time and $O(l_\eta)$ operations. Steps *2* and *3* can be done also in constant time and $O(l_\eta)$ operations. $\square$

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

Proof: The calls to Breslauer and Galil's [2, 3] string matching algorithm take $O(\log\log_{\lceil 1+p/n\rceil} \frac{2p}{n} l_\eta)$ time if $p \ge n$ and $\frac{p}{n} l_\eta$ 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. $\square$

4 A lower bound

Given a string $S[0..n]$, we say that it has an initial palindrome of length k if $S[i] = S[k-i-1]$ for $i = 0, ..., k-1$. We modify the lower bound of

Breslauer and Galil [3] to a lower bound for determining whether a string $S[0..n]$ has a an initial palindrome of length larger than $\frac{n}{2}$. Since there are minor modification in the details of the lower bound we include the complete proofs.

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 S in two ways: in one the resulting string has an initial palindrome of length larger than $\frac{n}{2}$ and in the other it does not have any such initial palindrome. This implies that any algorithm which terminates in less rounds can be fooled.

We say that an integer k is a possible period length of $S[0..n]$ if we can fix S consistently with answers to previous comparisons in such a way that k is a period of 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 $l \equiv j \, mod \, k$ then $S[l] = S[j]$.

We say that an integer k is a possible initial palindrome of $S[0..n]$ if we can fix S consistently with answers to previous comparisons in such a way that 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 $S[l] = S[j]$.

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

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

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

Using this notation, k is a palindromic-period length of S if for any two indices l and j that satisfy $\phi_k(l) = \phi_k(j)$ also $S[l] = S[j]$. If $l \equiv j \, mod \, k$ we say that l and j are in the *same residue class* modulo k. If $l = -j - 1 \, mod \, k$ 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 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 $\frac{1}{2}K_i \leq k_i \leq K_i$.

2. If $S[l]$ was fixed then for every j such that $\phi_{k_i}(l) = \phi_{k_i}(j)$, $S[j]$ 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 $\phi_{k_i}(l) \neq \phi_{k_i}(j)$.

   b. if the symbols were fixed then they were fixed to different values.

5. The number of fixed symbols f_i satisfies $f_i \leq K_i$.

Note that invariants 3 and 4 imply consistency of the answers given so far. Invariants 2, 3 and 4 imply that k_i is a possible palindromic-period length: if we fix all symbols in each unfixed extended residue class modulo k_i to a new symbol, using the same symbol within an extended residue class but different symbols for unrelated residue classes, we obtain a string consistent with the answers given so far that 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 1 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 $\frac{1}{2}K_{i+1} \ldots K_{i+1}$ are candidates for the new k_{i+1}. A comparison $S[l] = S[j]$ must be answered as equal if l and j are in the same extended residue class modulo k_{i+1}, that is, if $\phi_{k_{i+1}}(l) = \phi_{k_{i+1}}(j)$. We say that k_{i+1} *forces* this comparison.

Theorem 4.1 (see [8]): For large enough n, the number of primes between 1 and n denoted by $\pi(n)$ satisfies, $\frac{n}{\ln n} \leq \pi(n) \leq \frac{5}{4}\frac{n}{\ln n}$.

Corollary: The number of primes between $\frac{1}{2}n$ and n is greater than $\frac{1}{4}\frac{n}{\log n}$.

Lemma 4.2: If $p, q, r \geq \sqrt{\frac{2n}{k_i}}$ and are relatively prime, then a comparison $S[s] = S[t]$ is forced by at most two of pk_i and qk_i and rk_i.
Proof: A comparison can be forced by some pk_i because indices of the compared symbols are in the same residue class or because there are in symmetric residue classes.

Assume s and t are in the same residue classes modulo pk_i and qk_i, thus $s \equiv t \, mod \, pk_i$ and $s \equiv t \, mod \, qk_i$. Then also $s \equiv t \, mod \, pqk_i$. But $pqk_i \geq n$ and $0 \leq s, t \leq n$ what implies that $s = t$; a contradiction.

If s and t are in symmetric residue classes modulo pk_i and qk_i then $s \equiv -t-1 \, mod \, pk_i$ and $s \equiv -t-1 \, mod \, qk_i$. Then also $s+t+1 \equiv 0 \, mod \, pqk_i$. But $pqk_i > 2n$ 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 and qk_i and in symmetric residue classes modulo the other. In this case we look at the third candidate rk_i and consider the pairs rk_i and pk_i or rk_i and qk_i. One of these pairs is in one of the categories above; a contradiction to the existance of the third candidate. $\square$

Lemma 4.3: 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 $\frac{K_{i+1}}{4K_i \log n}$. Each such candidate satisfies the condition of Lemma 4.2.
Proof: These candidates are of the form pk_i for prime p. The number of such prime values of p can be estimated using the corollary to Lemma 4.1. It is at least

$$\frac{1}{4}\frac{K_{i+1}}{k_i \log \frac{K_{i+1}}{k_i}} \geq \frac{K_{i+1}}{4K_i \log n}.$$

Each one of these candidates also satisfies the condition of Lemma 4.2 since $k_i \leq K_i$, $pk_i \geq \frac{K_{i+1}}{2}$ and

$$p^2 \geq \frac{1}{k_i}\frac{K_{i+1}^2}{4K_i} = \frac{1}{k_i}\frac{n^{2-2\cdot 4^{-i}}}{4n^{1-4^{-(i-1)}}} = \frac{n}{k_i}\frac{1}{4}n^{2\cdot 4^{-i}} \geq \frac{2n}{k_i}. \ \square$$

Lemma 4.4: There exists a candidate for k_{i+1} in the range $\frac{1}{2}K_{i+1} \ldots K_{i+1}$ that forces at most $\frac{8nK_i \log n}{K_{i+1}}$ comparisons.
Proof: By Lemma 4.3 there are at least $\frac{K_{i+1}}{4K_i \log n}$ such candidates which are prime multiples of k_i and satisfy the condition of Lemma 4.2. By Lemma

4.2 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 2 comparisons forced by each candidate). Thus, there is a candidate that forces at most $\frac{8nK_i \log n}{K_{i+1}}$ comparisons. □

Lemma 4.5: For n large enough and $i \leq \frac{1}{4}\log\log n$, $1 + n^{2\cdot 4^{-i}} 64 \log n \leq n^{3\cdot 4^{-i}}$.

Proof: For n large enough,

$$\log\log(1 + 64\log n) < \frac{1}{2}\log\log n = (1 - \frac{2}{4})\log\log n$$

$$\log(1 + 64\log n) < 4^{-\frac{1}{4}\log\log n}\log n$$

$$1 + 64\log n < n^{4^{-\frac{1}{4}\log\log n}} \leq n^{4^{-i}},$$

from which the lemma follows. □

Lemma 4.6: 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 $\frac{8nK_i \log n}{K_{i+1}}$ comparisons. 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 4.4 such k_{i+1} exists. For each comparison that is forced by k_{i+1} and is of the form $S[l] = S[j]$ where $\phi_{k_{i+1}}(l) = \phi_{k_{i+1}}(j)$ the adversary fixes the residue class modulo k_{i+1} and its symmetric residue class (the extended residue class) to the same new symbol (a different symbol for different extended residue classes). The adversary answers comparisons between fixed symbols based on the value 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 before split into several extended residue classes, all are fixed. Any symbol fixed at this round causes its entire extended residue class modulo k_{i+1} to be fixed to the same symbol.

3. Equal answers of previous 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 to the same value or are within the same extended residue class modulo k_{i+1} and the entire extended residue class is fixed to the same value.

4. **a.** Unequal answers of previous 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 which involve symbols which were fixed before this round are consistent because fixed values dictate the answers to the comparisons. Unequal answers which involve symbols that are fixed at the end of this round and at least one was fixed at this round are consistent since a new symbol is used for each extended residue classes fixed.

5. We prove inductively that $f_{i+1} \leq K_{i+1}$. We fix at most $\frac{16nK_i \log n}{K_{i+1}}$ residue classes modulo k_{i+1}. There are k_{i+1} such classes and each class has at most $\lceil \frac{n}{k_{i+1}} \rceil \leq \frac{2n}{k_{i+1}}$ elements. By Lemma 4.5 and simple algebra the number of fixed elements satisfies:

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

$$\leq \ n^{1-4^{-i}} = K_{i+1}. \ \Box$$

Theorem 4.7: Any comparison-based parallel algorithm for finding the initial palindromes of a string $S[0..n]$ using n comparisons in each round requires $\frac{1}{4}\log\log n$ rounds.
Proof: Fix an algorithm which finds the initial palindromes of S and let the adversary described above answer the comparisons. After $i = \frac{1}{4}\log\log n$ rounds $f_{i+1}, k_{i+1} \leq n^{1-4^{-\frac{1}{4}\log\log n}} = \frac{n}{2^{\sqrt{\log n}}} \leq \frac{n}{2}$. The adversary can still fix S to have a palindromic-period length k_{i+1} by fixing each remaining residue class modulo k_{i+1} and its symmetric residue class to the same symbol, different symbol 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 symbols. Note that this choice is consistent with all the the comparisons answered so far by invariants 3 and 4, and the string does not have any initial palindrome of length larger than $\frac{n}{2}$. In fact, in the latter case, the string will not have any initial palindrome except the trivial of length one. Consequently, any algorithm which terminates in less than $\frac{1}{4}\log\log n$ rounds can be fooled.

This proof actualy gives a lower bound also for computing the period length. $\Box$

Theorem 4.8: The lower bound holds also for order comparisons.
Proof: The adversary gradually defines the linear order of the symbols. He does it in such a way that the answers to comparisons in round i are determined at the round or before. The order is determined by a lexicographic order on a name given to each symbol and is extended for unfixed symbols at each round.

At round i, after k_{i+1} is chosen and before fixing new symbols, all names of unfixed symbols $S[l]$ are extended to be $< \phi_{k_2}(l), \phi_{k_3}(l), ..., \phi_{k_{i+1}}(l) >$. If a symbol $S[l]$ was fixed at round $j \leq i$ its entire extended residue class and its symmetric residue class are assigned the the name $< \phi_{k_2}(l), \phi_{k_3}(l), ..., \phi_{k_{j+1}}(l) >$ which is never changed. Note that all other names can not have this name as a prefix. If two symbols compared at round i are in the same or symmetric residue classes modulo k_{i+1}, then they were fixed at this round or before and given the same name. Symbols which are compared at round i and are in different extended residue classes modulo k_{i+1} have different name unless they were fixed earlier to the same value. The result of the comparison ($<$,

$=, >)$ is already determined even if the full final name of the symbols is not defined yet. At the end, if the extended residue classes modulo k_{i+1} are fixed in such a way that k_{i+1} is the palindromic-period length, then the full names are given as before. If all symbols are fixed to be different, the name given to unfixed position l is $< \phi_{k_2}(l), \phi_{k_3}(l), ..., \phi_{k_{i+1}}(l), l >$. $\Box$

Theorem 4.9: Any comparison-based parallel algorithm for finding the initial palindromes of a string $S[0..n]$ using p comparisons, $n \leq p \leq n^2$, in each round requires at least $\Omega(\log \log_{\frac{2p}{n}} p)$ rounds.

Proof: We change n to p in the appropriate places of the proof. In particular we choose $K_i = p^{1-4^{-(i-1)}}$. The adversary can go on as long as $K_i \leq \frac{n}{2}$. By simple arithmetic $\Omega(\log \log_{\frac{2p}{n}} p)$ rounds are necessary. $\Box$

5 Discussion

The algorithm described in this paper uses a string matching procedure as a "black-box". Using Breslauer and Galil's [2] string matching algorithm would result in 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 [3]. It is unknown if a faster algorithm exists for string matching in the case of a fixed alphabet. If such 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 bounds.

6 Acknowledgments

We would like to thank Alberto Apostolico, Roberto Grossi, Kunsoo Park and Laura Toniolo for comments on early versions of this paper.

References

[1] Brent, R. P. (1974), The parallel evaluation of general arithmetic expressions, *J. ACM 21*, 201-206.

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

[3] Breslauer, D. and Galil, Z. (1991), A lower bound for parallel string matching, *Proc. 23nd ACM Symp. on Theory of Computation*, 439-443.

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

[5] Fischer, M. J. and Paterson, M. S. (1974), String-Matching and other products, *SIAM-AMS proceedings, Vol 7*, 113-125.

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

[7] 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.

[8] Rosser, J. B. and Schoenfeld, L. (1962), Approximate formulas for some functions of prime numbers, *Illinois Journal of Mathematics*, 6:64-94.

[9] Valiant, L. G. (1975), Parallelism in comparison models, *SIAM J. of comput. 4*, 348-355.

[10] Vishkin, U. (1985), Optimal parallel pattern matching in strings, *Information and Control 67*, 91-113.