Parallel Detection of all Palindromes in a String

Alberto Apostolico* Dany Breslauer† Zvi Galil‡

Abstract

This paper presents two efficient concurrent-read concurrent-write parallel algorithms that find all palindromes in a given string:

1. An $O(\log n)$ time, n-processor algorithm over general alphabets. In case of constant size alphabets the algorithm requires only $n/\log n$ processors, and thus achieves an optimal-speedup.
2. An $O(\log\log n)$ time, $n\log n/\log\log n$-processor algorithm over general alphabets. This is the fastest possible time with the number of processors used.

These new results improve on the known parallel palindrome detection algorithms by using smaller auxiliary space and either by making fewer operations or by achieving a faster running time.

1 Introduction

Palindromes are symmetric strings that read the same forward and backward. Palindromes have been studied for centuries as word puzzles and more recently have found several important uses in formal languages and computability theory.

Formally, a non-empty string w is a palindrome if $w = w^R$, where w^R denotes the string w reversed. It is convenient to distinguish between even length

*Computer Science Department, Purdue University, West Lafayette, IN 47907, and Dipartimento di Elettronica e Informatica, Università di Padova, 35131 Padova, Italy. Partially supported by NSF Grants CCR-89-00305 and CCR-92-01078, by NATO Grant CRG 900293 and by the National Research Council of Italy.

†Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche, Via S. Maria 46, 56126 Pisa, Italy. Partially supported by the European Research Consortium for Informatics and Mathematics postdoctoral fellowship. Part of this work was done while visiting at the Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France.

‡Computer Science Department, Columbia University, New York, NY 10027, and Computer Science Department, Tel-Aviv University, Ramat-Aviv 69978, Israel. Partially supported by NSF Grants CCR-90-14605 and CISE Institutional Infrastructure Grant CDA-90-24735.

palindromes that are strings of the form vv^R and odd length palindromes that are strings of the form vav^R, where a is a single alphabet symbol.

Given a string $\mathcal{S}[1..n]$, we say that there is an even palindrome of radius $\mathcal{R}$ *centered at* position k of $\mathcal{S}[1..n]$, if $\mathcal{S}[k-i] = \mathcal{S}[k+i-1]$ for $i = 1, \cdots, \mathcal{R}$. We say that there is an odd palindrome of radius $\hat{\mathcal{R}}$ *centered on* position k of $\mathcal{S}[1..n]$, if $\mathcal{S}[k-i] = \mathcal{S}[k+i]$ for $i = 1, \cdots, \hat{\mathcal{R}}$. The radius $\mathcal{R}$ (or $\hat{\mathcal{R}}$) is maximal if there is no palindrome of radius[1] $\mathcal{R}+1$ centered at the same position. In this paper we are interested in computing the maximal radii $\mathcal{R}[k]$ and $\hat{\mathcal{R}}[k]$ of the even and the odd palindromes which are centered at all positions k of $\mathcal{S}[1..n]$.

Manacher [17] discovered an elegant linear-time on-line sequential algorithm that finds all initial palindromes in a string. Galil [11] and Slisenko [18] presented real-time initial palindrome recognition algorithms for multi-tape Turing machines. It is interesting to note that the existence of efficient algorithms that find initial palindromes in a string was also implied by theoretical results on fast simulation [6, 10]. Knuth, Morris and Pratt [15] gave another linear-time algorithm that finds all initial palindromes in a string.

A closer look at Manacher's algorithm shows that it not only finds the initial palindromes, but it also computes the maximal radii of palindromes centered at all positions of the input string. Thus Manacher's algorithm solves the problem considered in this paper in linear time.

A parallel algorithm is said to be *optimal*, or to achieve an *optimal-speedup*, if its time-processor product, which is the total number of operation performed, is equal to the running time of the fastest sequential algorithm. Note that there exists a trivial constant-time CRCW-PRAM algorithm that finds all palindromes in a string using $O(n^2)$ processors. However, the large number of processors leaves much to be desired.

Fischer and Paterson [9] noticed that any string matching algorithm that finds all overhanging occurrences of a string in another can also find all initial palindromes. This observation has been used by Apostolico, Breslauer and Galil [1] to construct an optimal $O(\log\log n)$ time parallel algorithm that finds all initial palindromes in strings over general alphabets, improving an $O(\log n)$ time non-optimal algorithm of Galil [12]. Breslauer and Galil [5] show that any parallel algorithm that finds initial palindromes in strings over general alphabets requires $\Omega(\lceil n/p \rceil + \log\log_{\lceil 1+p/n \rceil} 2p)$ time using p processors. Thus, the fastest possible optimal parallel algorithm that finds initial palindromes must take $\Omega(\log\log n)$ time and this is the time required even with $n \log n$ processors.

Crochemore and Rytter [7] discuss a general framework for solving string problems in parallel. (Similar results have been discovered by Kedem, Landau and Palem [14].) Most problems they consider, including the problem of detecting

[1] For the convenience of our notation, we sometimes refer to indices in $\mathcal{S}[1..n]$ that are out of the defined range. It is agreed that all undefined symbols are distinct and different from the symbols in $\mathcal{S}[1..n]$.

all palindromes in a string, have $O(\log n)$ time, n-processor CRCW-PRAM algorithms. However, their method uses $O(n^{1+\epsilon})$ space and requires that the input symbols are drawn from an ordered alphabet, an unnecessary restriction in the palindrome detection problem.

This paper presents two new CRCW-PRAM algorithms for detecting all palindromes in a string. Both algorithms have the same time-processor product as the Crochemore-Rytter algorithm, use linear space and work under the general alphabet assumption, where the only access they have to the input symbols is by pairwise comparisons that determine if two symbols are equal.

1. The first algorithm takes $O(\log n)$ time using n processors. If the alphabet size is bounded by a constant, then the number of processors can be reduced to $n/\log n$, making the algorithm optimal.

2. The second algorithm takes $O(\log \log n)$ time using $n \log n/\log \log n$ processors. This algorithm is the fastest possible with the number of processors used since it takes at least this time to find the initial palindromes.

The paper is organized as follows. Section 2 overviews some parallel algorithms and tools that are used in the new algorithms. Section 3 gives important properties of periods and palindromes. Sections 4 and 5 describe the new algorithms. Concluding remarks and open problems are listed in Section 6.

2 The CRCW-PRAM Model

The algorithms described in this paper are for the concurrent-read concurrent-write parallel random access machine model. We use the weakest version of this model called the *common CRCW-PRAM*. In this model many processors have access to a shared memory. Concurrent read and write operations are allowed at all memory locations. If several processors attempt to write simultaneously to the same memory location, it is assumed that they always write the same value.

Our palindrome detection algorithms use an algorithm of Fich, Ragde and Wigderson [8] to compute the minima of n integers from the range $1, \cdots, n$, in constant time using an n-processor CRCW-PRAM. The second algorithm uses Breslauer and Galil's [4] parallel string matching algorithm that takes $O(\log \log n)$ time using an $n/\log \log n$-processor CRCW-PRAM.

One of the major issues in the design of PRAM algorithms is the assignment of processors to their tasks. In this paper, the assignment can be done using standard techniques and the following general theorem.

Theorem 2.1 *(Brent [3]) 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.*

3 Periods and Palindromes

Periods are regularities of strings that are exploited in many efficient string algorithms.

Definition 3.1 *A string $\mathcal{S}$ has* a period *u if $\mathcal{S}$ is a prefix of u^k for some large enough k. The shortest period of a string $\mathcal{S}$ is called* the period *of $\mathcal{S}$. Alternatively, a string $\mathcal{S}[1..m]$ has a period of length π if $\mathcal{S}[i] = \mathcal{S}[i+\pi]$, for $i = 1, \cdots, m-\pi$.*

Lemma 3.2 *(Lyndon and Schutzenberger [16]) If a string of length m has two periods of lengths p and q and $p + q \leq m$, then it also has a period of length* $\gcd(p, q)$.

Throughout the paper, we discuss only the detection of even palindromes. If interested also in the odd palindromes, one can convert the input string $\mathcal{S}[1..n]$ into a string $\hat{\mathcal{S}}[1..2n]$ that is obtained by doubling each symbol of the original string. It is not difficult to verify that the string $\hat{\mathcal{S}}[1..2n]$ has even palindromes that correspond to each odd and even palindrome of $\mathcal{S}[1..n]$. Thus, the palindrome detection algorithms can be presented with the string $\hat{\mathcal{S}}[1..2n]$ as their input, while their output is considered in the context of the original string $\mathcal{S}[1..n]$. Note that an odd palindrome in $\hat{\mathcal{S}}[1..2n]$ consists of equal symbols.

The palindrome detection algorithms use the following lemmas that allow them to handle efficiently long palindromes that are centered close to each other. The lemmas concern only even palindromes, but there exist similar versions for odd palindromes.

Lemma 3.3 *Assume that the string $\mathcal{S}[1..n]$ contains two even palindromes whose radii are at least r centered at positions k and l, such that $k < l$ and $l - k \leq r$. Then the substring $\mathcal{S}[k - r..l + r - 1]$ is periodic with period length $2(l - k)$.*

Proof: If $1 \leq i \leq r$, then

$$\mathcal{S}[k-i] = \mathcal{S}[k+i-1] = \mathcal{S}[l-(l-k)+i-1] = \mathcal{S}[l+(l-k)-i] = \mathcal{S}[k+2(l-k)-i]$$

and

$$\mathcal{S}[l+i-1] = \mathcal{S}[l-i] = \mathcal{S}[k+(l-k)-i] = \mathcal{S}[k-(l-k)+i-1] = \mathcal{S}[l-2(l-k)+i-1],$$

establishing that $\mathcal{S}[k - r..l + r - 1]$ is periodic with period length $2(l - k)$. □

Lemma 3.4 *Assume that the string $\mathcal{S}[1..n]$ contains an even palindrome whose radius is at least r centered at position k. Furthermore, let $\mathcal{S}[\epsilon_L..\epsilon_R]$ be the maximal substring that contains $\mathcal{S}[k-r..k+r-1]$ and is periodic with period length $2r$. Namely, $\mathcal{S}[i] = \mathcal{S}[i+2r]$ for $i = \epsilon_L, \cdots, \epsilon_R - 2r$, and $\mathcal{S}[\epsilon_L - 1] \neq \mathcal{S}[\epsilon_L + 2r - 1]$ and $\mathcal{S}[\epsilon_R + 1] \neq \mathcal{S}[\epsilon_R - 2r + 1]$.*

Then the maximal radii of the palindromes centered at positions $c = k + lr$, for integral positive or negative values of l, such that $\epsilon_L \leq c \leq \epsilon_R$, are given as follows:

- *If $c - \epsilon_L \neq \epsilon_R - c + 1$, then the radius is exactly $\min(c - \epsilon_L, \epsilon_R - c + 1)$.*
- *If $c - \epsilon_L = \epsilon_R - c + 1$, then the radius is larger than or equal to $c - \epsilon_L$. The radius is exactly $c - \epsilon_L$ if and only if $\mathcal{S}[\epsilon_L - 1] \neq \mathcal{S}[\epsilon_R + 1]$.*

Proof: By the periodicity of $\mathcal{S}[\epsilon_L..\epsilon_R]$, $\mathcal{S}[i] = \mathcal{S}[j]$ if $\epsilon_L \leq i, j \leq \epsilon_R$ and $i \equiv j \pmod{2r}$. Combined with the existence of the even palindrome with radius r centered at position k, we get that $\mathcal{S}[i] = \mathcal{S}[j]$ if $\epsilon_L \leq i, j \leq \epsilon_R$ and $i + j \equiv 2k - 1 \pmod{2r}$.

Consider the even palindrome centered at some position $c = k + lr$, for integral positive or negative values of l, such that $\epsilon_L \leq c \leq \epsilon_R$. Since $(c-i) + (c+i-1) \equiv 2k-1 \pmod{2r}$, we get that $\mathcal{S}[c-i] = \mathcal{S}[c+i-1]$ for $i = 1, \cdots, \min(c - \epsilon_L, \epsilon_R - c + 1)$, establishing that the radius of the palindrome centered at position c is at least $\min(c - \epsilon_L, \epsilon_R - c + 1)$.

If $c - \epsilon_L < \epsilon_R - c + 1$, then $\mathcal{S}[c - (c - \epsilon_L + 1)] \neq \mathcal{S}[c + (c - \epsilon_L + 1) - 1]$ since $\mathcal{S}[\epsilon_L - 1] \neq \mathcal{S}[\epsilon_L + 2r - 1]$ and $\mathcal{S}[2c - \epsilon_L] = \mathcal{S}[\epsilon_L + 2r - 1]$, establishing that the radius is exactly $c - \epsilon_L$. Similar arguments hold if $c - \epsilon_L > \epsilon_R - c + 1$.

Finally, if $c - \epsilon_L = \epsilon_R - c + 1$, then it is clear that the radius is larger than $c - \epsilon_L$ if and only if $\mathcal{S}[\epsilon_L - 1] = \mathcal{S}[\epsilon_R + 1]$. □

4 An $O(\log n)$ time algorithm

Theorem 4.1 *There exists an algorithm that computes the radii of all even palindromes in a string $\mathcal{S}[1..n]$ in $O(\log n)$ time using n processors and linear space.*

Proof: The algorithm consists of $\lfloor \log n \rfloor - 1$ steps. In step number η, $0 \leq \eta \leq \lfloor \log n \rfloor - 2$, the input string $\mathcal{S}[1..n]$ is partitioned into consecutive blocks of length 2^η. (Only palindrome centers are partitioned. The palindromes themselves may overlap.) The algorithm proceeds simultaneously in all $\lfloor n/2^\eta \rfloor$ blocks. It takes constant time and makes $O(2^\eta)$ operations in each block. Therefore, each step takes constant time using n processors.

The description below concentrates on a single block. The ideal situation is when the radii of all palindromes that are centered in the block are determined by the end of the step. However, this will not always be the case. The algorithm maintains the following invariant at the completion of step number η:

The palindromes whose radii are determined are exactly those whose radii are smaller than $2^{\eta+2}$.

The main observation is that at the beginning of step number η, the position of all undetermined radii in the block form an arithmetic progression. Let $c_1 < c_2 < \cdots < c_l$ be the positions of palindromes whose radii are not determined. We show that if $l \geq 3$, then $c_{i+1} - c_i = c_i - c_{i-1}$ for $i = 2, \cdots, l-1$. By the invariant and Lemma 3.3, $\mathcal{S}[c_{i-1} - 2^{\eta+1}..c_i + 2^{\eta+1}]$ is periodic with period length $2(c_i - c_{i-1})$ and $\mathcal{S}[c_i - 2^{\eta+1}..c_{i+1} + 2^{\eta+1}]$ is periodic with period length $2(c_{i+1} - c_i)$. Therefore, by Lemma 3.2, $\mathcal{S}[c_i - 2^{\eta+1}..c_i + 2^{\eta+1}]$ is periodic with period length $2c$, where $c = \gcd(c_i - c_{i-1}, c_{i+1} - c_i)$. But by Lemma 3.4, if $c_i - c_{i-1} > c$, then the radius of the palindrome centered at position $c_i - c$ is larger than the radius of the palindrome centered at position c_{i-1} or the palindrome centered at position c_i, violating the invariant. Therefore, $c_i - c_{i-1} = c$ and similarly also $c_{i+1} - c_i = c$, establishing that the sequence of undetermined radii $\{c_i\}$ forms an arithmetic progression.

Note that an arithmetic progression can be represented by three integers: the start, the difference and the sequence length. If the undetermined radii in the two $2^{\eta-1}$-blocks that compose the current 2^η-block are represented this way, then the two representations are merged using constant number of operations. This permits an efficient access to all palindromes whose radii are undetermined.

We show next how to maintain the invariant at the end of each step. The computation takes constant time and $O(2^\eta)$ operations using symbol comparisons and the integer minima algorithm.

1. If the block contains a single undetermined radius, then the algorithm checks if the radius is at least $2^{\eta+2}$ or finds the radius exactly if it is shorter.

2. If the block contains a non-trivial arithmetic progression of undetermined radii $\{c_i\}$, $i = 1..l$, with difference c, then let $\mathcal{S}[\epsilon_L..\epsilon_R]$ be the maximal substring that contains $\mathcal{S}[c_1 - c..c_l + c - 1]$ and is periodic with period length $2c$. By Lemma 3.4, the radius of the palindrome centered at position c_i is exactly $\min(c_i - \epsilon_L, \epsilon_R - c_i + 1)$ except for at most one of the c_i's that satisfies $c_i - \epsilon_L = \epsilon_R - c_i + 1$.

   The algorithm checks if $\mathcal{S}[c_1 - 2^{\eta+2}..c_l + 2^{\eta+2} - 1]$ is periodic with period length $2c$. If this substring is not periodic, then the algorithm has found at least one of ϵ_L and ϵ_R and it can determine all radii which are smaller than $2^{\eta+2}$ by Lemma 3.4. If the algorithm found both ϵ_L and ϵ_R and there is a palindrome with undetermined radius centered at position $(\epsilon_L + \epsilon_R + 1)/2$, then the algorithm checks if the radius of this palindrome is at least $2^{\eta+2}$ or finds the radius exactly if it is shorter.

   Sometime the algorithm finds radii of longer palindromes but we prefer to leave these radii undetermined to maintain the invariant.

In the beginning of step number 0 there is a single undetermined radius in each block and the invariant is satisfied at the end of the step. At the end of step number $\lfloor \log n \rfloor - 2$ all radii have been determined. □

If the size of the alphabet is bounded by some constant, then the $O(\log n)$ time algorithm described above can be implemented using only $n/\log n$ processors, similarly to Galil's [12] string matching algorithm. This is achieved using the "four Russians trick" [2] of packing $\log n$ symbols into one number, in order to facilitate comparisons of up to $\log n$ symbols in a single operation. Note, that over general alphabet the algorithm described above is obsolete since the algorithm given in the next section is more efficient.

5 An $O(\log \log n)$ time algorithm

Theorem 5.1 *There exists an algorithm that computes the radii of all even palindromes in a string $\mathcal{S}[1..n]$ in $O(\log \log n)$ time using $n \log n / \log \log n$ processors and linear space.*

Proof: The algorithm proceeds in independent stages which are computed simultaneously. In stage number η, $0 \leq \eta \leq \lfloor \log n \rfloor - 3$, the algorithm computes all entries $\mathcal{R}[i]$ of the radii array such that $4l_\eta \leq \mathcal{R}[i] < 8l_\eta$, for $l_\eta = 2^\eta$.

Note that each stage computes disjoint ranges of the radii values and that all radii that are greater than or equal to 4 are computed by some stage. The radii between 0 and 3 are computed in a special stage that takes constant time and $O(n)$ operations. (The special stage assigns one processor to each entry of the radii array to check sequentially if the corresponding radius is between 0 and 3.)

We denote by T_η the time it takes to compute stage number η using O_η operations. We show that each stage η can be computed in $T_\eta = O(\log \log l_\eta)$ time and $O_\eta = O(n)$ operations. Since the stages are computed simultaneously, the time is $\max T_\eta = O(\log \log n)$. The total number of operation performed is $\Sigma_\eta O_\eta = O(n \log n)$. By Theorem 2.1, the algorithm can be implemented in $O(\log \log n)$ time using $n \log n / \log \log n$ processors. □

The remainder of this section describes a single stage η, $0 \leq \eta \leq \lfloor \log n \rfloor - 3$, that computes all values of the radii array $\mathcal{R}[1..n]$ that are between $4l_\eta$ and $8l_\eta - 1$, in $O(\log \log l_\eta)$ time and $O(n)$ operations.

Partition the input string $\mathcal{S}[1..n]$ into consecutive blocks of length l_η. Namely, block number k is $\mathcal{S}[(k-1)l_\eta + 1..kl_\eta]$. Stage number η consists of independent sub-stages that are also computed simultaneously. There is a sub-stage for each block. The sub-stage finds the radii of all palindromes which are centered in the block and whose radii are in the range computed by stage η. Sometimes palindromes whose radii are out of this range can be detected, but these radii do not have to be written into the output array since they are guaranteed to be found in another stage.

The sub-stage that is assigned to block number k starts with a call to the string matching algorithm to find all occurrences of the four consecutive blocks $\mathcal{S}[(k-4)l_\eta+1..kl_\eta]$, reversed, in $\mathcal{S}[(k-2)l_\eta+1..(k+4)l_\eta-1]$. Let p_i, $i=1,\cdots,r$, denote the indices of all these occurrences. The sequence $\{p_i\}$ has a "nice" structure as we show next.

Lemma 5.2 *Assume that the period length of a string $A[1..l]$ is p. If $A[1..l]$ occurs only at positions $p_1 < p_2 < \cdots < p_k$ of a string B and $p_k - p_1 \leq \lceil l/2 \rceil$, then the p_i's form an arithmetic progression with difference p.*

Proof: A simple consequence of Lemma 3.2. (See the paper by Apostolico, Breslauer and Galil [1].) □

Lemma 5.3 *The sequence $\{p_i\}$, which is defined above, forms an arithmetic progression.*

Proof: The sequence $\{p_i\}$ lists the indices of all occurrences of a string of length $4l_\eta$ in a string of length $6l_\eta - 1$. By Lemma 5.2, the p_i's form an arithmetic progression. □

By the last lemma, the sequence $\{p_i\}$ can be represented by three integers: the start, the difference and the sequence length. This representation is computed from the output of the string matching algorithm in constant time and $O(l_\eta)$ operations using the integer minima algorithm.

The next lemma states that we essentially found all "interesting" palindromes.

Lemma 5.4 *There exists a correspondence between the elements of the $\{p_i\}$ sequence and all palindromes that are centered in block number k and whose radii are large enough.*

- *If $p_i + kl_\eta$ is odd, then p_i corresponds to an even palindrome which is centered at position $(p_i + kl_\eta + 1)/2$.*
- *If $p_i + kl_\eta$ is even, then p_i corresponds to an odd palindrome which is centered on position $(p_i + kl_\eta)/2$.*

Each palindrome whose radius is at least $4l_\eta - 1$ has some corresponding p_i, while palindromes that correspond to some p_i are guaranteed to have radii that are at least $3l_\eta$.

Proof: Assume that there is an even palindrome whose radius is at least $4l_\eta - 1$ which is centered at position c, such that $(k-1)l_\eta < c \leq kl_\eta$. That is, $\mathcal{S}[c-i] = \mathcal{S}[c+i-1]$ for $i = 1,\cdots,4l_\eta - 1$. In particular, $\mathcal{S}[c-i] = \mathcal{S}[c+i-1]$ for $c - kl_\eta \leq i \leq c - (k-4)l_\eta - 1$, establishing that there is an occurrence of $\mathcal{S}[(k-4)l_\eta + 1..kl_\eta]$, reversed, starting at position $2c - kl_\eta - 1$.

Conversely, if there is an occurrence of $\mathcal{S}[(k-4)l_\eta + 1..kl_\eta]$, reversed, starting at position p_i, then $\mathcal{S}[kl_\eta - j] = \mathcal{S}[p_i + j]$ for $j = 0, \cdots, 4l_\eta - 1$. In particular, if $p_i + kl_\eta$ is odd, then $\mathcal{S}[kl_\eta - j] = \mathcal{S}[p_i + j]$ for $j = (kl_\eta - p_i + 1)/2, \cdots, 4l_\eta - 1$, establishing that there is an even palindrome of radius $4l_\eta - (kl_\eta - p_i + 1)/2 \geq 3l_\eta$ centered at position $(p_i + kl_\eta + 1)/2$.

Similar arguments hold for odd palindromes. □

We could design the algorithm to find the odd palindromes directly, but we rather use the reduction to even palindromes that was given in Section 3. Define the sequence $\{q_i\}$ for $i = 1, \cdots, l$, to list all centers of the even palindromes that correspond to elements in $\{p_i\}$. By the last lemma, if the difference of the arithmetic progression $\{p_i\}$ is even or if there is only a single element, then all the p_i's correspond either to odd or to even palindromes. If the difference of the arithmetic progression $\{p_i\}$ is odd, then every second element corresponds to an even palindrome. Thus, the sequence $\{q_i\}$ also forms an arithmetic progression and therefore it can be computed efficiently.

If the $\{q_i\}$ sequence does not have any elements, then there are no even palindromes whose radius is at least $4l_\eta$ that are centered in the current block. If there is only one element q_1, then we can find in constant time and $O(l_\eta)$ operations what is the radius of the palindrome that is centered at q_1 or we can conclude that it is too large to be computed in this stage.

If there are more elements, let q denote the difference of the arithmetic progression $\{q_i\}$. The next lemma shows how to find the radii of the palindromes centered at $\{q_i\}$ efficiently.

Lemma 5.5 *It is possible to find the radii of all palindromes centered at positions in $\{q_i\}$ in constant time and $O(l_\eta)$ operations.*

Proof: For each ζ, such that $q_1 - 8l_\eta \leq \zeta < q_1$, verify if $\mathcal{S}[\zeta] = \mathcal{S}[\zeta + 2q]$. Let ζ_L be the smallest such index ζ that satisfies $\mathcal{S}[\zeta_L..q_1 - 1] = \mathcal{S}[\zeta_L + 2q..q_1 + 2q - 1]$. Similarly, for each ζ, such that $q_l \leq \zeta < q_l + 8l_\eta$, verify if $\mathcal{S}[\zeta] = \mathcal{S}[\zeta - 2q]$. Let ζ_R be the largest such index ζ that satisfies $\mathcal{S}[q_l - 2q..\zeta_R - 2q] = \mathcal{S}[q_l..\zeta_R]$. The indices ζ_L and ζ_R are computed in constant time and $O(l_\eta)$ operations using symbol comparisons and the integer minima algorithm.

By Lemma 5.4, the palindromes centered at the positions q_i have radii that are larger than q. Therefore, by Lemma 3.3, $\zeta_L < q_1 - q$ and $\zeta_R \geq q_l + q$ and by Lemma 3.4, the radius of the palindrome centered at position q_i is at least $\rho_i = \min(q_i - \zeta_L, \zeta_R - q_i + 1)$.

If $\rho_i \geq 8l_\eta$, then the radius of the palindrome centered at q_i is too large to be computed in this stage and it does not have to be determined exactly. Otherwise, the radius is exactly ρ_i except for at most one of the q_i's which satisfies $q_i - \zeta_L = \zeta_R - q_i + 1$. For this particular q_i, we can find in constant time and $O(l_\eta)$ operations what is the radius of the palindrome or we can conclude that it is too large to be computed in this stage. □

Lemma 5.6 *Stage number η correctly computes all entries of the output array $\mathcal{R}[1..n]$ that are in the range $4l_\eta, \cdots, 8l_\eta - 1$. It takes $O(\log\log l_\eta)$ time and make a total of $O(n)$ operations.*

Proof: There are $\lfloor n/l_\eta \rfloor$ sub-stages in stage η, each uses $O(l_\eta)$ operations. Thus the number of operations used is $O(n)$. Breslauer and Galil's [4] parallel string matching algorithm takes $O(\log\log l_\eta)$ time to find the $\{p_i\}$ sequence and the rest of the work is done in constant time. □

6 Conclusion

The question whether there exists an optimal $O(\log\log n)$ time parallel algorithm that finds all palindromes in strings over general alphabets remains open. It is possible that one could prove an $\Omega(n \log n)$ lower bound on the operation count of any $O(\log\log n)$ time algorithm and even for $O(\log n)$ time algorithms.

The recognition problem of *Palstar*, the language of strings that are obtained as concatenation of non-trivial palindromes, is an interesting related problem. In the sequential setting Knuth, Morris and Pratt [15] described a linear time algorithm that recognizes strings that are composed of even palindromes. Galil and Seiferas [13] solved the general problem by giving a linear-time on-line algorithm that recognizes *Palstar*.

In parallel, Crochemore and Rytter [7] designed an $O(\log n)$ algorithm for the case of even palindromes and composition of k palindromes, for $k = 2, 3, 4$. Their algorithm uses the radii of all palindromes, which are computed more efficiently by the new algorithms that were described in this paper. However, the other steps of their algorithms seem to require $O(\log n)$ time, and the question of fast parallel recognition of *Palstar* is still open.

7 Acknowledgments

We thank Roberto Grossi and Laura Toniolo for several discussions about palindromes and for comments on this paper.

References

[1] A. Apostolico, D. Breslauer, and Z. Galil. Optimal Parallel Algorithms for Periods, Palindromes and Squares. In *Proc. 19th International Colloquium on Automata, Languages, and Programming*, number 623 in Lecture Notes in Computer Science, pages 296–307. Springer-Verlag, Berlin, Germany, 1992.

[2] V.L. Arlazarov, E.A. Dinic, M.A. Kronrod, and I.A. Faradzev. On economic construction of the transitive closure of a directed graph. *Soviet Math. Dokl.*, 11:1209–1210, 1970.

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

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

[5] D. Breslauer and Z. Galil. Finding all Periods and Initial Palindromes of a String in Parallel. *Algorithmica*, 1994. To appear.

[6] S.A. Cook. Linear time simulation of deterministic two-way pushdown automata. In *Information Processing 71*, pages 75–80. North Holland Publishing Co., Amsterdam, the Netherlands, 1972.

[7] 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.

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

[9] M.J. Fischer and M.S. Paterson. String matching and other products. In R.M. Karp, editor, *Complexity of Computation*, pages 113–125. American Mathematical Society, Prividence, RI., 1974.

[10] Z. Galil. Two fast simulations which imply some fast string matching and palindrome-recognition algorithms. *Inform. Process. Lett.*, 4(4):85–87, 1976.

[11] Z. Galil. Palindrome Recognition in Real Time by a Multitape Turing Machine. *J. Comput. System Sci.*, 16(2):140–157, 1978.

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

[13] Z. Galil and J. Seiferas. A Linear-Time On-Line Recognition Algorithm for "Palstar". *J. Assoc. Comput. Mach.*, 25(1):102–111, 1978.

[14] Z. Kedem, G.M. Landau, and K. Palem. Optimal Parallel Suffix-Prefix Matching Algorithm and Applications. In *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, pages 388–398, 1989.

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

[16] 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.

[17] 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.

[18] A.O. Slisenko. Recognition of palindromes by multihead Turing machines. In V.P. Orverkov and N.A. Sonin, editors, *Problems in the Constructive Trend in Mathematics VI (Proceedings of the Steklov Institute of Mathematics, No. 129)*, pages 30–202. Academy of Sciences of the USSR, 1973. English Translation by R.H. Silverman, pp. 25–208, Amer. Math. Soc., Providence, RI, 1976.