Fast Parallel String Prefix-Matching

Dany Breslauer
Columbia University

CUCS-041-92

Abstract

An $O(\log\log m)$ time $\frac{n\log m}{\log\log m}$-processor CRCW-PRAM algorithm for the string prefix-matching problem over a general alphabet is presented. The algorithm can also be used to compute the KMP failure function in $O(\log\log m)$ time on $\frac{m\log m}{\log\log m}$ processors. These results improve on the running time of the best previous algorithm for both problems, which was $O(\log m)$, while preserving the same number of operations.

1 Introduction

String matching is the problem of finding all occurrences of a short pattern string $\mathcal{P}[1..m]$ in a longer text string $\mathcal{T}[1..n]$. The classical sequential algorithm of Knuth, Morris and Pratt [12] solves the string matching problem in time that is linear in the length of the input strings. The Knuth-Morris-Pratt [12] string matching algorithm can be easily generalized to find the longest pattern prefix that starts at each text position within the same time bound. We refer to this problem as *string prefix-matching*.

In parallel, the string matching problem can be solved in $O(\log\log m)$ time on a $\frac{n}{\log\log m}$-processor CRCW-PRAM as shown by Breslauer and Galil [7]. However, the best parallel algorithms for the string prefix-matching problem and for computing the KMP failure function were simple derivations of Galil's [11] $O(\log m)$ time n-processor string matching algorithm. (The KMP failure function is a table that is computed in the pattern processing step of the Knuth-Morris-Pratt string matching algorithm and is used to guide that algorithm when comparisons fail.) These bounds are over a general alphabet where the only access an algorithm has to the input strings is by pairwise symbol comparisons. In fact, Galil's [11] algorithm can be implemented using only $\frac{n}{\log m}$ processors if the size of the alphabet is a constant.

This paper presents a new algorithm for the string prefix-matching problem over a general alphabet. The algorithm takes $O(\log\log m)$ time on a $\frac{n\log m}{\log\log m}$-processor CRCW-PRAM. It is also shown that this algorithm can be used to compute the KMP failure function of a string $\mathcal{P}[1..m]$ in $O(\log\log m)$ time on $\frac{m\log m}{\log\log m}$ processors.

A parallel algorithm is said to achieve *an optimal speedup* if its time-processor product is the same as the running time of the fastest sequential algorithm. The new algorithms that are presented in this paper are still a factor of $\log m$ processors away from optimality, but they have the same time-processor product as the best previous parallel algorithms [11] for the two problems. Both algorithms are the fastest possible with the number of processors used as implied by a lower bound that was given by Breslauer and Galil [8] for the string matching problem. Note that both problems can be solved even in a constant time if more processors are available.

The string prefix matching algorithm follows techniques that were used in solving several other parallel string problems [1, 2, 5, 6, 9]. In particular, it uses the parallel string matching algorithm of Breslauer and Galil [7] as a procedure that solves several string matching problems simultaneously and then combines the results of the string matching problems into an answer to the string prefix-matching problem.

The paper is organized as follows. Section 2 overviews some parallel algorithms and tools that are used in the new algorithms. Section 3 describes the prefix-matching algorithm and Section 4 shows how to use that algorithm to compute the KMP failure function.

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 they always write the same value.

The prefix matching algorithm uses a string matching algorithm as a procedure to find all occurrences of a given pattern in a given text. The input to the string matching algorithm consists of two strings, $pattern[1..m]$ and $text[1..n]$, and the output is a Boolean array $match[1..n]$ that has a "*true*" value at each position where an occurrence of the pattern starts in the text. We use Breslauer and Galil's [7] parallel string matching algorithm that takes $O(\log\log m)$ time on a $\frac{n}{\log\log m}$-processor CRCW-PRAM. This algorithm is the fastest optimal parallel string matching algorithm possible over a general alphabet as shown by Breslauer and Galil [8].

We also use an algorithm of Fich, Ragde and Wigderson [10] to compute the minima of n integers from the range $1 \cdots n$ in a constant time using an n-processor CRCW-PRAM. We use this algorithm, for example, to find the first occurrence of a string in an other string; After all the occurrences are computed by the string matching algorithm mentioned above, the minima algorithm is used to find the smallest i such that $match[i] =$ "*true*".

For the computation of the KMP failure function we use an algorithm that computes the prefix maxima of a sequence. Berkmen, Schieber and Vishkin [3] noticed that the parallel maxima algorithm of Shiloach and Vishkin [14] can be modified to find the maxima of each prefix of an n element sequence in $O(\log\log n)$ time on a $\frac{n}{\log\log n}$-processor CRCW-PRAM.

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 [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. Coming back to the example above, that finds the first occurrence of one string in another, one sees that the second step of finding the smallest index of an occurrence takes a constant time on n processors, while the call to the string matching procedure takes $O(\log\log m)$ time on $\frac{n}{\log\log m}$ processors. By Theorem 2.1 the second step can be slowed down to work in $O(\log\log m)$ time on $\frac{n}{\log\log m}$ processors.

As mentioned in the introduction, the string prefix matching problem can be solved faster if more processors are available.

Theorem 2.2 *The string prefix-matching problem takes a constant time on a nm-processor CRCW-PRAM.*

Proof: The following trivial string prefix matching algorithm takes a constant time.

- Assign m processors to each text position to find the length of the longest pattern prefix that starts at that position. Each of the m processors simultaneously compares the symbols of the pattern with the corresponding symbols of the text.

- Find the position of the first comparison that failed in each group of m comparisons that were assigned to specific text position. The successful comparisons up to the first comparison that failed correspond to the longest pattern prefix that occurs starting at this text position.

  This step takes a constant time on m processors using the Fich, Ragde and Wigderson [10] integer minima algorithm.

Since there are m processors assigned to each of the n text positions the total number of processors used is nm. $\Box$

3 The Prefix-Matching Algorithm

We describe an algorithm that given the text string $\mathcal{T}[1..n]$ and the pattern string $\mathcal{P}[1..m]$ will compute the longest pattern prefix that occurs starting at each text position. The output will be an array $\Phi[1..n]$ such that $\mathcal{T}[i..i+\Phi[i]-1] = \mathcal{P}[1..\Phi[i]]$ and if $\Phi[i] < m$, then $\mathcal{T}[i+\Phi[i]] \neq \mathcal{P}[\Phi[i]+1]$. Using this notation, if $\Phi[i] = m$, then a complete occurrence of the pattern starts at text position i.

Theorem 3.1 *There exist an algorithm that given the input strings $\mathcal{T}[1..n]$ and $\mathcal{P}[1..m]$, will compute the longest pattern prefix that starts at each text position in $O(\log\log m)$ time on $\frac{n\log m}{\log\log m}$ processors.*

Proof: To simplify the presentation assume without loss of generality that the algorithm can access indices of the input strings which are out of the string boundaries and that all comparisons to these symbols fail. All entries of the output array $\Phi[1..n]$ are initialized to be zero.

The algorithm will proceed in independent stages which are computed simultaneously. In stage number η, $0 \le \eta \le \lfloor \log m \rfloor$, the algorithm computes all entries $\Phi[i]$ of the output array such that $2^\eta \le \Phi[i] < 2^{\eta+1}$. Note that the each stage computes disjoint ranges of the output array values and that all possible values are covered.

We denote by T_η the time it takes to compute stage number η on P_η processors. The number of operations in stage η is $O_\eta = T_\eta P_\eta$. In the next section it is shown that each stage η can be computed in $T_\eta = O(\log\log 2^\eta)$ time and $O_\eta = O(n)$ operations using Breslauer and Galil's [7] parallel string matching algorithm.

Since the stages of the algorithm are computed simultaneously, the total number of operations performed in all stages is $\sum_\eta O_\eta = O(n\log m)$ and the time is $\max T_\eta = O(\log\log m)$. By Theorem 2.1 the algorithm can be implemented in $O(\log\log m)$ time on $\frac{n\log m}{\log\log m}$ processors. $\Box$

3.1 A Single Stage

This section describes a single stage η, $0 \le \eta \le \lfloor \log m \rfloor$, that computes all values of the output array Φ that are in the range $2^\eta \cdots 2^{\eta+1}-1$, in $O(\log\log 2^\eta)$ time and n operations.

Stage number η starts with a call to a string matching algorithm to find all occurrences of the pattern prefix $\mathcal{P}[1..2^\eta]$ in the text. Note that a pattern prefix which is long enough to be in the range that has to be computed by this stage can only start at these occurrences. In the rest of this section we show how to find efficiently the maximal length of the pattern prefixes that start at each of these occurrences or to verify that the prefixes are long enough to be computed by another stage.

If an occurrence is found starting at some text position q, then the algorithm knows that a pattern prefix whose length is at least 2^η starts at that text position. Similarly to Theorem 2.2, using only 2^η processors, the algorithm can find in a constant time the length of the pattern prefix that starts at text position q or it can conclude that the prefix is at least of length $2^{\eta+1}$ and therefore out of the range that has to be computed by this stage.

This last step is very efficient. However, since there can be many occurrences of $\mathcal{P}[1..2^\eta]$ in the text, repeating this step for all these occurrences can be too costly. We restrict our attention to a small part of the text string and solve the problem simultaneously in each part. This allows us to use some periodicity properties of strings which are described below.

We partition the text string $\mathcal{T}[1..n]$ into consecutive blocks of length $\lfloor 2^{\eta-1} \rfloor + 1$ each. For the rest of this section we restrict our attention to a single block. Let q_i, $i = 1..r$, be the indices of all occurrences of the pattern prefix $\mathcal{P}[1..2^\eta]$ that start at text positions in one such block.

Definition 3.2 *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 $i = 1..m-\pi$.*

Lemma 3.3 *(Lyndon and Schutzenberger [13]) 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 $\gcd(p,q)$.*

Lemma 3.4 *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 $p_k - p_1 \leq \lceil \frac{l}{2} \rceil$, 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 of lengths p and $q = p_{i+1} - p_i$. Since $p \leq q \leq \lceil \frac{l}{2} \rceil$, by Lemma 3.3 it also has a period of length $gcd(p,q)$. But p is the length of 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 of length p. If $q > p$, then there must be another occurrence of A at position $p_i + p$ of B; a contradiction. $\Box$

Lemma 3.5 *The sequence $\{q_i\}$, which is defined above, forms an arithmetic progression with difference π, where π is the period length of $\mathcal{P}[1..2^\eta]$.*

Proof: The sequence $\{q_i\}$ lists the indices of all occurrences of $\mathcal{P}[1..2^\eta]$ that start in a text block of length $\lfloor 2^{\eta-1} \rfloor + 1$. By Lemma 3.4 the q_i's form an arithmetic progression with difference π, the period length of $\mathcal{P}[1..2^\eta]$. $\Box$

The sequence $\{q_i\}$ can be represented using three integers: the start, the difference, and the length of the sequence. This representation can be easily computed from the output of the string matching problem using Fich, Ragde and Wigderson's [10] minima algorithm in a constant time and 2^η processors.

Let ψ be the position where the period $\mathcal{P}[1..\pi]$ of $\mathcal{P}[1..2^\eta]$ terminates in the pattern prefix $\mathcal{P}[1..2^{\eta+1}]$ and $2^{\eta+1}+1$ if it does not terminate in this prefix. Let θ be the position where the period of $\mathcal{P}[1..2^\eta]$ terminates in the text substring $\mathcal{T}[q_r..q_r+2^{\eta+1}-1]$ and $q_r+2^{\eta+1}$ if it does not terminate in this substring. By terminated periodicity we mean that $\mathcal{P}[1..\psi-\pi-1] = \mathcal{P}[\pi+1..\psi-1]$ and $\mathcal{P}[\psi-\pi] \neq \mathcal{P}[\psi]$ and that $\mathcal{T}[q_i..\theta-\pi-1] = \mathcal{T}[q_i+\pi..\theta-1]$ and $\mathcal{T}[\theta-\pi] \neq \mathcal{P}[\theta]$. The indices ψ and θ can be computed in a constant time on 2^η processors.

If the sequence $\{q_i\}$ has only a single element q_1, the algorithm can find the length of the pattern prefix that starts at text position q_1 using the approach which is described before. Otherwise, if the sequence $\{q_i\}$ has more than one element, the algorithm finds the length of the pattern prefixes that start at text positions in $\{q_i\}$ as described next in Lemma 3.6. The algorithm might still be required to use the approach that was described before to find the length of the pattern prefix that starts at one of the $\{q_i\}$ text positions.

Lemma 3.6 *Let $\lambda = \min(\theta - q_i, \psi - 1)$. Then, the longest pattern prefix that starts at text position q_i is at least of length λ. Furthermore,*

1. *If $\theta - q_i \neq \psi - 1$, then the length of that prefix is exactly λ.*
2. *If $\theta - q_i = \psi - 1$, then that prefix can continue to any length and it is necessary compare more symbols to compute its length.*

   Note that at most one of the q_i's can fall under this category.

Proof: Both the pattern prefix $\mathcal{P}[1..\psi - 1]$ and the text substring $\mathcal{T}[q_i..\theta - 1]$ have period $\mathcal{P}[1..\pi]$, the period of the pattern prefix $\mathcal{P}[1..2^\eta]$. Therefore, it is clear that $\mathcal{P}[1..\lambda] = \mathcal{T}[q_i..q_i + \lambda - 1]$.

1. If $\theta - q_i \neq \psi - 1$, then either,

$$\mathcal{P}[\lambda + 1] = \mathcal{P}[\lambda - \pi + 1] \text{ and } T[q_i + \lambda] \neq \mathcal{T}[q_i + \lambda - \pi]$$

   or,

$$\mathcal{P}[\lambda + 1] \neq \mathcal{P}[\lambda - \pi + 1] \text{ and } T[q_i + \lambda] = \mathcal{T}[q_i + \lambda - \pi].$$

   Since $\lambda \geq 2^\eta \geq \pi$ and $\mathcal{P}[\lambda - \pi + 1] = \mathcal{T}[q_i + \lambda - \pi]$, in both cases $\mathcal{P}[\lambda + 1] \neq \mathcal{T}[q_i + \lambda]$ proving that the length of the pattern prefix that starts at text position q_i is exactly λ.

2. If $\theta - q_i = \psi - 1$, then it suffices to compare $P[1..2^{\eta+1}]$ to $\mathcal{T}[q_i..q_i + 2^{\eta+1} - 1]$ to find the length of the pattern prefix that starts at text position q_i or to conclude that the prefix is at least of length $2^{\eta+1}$ and therefore out of the range that has to be computed by this stage.

   The extra comparisons are necessary since, if $\lambda < 2^{\eta+1}$, then $\mathcal{P}[\lambda + 1] \neq \mathcal{P}[\lambda - \pi + 1]$ and $T[q_i + \lambda] \neq \mathcal{T}[q_i + \lambda - \pi]$ and it is possible that $\mathcal{P}[\lambda + 1] = \mathcal{T}[q_i + \lambda]$.

□

The computation in stage η proceeds in each text block of length $\lfloor 2^{\eta-1} \rfloor + 1$ simultaneously and can be summarized as follows:

1. Find all occurrences of the pattern prefix $\mathcal{P}[1..2^\eta]$ in the considered text block and compute the $\{q_i\}$ sequence.
2. Compute the period length π of the pattern prefix $\mathcal{P}[1..2^\eta]$.
3. Compute θ and ψ.

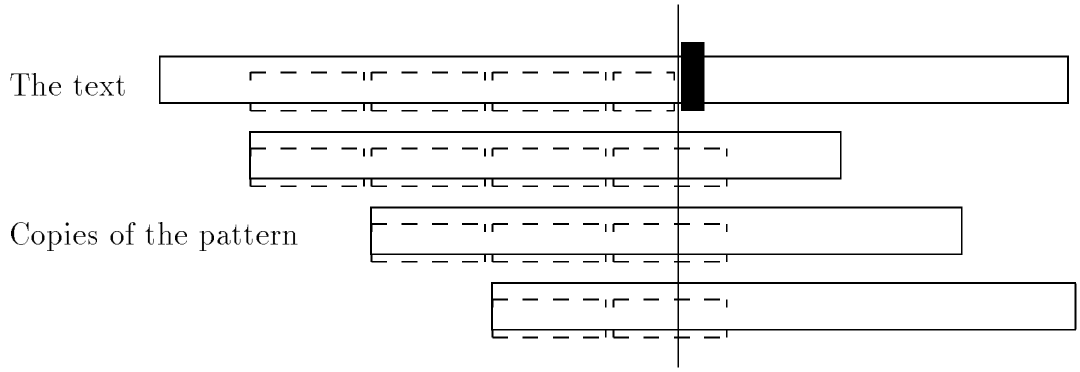

Figure 1: If $\theta - q_i < \psi - 1$, then all pattern prefixes that start at text positions $\{q_i\}$ terminate at the same text position.

4. Find the length of the pattern prefix that starts at each text position q_i. By Lemma 3.6 the length is given by θ and ψ except for at most one of the q_i's that has to be found separately.

   If the length of the pattern prefix that starts at text position q_i is out of the range that has to be computed by this stage do not update the output array entry $\Phi[q_i]$ since it will be updated in another stage.

Lemma 3.7 *Stage number η correctly computes all entries of the output array $\Phi[1..n]$ that are in the range $2^\eta \cdots 2^{\eta+1} - 1$. It takes $O(\log\log 2^\eta)$ time and a total of n operations.*

Proof: The calls to Breslauer and Galil's [7] string matching algorithm take $O(\log\log 2^\eta)$ time and n operations.

The sequence q_i can be represented by three integers which can be computed from the output of the string matching algorithm (that is assumed to be a Boolean vector representing all occurrences) in a constant time and 2^η operations in each block. The rest of the work in each block also takes a constant time and 2^η operations.

There are $\frac{n}{\lfloor 2^{\eta-1} \rfloor + 1}$ blocks and thus, stage η takes $O(\log\log 2^\eta)$ time and $O(n)$ operations.
$\square$

4 The KMP Failure Function

The Knuth-Morris-Pratt [12] string matching algorithm computes in its pattern preprocessing step a table that is used later to guide the text processing step when comparisons fail. This table is often called the *KMP failure function.*

Knuth, Morris and Pratt [12] actually define two function: $\mathcal{F}[1..m]$ and $next[1..m]$. Both function can be used to guide the comparisons that fail, but the $next[]$ function has more information and therefore it is more efficient. In this section we show that using the string prefix-matching algorithm one can compute both functions efficiently.

Both the $\mathcal{F}[]$ and the $next[]$ functions are strongly related to the periods of the pattern prefixes and are actually a simple shift of the $\Pi[]$ and $\hat{\Pi}[]$ functions of the pattern that are defined next:

- Given a string $\mathcal{S}[1..m]$, the function $\Pi[1..m]$ is defined for $\mathcal{S}[1..m]$ such that $\Pi[i]$ is the period length of the prefix $\mathcal{S}[1..i]$.
- Given a string $\mathcal{S}[1..m]$, the table $\hat{\Pi}[1..m]$ is defined for $\mathcal{S}[1..m]$ such that $\hat{\Pi}[i]$ is the length of the shortest terminated period at position i of $\mathcal{S}[1..m]$ if such a period exists.

  That is, $\hat{\Pi}[i]$ is the length of the shortest period of $\mathcal{S}[1..i-1]$ that is not a period of $\mathcal{S}[1..i]$. If all periods of $\mathcal{S}[1..i-1]$ are also periods of $\mathcal{S}[1..i]$ then $\hat{\Pi}[i]$ is undefined.

Theorem 4.1 *The function $\Pi[1..m]$ can be computed in $O(\log\log m)$ time on a $\frac{m\log m}{\log\log m}$-processor CRCW-PRAM.*

Proof: The algorithm will start by solving a string prefix-matching problem with the input string $\mathcal{S}[1..m]$ given as both pattern and text. The output of the string prefix-matching problem contains essentially all the information needed for the $\Pi[1..m]$ function. Note that an integer k is a period length of all prefixes $\mathcal{S}[1..i]$ such that $k \le i \le k+\Phi[k+1]$. Therefore,

$$\Pi[i] = \min\{k \mid 1 \le k \le i \le k + \Phi[k+1]\}.$$

We show that $\Pi[1..m]$ can be computed on a CRCW-PRAM in $O(\log\log m)$ time on $\frac{m}{\log\log m}$ processors if $\Phi[1..m]$ is given. The computation follows three steps:

1. Compute a function $\mathcal{K}[1..m]$ such that,

   $$\mathcal{K}[i] = \max_{k\le i}\{k + \Phi[k+1]\}.$$

   Using this notation an integer i is the period length of all prefixes $\mathcal{S}[1..k]$ such that $\mathcal{K}[i-1] < k$ and $k \le \mathcal{K}[i]$.

2. Compute a function $\mathcal{B}[1..m]$ such that $\mathcal{B}[\mathcal{K}[i-1]+1] = i$ if $\mathcal{K}[i] > \mathcal{K}[i-1]$ and $\mathcal{B}[k] = 0$ otherwise.

3. Compute the $\Pi[1..m]$ function.

   $$\Pi[i] = \max_{k\le i}\{\mathcal{B}[k]\}.$$

Note that both maxima computations can be done by Berkman, Schieber and Vishkin's [3] prefix maxima algorithm. □

For the computation of the $\hat{\Pi}[1..m]$ function we use a more powerful CRCW-PRAM model which is called the *priority CRCW-PRAM.* In this model each processor has a pre-assigned priority and simultaneous writes of different values to a memory cell are allowed. The actual value written is that of the processor with the highest priority.

Theorem 4.2 *The function $\hat{\Pi}[1..m]$ can be computed in $O(\log\log m)$ time on a $\frac{m \log m}{\log\log m}$-processor priority CRCW-PRAM.*

Proof: The algorithm will start by solving a string prefix-matching problem with the input string $\mathcal{S}[1..m]$ given as both pattern and text. The output of the string prefix-matching problem contains essentially all the information needed for the $\hat{\Pi}[1..m]$ function. Note that a period of length k terminates at position $k + \Phi[k+1] + 1$ of the input string $\mathcal{S}[1..m]$. Thus,

$$\hat{\Pi}[i] = \min\{k \mid i = k + \Phi[k+1] + 1\}.$$

The $\hat{\Pi}[1..m]$ array can be computed in a constant time on a priority CRCW-PRAM once that $\Phi[1..m]$ is given:

1. Initialize all entries of the $\hat{\Pi}[1..m]$ array to be undefined.
2. For each integer k, $1 \le k \le m$ assign a processor with priority k that attempts to write the value k into $\hat{\Pi}[k + \Phi[k+1] + 1]$.

If the write conflict are resolved in such a way that the processor with the smallest priority value succeeds in writing at each memory location, then the computation of $\hat{\Pi}[1..m]$ is complete. $\Box$

5 Acknowledgements

I would like to thanks Alberto Apostolico and Laura Toniolo for helpful discussions.

References

[1] A. Apostolico and D. Breslauer. An Optimal $O(\log\log n)$ Time Parallel Algorithm for Detecting all Squares in a String. manuscript, 1992.

[2] A. Apostolico, D. Breslauer, and Z. Galil. Optimal Parallel Algorithms for Periods, Palindromes and Squares. In *Proc. 19th International Colluquium on Automata, Languages, and Programming*. Springer-Verlag, Berlin, Germany, 1992. 296–307.

[3] O. Berkman, B. Schieber, and U. Vishkin. Some doubly logarithmic optimal parallel algorithms based on finding nearest smallers. manuscript, 1988.

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

[5] D. Breslauer. A Parallel String Superprimitivity Test. manuscript, 1992.

[6] D. Breslauer. *Efficient String Algorithmics*. PhD thesis, Dept. of Computer Science, Columbia University, New York, NY, 1992.

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

[8] D. Breslauer and Z. Galil. A Lower Bound for Parallel String Matching. *SIAM J. Comput.*, 21(5):856–862, 1992.

[9] D. Breslauer and Z. Galil. Finding all periods and initial palindromes of a string in parallel. manuscript, 1992.

[10] 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.

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

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

[13] 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.

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