# A lower bound for parallel string matching

*Dany Breslauer*
INRIA and Columbia University

April 26, 1993

[summary by Mireille Regnier]

**Abstract**

This talk presents the derivation of an $\Omega(\log\log m)$ lower bound on the number of rounds necessary for finding occurrences of a pattern string $P[1..m]$ in a text string $T[1..2m]$ in parallel using $m$ comparisons in each round. The parallel complexity of the string matching problem using $p$ processors for general alphabets follows.

## 1. Introduction

Better and better parallel algorithms have been designed for string-matching. All are on CRCW-PRAM with the weakest form of simultaneous write conflict resolution: all processors which write into the same memory location must write the same value of 1. The best CREW-PRAM algorithms are those obtained from the CRCW algorithms for a logarithmic loss of efficiency. Optimal algorithms have been designed: $O(\log m)$ time in [8, 17] and $O(\log\log m)$ time in [4]. (An optimal algorithm is one with $pt = O(n)$ where $t$ is the time and $p$ is the number of processors used.) Recently, Vishkin [18] developed an optimal $O(\log^* m)$ time algorithm. Unlike in the case of the other algorithms this time bound does not account for the preprocessing of the pattern: the preprocessing takes $O(\frac{\log^2 m}{\log\log m})$. Vishkin's super fast algorithm raised the question whether an optimal constant-time algorithm is possible.

We show that a CRCW-PRAM with $m$ processors requires $\Omega(\log\log m)$ time to perform string matching. Thus, our $O(\log\log m)$ optimal parallel algorithm cannot be improved and Vishkin's algorithm crucially depends on a slower preprocessing. Our result is the first lower bound for parallel algorithms that solve the string matching problem. More precisely, the exact parallel complexity of string matching for general alphabets on the CRCW-PRAM is: $\Theta(\lceil\frac{m}{p}\rceil + \log\log_{\lceil 1+p/m\rceil} 2p)$.

Our model is similar to Valiant's parallel comparison tree model [16]. We assume the only access the algorithm has to the input strings is by comparisons which 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 comparisons which can result in a *less than*, *equal* or *greater than* answers.

As the execution can be partitioned into comparison rounds followed by computation rounds, our lower bound immediately translates into a lower bound for the time of the CRCW-PRAM.

If the pattern is given in advance and any preprocessing is free, then this lower bound does not hold, as Vishkin's $O(\log^* m)$ algorithm shows. The lower bound also does not hold for CRCW-PRAM over a fixed alphabet strings. Similarly, finding the maximum in the parallel decision tree model has the same lower bound [16], but for small integers the maximum can be found in constant time on a CRCW-PRAM [7].

## 2. A lower bound for finding the period of a string and string matching

Given a string $S[1..m]$, we say that $k$ is a *period length* of $S$ if $S[i+k] = S[i]$ for $i = 1, \ldots, m-k$. We call $k$ *the period length* of $S$ if it is the minimal period length of $S$. In this section we prove a lower bound for

the problem of finding the period length of a string $S[1..m]$ using $m$ comparisons in each round. Our lower bound also holds for determining whether such a string has a period of length smaller than $\frac{m}{2}$.

We show a strategy for an adversary to answer $\frac{1}{4}\log\log m$ 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 a period of length smaller than $\frac{m}{2}$ and in the other it does not have any such period. This implies that any algorithm which terminates in less rounds can be fooled. Also, let $S$ be a string of length $2m$ generated by this adversary. Assume we present to some string matching algorithm pattern $S[1..m]$ and text $S[2..2m]$. The choice left open above determines the occurrence or not of $P$ in $T$. Thus, the lower bound holds also for any string matching algorithm.

At the beginning of round $i$ the adversary will maintain an integer $k_i$ which is a possible period length. I.e. we can fix $S$ consistently with answers to previous comparisons in such a way that $k$ is a period length 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 $k_i$ *forces* this comparison.

The adversary answers the comparisons of round $i$ in such a way that some $k_{i+1}$ is a possible period length and few symbols of $S$ are fixed. Hence, few comparisons are forced. It maintains the following invariants which hold at the beginning of round number $i$:

(1) $k_i$ satisfies $\frac{1}{2}K_i \le k_i \le K_i$ and the number of fixed symbols $f_i$ satisfies $f_i \le K_i$.
(2) If $S[l]$ was fixed then for every $j \equiv l \mod k_i$ $S[j]$ 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 was answered as unequal, then
    (a) it was between different residues modulo $k_i$;
    (b) if the symbols were fixed then they were fixed to different values.

Note that invariants (3) and (4) imply consistency of the answers given so far. Joined with invariant (2), they imply that $k_i$ is a possible period length: we fix all symbols in each unfixed residue class modulo $k_i$ to a new symbol, choosing them different for different residue classes.

We start at round number 1 with $k_1 = K_1 = 1$: invariants hold initially. Now, all multiples of $k_i$ in the range $\frac{1}{2}K_{i+1} \ldots K_{i+1}$ are candidates for the new $k_{i+1}$. The proof relies on the existence of a "good candidate":

LEMMA 1. *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 $\frac{4mK_i \log m}{K_{i+1}}$ comparisons.*

PROOF. The number of prime multiples of $k_i$ that satisfy $\frac{1}{2}K_{i+1} \le k_{i+1} \le K_{i+1}$ is greater than $\frac{K_{i+1}}{4K_i \log m}$. From [15], the number of primes between $\frac{1}{2}n$ and $n$ is greater than $\frac{1}{4}\frac{n}{\log n}$. Also, let $p, q \ge \sqrt{\frac{m}{k_i}}$ be relatively prime, and $l, q$ be two different integers: $1 \le k < l \le m$. The double condition $l \equiv k \mod pk_i$, $l \equiv k \mod qk_i$ implies $l \equiv k \mod pqk_i$, hence $l = k$, a contradiction. Hence, a comparison $S[l] = S[k]$ is forced by at most one of $pk_i$ and $qk_i$. As the total number of comparisons forced by all these candidates is at most $m$, there is a candidate that forces at most $\frac{4mK_i \log m}{K_{i+1}}$ comparisons. □

We are now ready to prove that the adversary can answer the comparisons in round $i$ so that the invariants also hold at the beginning of round $i+1$. Since our "good candidate" $k_{i+1}$ is a multiple of $k_i$, the residue classes modulo $k_i$ split; each class splits into $\frac{k_{i+1}}{k_i}$ residue classes modulo $k_{i+1}$. Note that if two indices are in different residue classes modulo $k_i$, then they are also in different residue classes modulo $k_{i+1}$; if two indices are in the same residue class modulo $k_{i+1}$, then they are also in the same residue class modulo $k_i$. For each comparison forced by $k_{i+1}$ involving two positions $(l, j)$ in the same residue class modulo $k_{i+1}$, the adversary fixes the residue class modulo $k_{i+1}$ to the same new symbol (a different symbol for different residue classes). The adversary answers comparisons between fixed symbols based on the value they are fixed to. All other comparisons involve at least one unfixed symbol: They are always answered as unequal.

We show that the invariants still hold.

(1) This is clear by simple induction computation.

(2) Residue classes previously fixed satisfied (2). This is maintained by the splitting process into several residue classes. Any symbol fixed at this round causes its entire residue class modulo $k_{i+1}$ to be fixed to the same symbol.

(3) Equal answers of previous rounds are not affected. Equal answers of this round are either between symbols which were fixed before to the same value or are within the same residue class modulo $k_{i+1}$ and the entire residue class is fixed to the same value.

(4) (a) Unequal answers of previous rounds are between different residue classes modulo $k_{i+1}$ since residue classes modulo $k_i$ split. Unequal answers of this round are between different residue classes because comparisons within the same 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 residue class fixed.

THEOREM 1. *Any comparison-based parallel algorithm for finding the period length of a string $S[1..m]$ using $m$ comparisons in each round requires $\frac{1}{4}\log\log m$ rounds.*

PROOF. The choice is still open after each round. □

THEOREM 2. *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. □

From the remark at the beginning of this section:

THEOREM 3. *The lower bound holds also for any comparison-based string matching algorithm.*

## 3. More comparisons in each round

One can use the trivial algorithm to solve the string matching problem in constant time if $m^2$ comparisons are available in each round on a CRCW-PRAM. Therefore, no more than $m^2$ processors are necessary. If the number of processors $p$ is smaller than $\frac{m}{\log\log m}$ then one can slow down the $\log\log m$ algorithm in [4] to run in $O(\frac{m}{p})$ time. Additionally:

THEOREM 4. *Any comparison-based parallel algorithm for finding the period length of a string $S[1..m]$ using $p$ comparisons, $m \leq p \leq m^2$, in each round requires at least $\Omega(\log\log_{\frac{2p}{m}} p)$ rounds.*

PROOF. We change $m$ 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 $K_i \leq \frac{m}{2}$, i.e. $i = \Theta(\Omega(\log\log_{\frac{2p}{m}} p))$. □

## Bibliography

[1] Apostolico (A.), Iliopoulos (C.), Landau (G. M.), Schieber (B.), and Vishkin (U.). – Parallel construction of a suffix tree with applications. *Algorithmica*, vol. 3, 1988, pp. 347–365.

[2] Borodin (A. B.), Fischer (M. J.), Kirkpatrick (D. G.), Lynch (N. A.), and Tompa (M.). – A time-space tradeoff for sorting on non-oblivious machines. In *Proceedings 20th IEEE Symposium on Foundations of Computer Science*, pp. 294–301. – 1979.

[3] Boyer (R. S.) and Moore (J. S.). – A fast string searching algorithm. *Communications of the ACM*, vol. 20, 1977, pp. 762–772.

[4] Breslauer (D.) and Galil (Z.). – An optimal $O(\log\log n)$ parallel string matching algorithm. *SIAM Journal on Computing*, vol. 19, n° 6, 1990, pp. 1051–1058.

[5] Crochemore (M.). – String-matching and periods. *Bulletin of the EATCS*, October 1989.

[6] Crochemore (M.) and Perrin (D.). – *Two Way Pattern Matching.* – Technical report, LITP, 1989.

[7] Fich (F. E.), Ragde (R. L.), and Wigderson (A.). – Relations between concurrent-write models of parallel computation. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pp. 179–189. – 1984.

[8] Galil (Z.). – Optimal parallel algorithms for string matching. *Information and Control*, vol. 67, 1985, pp. 144–157.

[9] Galil (Z.) and Seiferas (J.). – Saving space in fast string-matching. *SIAM Journal on Computing*, no2, 1980, pp. 417–438.

[10] Galil (Z.) and Seiferas (J.). – Time-space-optimal string matching. *Journal of Computer System Science*, vol. 26, 1983, pp. 280–294.

[11] Geréb-Graus (M.) and Li (M.). – Three one-way heads cannot do string matching. – Manuscript.

[12] Knuth (D. E.), Morris (J.), and Pratt (V.). – Fast pattern matching in strings. *SIAM Journal on Computing*, vol. 6, 1977, pp. 323–350.

[13] Li (M.). – *Lower bounds on string-matching.* – Technical Report n° TR 84-636, Department of Computer Science, Cornell University, 1984.

[14] Li (M.) and Yesha (Y.). – String-matching cannot be done by a two-head one-way deterministic finite automaton. *Information Processing Letters*, vol. 22, 1986, pp. 231–235.

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

[16] Valiant (L. G.). – Parallelism in comparison models. *SIAM Journal on Computing*, vol. 4, 1975, pp. 348–355.

[17] Vishkin (U.). – Optimal parallel pattern matching in strings. *Information and Control*, vol. 67, 1985, pp. 91–113.

[18] Vishkin (U.). – Deterministic sampling: A new technique for fast pattern matching. In *STOCS'90*, volume 22, pp. 170–179. – 1990. Baltimore, MD.