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

[L] Li, M. (1984), Lower bounds on string-matching, TR 84-636 Department of Computer Science, Cornell University.

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

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

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

[V85] Vishkin, U. (1985), Optimal parallel pattern matching in strings, *Information and Control 67*, 91-113.

[V90] Vishkin, U. (1990), Deterministic sampling - A new technique for fast pattern matching, *Proc. 22nd ACM Symp. on Theory of Computation*, 170-179.

References

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

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

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

[BM] Boyer, R. S., and Moore, J. S. (1977), A fast string searching algorithm, *Comm. ACM 20*, 762-772.

[C] Crochemore, M., String-Matching and Periods, In *Bulletin of EATCS*, October 1989.

[CP] Crochemore, M. and Perrin, D. (1989), Two way pattern matching, *JACM*, to appear.

[FRW] 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.

[G] Galil, Z. (1985), Optimal parallel algorithms for string matching, *Information and Control 67*, 144-157.

[GS80] Galil, Z., Seiferas, J. (1980), Saving space in fast string-matching, *SIAM J. on Comput. 2*, 417-438.

[GS83] Galil, Z., Seiferas, J. (1983), Time-space-optimal string matching, *J. Comput. Syst. Sci. 26*, 280-294.

[GL] Geréb-Graus, M. and Li, M. (1990), Three one-way heads cannot do string matching, manuscript.

the time complexity of the string matching problem becomes $\Theta\left(\log\log_{\frac{2p}{m}} p\right)$.

Lemma 4.1: If $c > 1$ and $p^{1-c^{-k}} = \frac{m}{2}$, then $k = \Theta\left(\log\log_{\frac{2p}{m}} p\right)$.

Proof: We have $\frac{2p}{m} = p^{c^{-k}}$. Taking logarithms twice we get

$$k = \log_c \frac{\log p}{\log \frac{2p}{m}} = \log_c \log_{\frac{2p}{m}} p. \ \square$$

Theorem 4.2: The algorithm in [BG] takes $O(\log\log_{\frac{2p}{m}} p)$ time if we use $m \leq p \leq m^2$ processors.

Proof: The algorithm in [BG] consists of two phases. The first phase is the pattern analysis phase which is easily modified to check in round i for period lengths in the range $p^{1-2^{-(i-1)}} \ldots p^{1-2^{-i}}$. This phase terminates when $p^{1-2^{-i}} \geq \frac{m}{2}$ and by Lemma 4.1 $O(\log\log_{\frac{2p}{m}} p)$ rounds are sufficient. The second phase is the text analysis phase which can also be modified to work within the same time bound. $\square$

Theorem 4.3: 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 $K_i \leq \frac{m}{2}$. By Lemma 4.1, $\Omega(\log\log_{\frac{2p}{m}} p)$ rounds are necessary. $\square$

5 Acknowledgments

We would like to thank Maxime Crochmemore, Raffaele Giancarlo, Pino Italiano, Kunsoo Park, Dominique Perrin and Neil Sarnak for many helpful discussions and comments on early versions of this paper.

If a symbol $S[l]$ was fixed at round $j \leq i$ its entire residue class is assigned the the name $< l \bmod k_2, l \bmod k_3, ..., l \bmod k_{j+1} >$ 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 residue class 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 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 residue classes modulo k_{i+1} are fixed in such a way that k_{i+1} is the 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 $< l \bmod k_2, l \bmod k_3, ..., l \bmod k_{i+1}, l >$. □

3 A lower bound for string matching

Theorem 3.1: The lower bound holds also for any comparison-based string matching algorithm.

Proof: Fix a string matching algorithm. We present to the algorithm a pattern $P[1..m]$ which is $S[1..m]$ and a text $T[1..2m-1]$ which is $S[2..2m]$, where S is a string of length $2m$ generated by the adversary in the way described above. (We use the same adversary that we used in the previous proof; the adversary sees all comparisons as comparisons between symbols in S.) After $\frac{1}{4} \log \log 2m$ rounds the adversary still has the choice of fixing S to have a period length smaller than m, in which case we will have an occurrence of P in T, or to fix all unfixed symbols to completely different characters, what implies that there would be no such occurrence. Thus, the lower bound holds also for any such string matching algorithm. □

4 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 [BG] to run in $O(\frac{m}{p})$ time. We prove below that for $m \leq p \leq m^2$

5. We prove inductively that $f_{i+1} \leq K_{i+1}$. We fix at most $\frac{4mK_i \log m}{K_{i+1}}$ residue classes modulo k_{i+1}. There are k_{i+1} such classes and each class has at most $\lceil \frac{m}{k_{i+1}} \rceil \leq \frac{2m}{k_{i+1}}$ elements. By Lemma 2.5 and simple algebra the number of fixed elements satisfies:

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

Theorem 2.7: 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: Fix an algorithm which finds the period of S and let the adversary described above answer the comparisons. After $i = \frac{1}{4} \log \log m$ rounds $f_{i+1}, k_{i+1} \leq m^{1-4^{-\frac{1}{4} \log \log m}} = \frac{m}{\sqrt{\log m}} \leq \frac{m}{2}$. The adversary can still fix S to have a period length k_{i+1} by fixing each remaining residue class modulo k_{i+1} to the same symbol, different symbol for each class. 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 period length smaller than $\frac{m}{2}$. Consequently, any algorithm which terminates in less than $\frac{1}{4} \log \log m$ rounds can be fooled. $\square$

Theorem 2.8: 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 $< l \bmod k_2, l \bmod k_3, ..., l \bmod k_{i+1} >$.

Proof: By Lemma 2.4 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 $l \equiv j \mod k_{i+1}$ 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 two positions in different residue classes modulo k_{i+1} (and at least one unfixed symbol) and are always answered as unequal.

Since 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.

We show that the invariants still hold.

1. The candidate we chose for k_{i+1} was in the required range.

2. Residue classes which were fixed before split into several residue classes, all are fixed. 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 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 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.

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 2.1. It is at least

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

Each one of these candidates also satisfies the condition of Lemma 2.2 since $k_i \leq K_i$, $pk_i \geq \frac{K_{i+1}}{2}$ and

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

Lemma 2.4: 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 $\frac{4mK_i \log m}{K_{i+1}}$ comparisons.

Proof: By Lemma 2.3 there are at least $\frac{K_{i+1}}{4K_i \log m}$ such candidates which are prime multiples of k_i and satisfy the condition of Lemma 2.2. By Lemma 2.2 each of the m comparisons is forced by at most one of them. So the total number of comparisons forced by all these candidates is at most m. Thus, there is a candidate that forces at most $\frac{4mK_i \log m}{K_{i+1}}$ comparisons. $\square$

Lemma 2.5: For m large enough and $i \leq \frac{1}{4}\log\log m$, $1 + m^{2\cdot 4^{-i}} 16 \log m \leq m^{3\cdot 4^{-i}}$.

Proof: For m large enough,

$$\log\log(1 + 16\log m) < \frac{1}{2}\log\log m = (1 - \frac{2}{4})\log\log m$$

$$\log(1 + 16\log m) < 4^{-\frac{1}{4}\log\log m}\log m$$

$$1 + 16\log m < m^{4^{-\frac{1}{4}\log\log m}} \leq m^{4^{-i}},$$

from which the lemma follows. $\square$

Lemma 2.6: 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 $\frac{4mK_i \log m}{K_{i+1}}$ comparisons. Then the adversary can answer the comparisons in round i so that the invariants also hold at the beginning of round $i+1$.

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.

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 period length: if we fix all symbols in each unfixed residue class modulo k_i to a new symbol, a different symbol for different residue classes, we obtain a string consistent with the answers given so far that has a period length 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 $l \equiv j \mod k_{i+1}$. We say that k_{i+1} *forces* this comparison.

Theorem 2.1 (see [RS]): 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 2.2: If $p, q \geq \sqrt{\frac{m}{k_i}}$ and are relatively prime, then a comparison $S[l] = S[k]$ is forced by at most one of pk_i and qk_i.

Proof: Assume $l \equiv k \mod pk_i$, $l \equiv k \mod qk_i$ for $1 \leq l, k \leq m$. Then also $l \equiv k \mod pqk_i$. But $pqk_i \geq m$ and $1 \leq l, k \leq m$ which implies $l = k$, a contradiction. □

Lemma 2.3: 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 $\frac{K_{i+1}}{4K_i \log m}$. Each such candidate satisfies the condition of Lemma 2.2.

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 [Va], but for small integers the maximum can be found in constant time on a CRCW-PRAM [FRW].

In Section 2 we present a lower bound for a related problem of finding the period length of a string. In Section 3 we show how this lower bound extends to string matching. In Section 4 we extend our bound to cases where the number of comparisons is greater than m.

2 A lower bound for finding the period of a string

Given a string $S[1..m]$, we say that k is a *period length* of S if $S[i+k] = S[i]$ for $i = 1,..,m-k$. 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.

We say that an integer k is a possible period length if 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 $S[l] = S[j]$.

At the beginning of round i the adversary will maintain an integer k_i which is a possible period length. 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. Let $K_i = m^{1-4^{-(i-1)}}$. The adversary will maintain the following invariants which hold at the beginning of round number i:

1. k_i satisfies $\frac{1}{2}K_i \leq k_i \leq K_i$.

As a result we derive the exact parallel complexity of string matching for general alphabets on the CRCW-PRAM. In the case $1 \leq p \leq \frac{m}{\log\log m}$, it is known to be $\Theta(\frac{m}{p})$; when $\frac{m}{\log\log m} \leq p \leq m$ it is $\Theta(\log\log m)$; when $m \leq p \leq m^2$ it is $\Theta(\log\log_{\frac{2p}{m}} p)$; and when $p \geq m^2$ it is known to be $\Theta(1)$. All these bounds can be summerized in one expression: $\Theta(\lceil \frac{m}{p} \rceil + \log\log_{\lceil 1+p/m \rceil} 2p)$.

Various string matching algorithms use different models. The weakest model is the comparison model in which we test for equality of two symbols. Comparisons of this type are sufficient to solve the string matching problem in the sequential case [BM, KMP, GS83] and in the parallel case [BG, V85]. However, although the definition of the string matching problem does not require the alphabet to be ordered, an arbitrary order is used in some algorithms [C, CP]. This assumption is reasonable since the alphabet symbols are encoded numerically, which introduces a natural order. Some other algorithms use a more restricted model where the alphabet symbols are assumed to be encoded as integers in a small range [AILSV, G]. These algorithms usually take advantage of the fact that the alphabet symbols can be used as indices of an array or that many symbols can be packed together in one register. The case where only comparisons are used is usually referred to in the literature as the general alphabet assumption, while the latter case is usually called fixed alphabet.

Our model is similar to Valiant's parallel comparison tree model [Va]. 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.

Consider a CRCW-PRAM that solves the string matching problem over a general alphabet. In this case the PRAM can only perform comparisons but no computation with its input symbols. Thus, its execution can be partitioned into comparison rounds followed by computation rounds. Therefore, a lower bound for the number of rounds in the parallel comparison model 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

1 Introduction

The string matching problem is defined as follows: given a string $P[1..m]$ called pattern and a string $T[1..n]$ called text find all occurrences of the pattern in the text. In the parallel case the output of a string matching algorithm will be a Boolean array $MATCH[1..n]$ which contains a *true* value at each position where an occurrence of the pattern starts. We consider the case where $n = 2m$. The string matching problem has so far defied all attempts of proving lower bounds.

In the sequential case, at a certain time when the problem had a logarithmic space algorithm [GS80] and linear time algorithm [KMP], it was conjectured to have a time-space tradeoff [BFKLT]. But this conjecture was later disproved when a linear-time constant space algorithm was discovered [GS83]. Moreover, even a 6-head two-way finite automaton can perform string matching in linear time. It is still an open problem whether a k-head one-way finite automaton can perform string matching. The only known cases are for $k = 1, 2, 3$ [GL, LY, L] where the answer is negative.

In the parallel case, better and better algorithms have been designed, all 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. Galil [G] designed an optimal $O(\log m)$ time algorithm for the case of a constant size alphabet. (An optimal algorithms is one with $pt = O(n)$ where t is the time and p is the number of processors used.) Vishkin [V85] gave an optimal $O(\log m)$ time algorithm for general alphabet. Breslauer and Galil [BG] obtained an optimal $O(\log \log m)$ time algorithm for general alphabet. Recently, Vishkin [V90] 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 in Vishkin's algorithm 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.

In this paper we prove the first lower bound for parallel algorithms that solve the string matching problem. 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.

A Lower Bound for Parallel String Matching*

Dany Breslauer†
Columbia University

Zvi Galil‡
Columbia University and
Tel-Aviv University

CUCS-003-92

Abstract

We present 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 bound is within a constant factor of the fastest algorithm for this problem [BG] and also holds for an m-processor CRCW-PRAM in the case of a general alphabet. Consequently we derive the parallel complexity of the string matching problem using p processors for general alphabets which is,

- $\Theta(\frac{m}{p})$ if $p \leq \frac{m}{\log\log m}$
- $\Theta(\log\log m)$ if $\frac{m}{\log\log m} \leq p \leq m$
- $\Theta(\log\log_{\frac{2p}{m}} p)$ if $m \leq p \leq m^2$
- $\Theta(1)$ if $p \geq m^2$

or in short $\Theta(\lceil\frac{m}{p}\rceil + \log\log_{\lceil 1+p/m\rceil} 2p)$.

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

†Partially supported by an IBM Graduate Fellowship. Work done while visiting the IBM T.J Watson Research Center.

‡Work done while visiting LITP, Paris.