SIAM J. COMPUT.
Vol. 1, No. 1, pp. 000–000, October 1993

000

CONSTANT-TIME RANDOMIZED PARALLEL STRING MATCHING*

MAXIME CROCHEMORE†, ZVI GALIL‡, LESZEK GASIENIEC§, KUNSOO PARK¶, and WOJCIECH RYTTER♮

Abstract. Given a pattern string of length m for the string matching problem, we design an algorithm that computes deterministic samples of a sufficiently long substring of the pattern in constant time. This problem used to be the bottleneck in the pattern preprocessing for one- and two-dimensional pattern matching. The best previous time bound was $O(\log^2 m/\log\log m)$. We use this algorithm to obtain the following results. All algorithms below are optimal parallel algorithms on a CRCW PRAM.

1. A deterministic string-matching algorithm which takes $O(\log\log m)$ time for preprocessing and constant time for text search, which are the best possible in both preprocessing and text search.
2. A constant-time deterministic string-matching algorithm in the case that the text length n satisfies $n = \Omega(m^{1+\epsilon})$ for a constant $\epsilon > 0$.
3. A simple string-matching algorithm that has constant time with high probability for random input.
4. The main result: A constant expected time Las-Vegas algorithm for computing the period of the pattern and all witnesses and thus for string matching itself. In both cases an $\Omega(\log\log m)$ lower bound is known for deterministic algorithms.

Key words. parallel string matching, randomized algorithms, deterministic samples

AMS subject classifications. 68Q22, 68Q25, 68R15

1. Introduction. The string matching problem is defined as follows: Given pattern $P[0..m-1]$ and text $T[0..n-1]$, find all occurrences of P in T. We study parallel complexity of string matching on a CRCW PRAM. The PRAM (Parallel Random Access Machine) is a shared memory model of parallel computation which consists of a collection of identical processors and a shared memory. Each processor is a RAM, working synchronously and communicating via the shared memory. The CRCW (Concurrent Read Concurrent Write) PRAM allows both concurrent reads and concurrent writes to a memory location, and it has several variants depending on how c oncurrent writes are handled. We use the weakest version (called Common in [7]) in which the only concurrent writes allowed are of the same value 1. A parallel algorithm for a problem is *optimal* if its total work is asymptotically the same as the minimum possible work for the problem. All optimal algorithms in this paper

*Received by the editors May 1, 1992; accepted for publication (in revised form) December 1, 1992.

†Institut Gaspard Monge, University of Marne-la-Vallée (mac@univ-mlv.fr). Partially supported by PRC "Algorithmique, Modèles, Infographie" and GREG "Motifs dans les séquences".

‡Department of Computer Science, Columbia University and Tel-Aviv University (galil@cs.columbia.edu). Partially supported by NSF grant CCR-90-14605 and CISE Institutional Infrastructure Grant CDA-90-24735.

§Instytut Informatyki, Uniwersytet Warszawski, 02-097 Warszawa, Poland (lechu@mimuw.edu.pl). Partially supported by KBN grant 2-11-90-91-01 and EC Cooperative Action IC-1000 (project ALTEC).

¶Department of Computer Engineering, Seoul National University, Seoul 151-742, Korea (kpark@theory.snu.ac.kr). Supported by KOSEF grant 951-0906-069-2.

♮Instytut Informatyki, Uniwersytet Warszawski, 02-097 Warszawa, Poland (rytter@mimuw.edu.pl). Work was done while visiting UC Riverside. Supported by the project ALTEC.

$$
\begin{array}{r|cccccccc|l}
\cline{2-9}
 & 0&1&0&1&0&0&0&1 & 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1 \\
0 & 1&0&1&0&0&0&1&0 & 1\ 1\ 1\ 0\ 1\ 0\ 1 \\
x \rightarrow\ 0\ 1 & \underline{0}&\underline{1}&0&\underline{0}&0&1&0&1 & 1\ 1\ 0\ 1\ 0\ 1 \\
0\ 1\ 0 & 1&0&0&0&1&0&1&1 & 1\ 0\ 1\ 0\ 1 \\
0\ 1\ 0\ 1 & 0&0&0&1&0&1&1&1 & 0\ 1\ 0\ 1 \\
0\ 1\ 0\ 1\ 0 & 0&0&1&0&1&1&1&0 & 1\ 0\ 1 \\
0\ 1\ 0\ 1\ 0\ 0 & 0&1&0&1&1&1&0&1 & 0\ 1 \\
0\ 1\ 0\ 1\ 0\ 0\ 0 & 1&0&1&1&1&0&1&0 & 1 \\
\cline{2-9}
\end{array}
$$

Fig. 1. *A 3-size DS of x for 8 shifts: $A = \{2, 3, 5\}$ and $f = 6$.*

have linear work. Most string matching algorithms consist of two stages. The first preprocesses the pattern and the second uses the data structure constructed in the first to search the text. For the text search, a constant-time optimal parallel algorithm (following optimal $O(\log^2 m/\log\log m)$-time preprocessing) is known [8]. On the other hand, an $\Omega(\log\log m)$ lower bound with a linear number of processors is known for the entire string matching problem [3].

Let $x[0..r-1]$ be a string of length r. String $x[0..p-1]$, $1 \le p < r$, is a *period* of x if $x[i] = x[i+p]$ for all $0 \le i < r-p$. The shortest period of x is called *the period* of x. We also use the term period for the *length* of the corresponding string. If the period of x is shorter than a half (fourth) of x, x is called *periodic* (*4-periodic*). A *witness* of x against (the periodicity of) i is a position w such that $x[w] \ne x[w-i]$ [12]. Let p be the period of x. When we say that we compute the period of x, we mean computing $\min(p, r/4)$. When we say that we compute the witnesses of x, we mean computing the witnesses against all non-periods i, $1 \le i < r/4$. The witnesses of x can be computed in optimal $O(\log\log r)$ time by [2]. (It is possible to compute large periods and witnesses using techniques of [1]. But we will not need them here.) Given two strings x and y, and a position i of y such that x does not occur at position i of y, a *witness to* non-occurrence at i is a position w such that $y[w] \ne x[w-i]$. A substring of x of length i is called an *i-block* of x. The positions of all strings in this paper start from 0.

Consider a nonperiodic pattern string x of length m. Align $k \le m/2$ copies of x one on top of the other so that the ith copy starts above the ith symbol of the first copy. A deterministic sample (DS) of x for k shifts is an ordered set A of positions and a number f, $1 \le f \le k$, such that $f-1$ consecutive copies of x to the left and $k-f$ consecutive copies to the right have at least one mismatch with copy number f of x in the positions of the set A. The size of a DS is the size of the ordered set A. See Fig. 1. Vishkin [13] introduced the notion of deterministic samples and proved the existence of a DS of size at most $\log m$ for $m/2$ shifts.

DS is crucial for very fast optimal parallel search of the pattern in a given text. During the text search we maintain a subset of text positions, referred to as *candidates*, which can be start positions of pattern occurrences. Assume that we can somehow reduce the number of candidates to one in every $\log m$-block of the text. Then for every candidate we compare the symbols at the positions of the set A of DS with the corresponding symbols of the text. If a candidate has mismatches, it is no longer a candidate for an occurrence of the pattern. On the other hand, if a candidate has matches in all the positions of A, then by the definition of DS we can eliminate all other candidates in an $m/2$-block of the text. This method was used in a constant-time optimal text search [8]. Very recently, it was also used in a constant-time two-dimensional text search [5]. However, the optimal algorithm suggested by Vishkin and used in [8] for computing DS was very expensive, taking $O(\log^2 m/\log\log m)$ time. This resulted in two "best" algorithms for string matching: an optimal $O(\log\log m)$-time algorithm for the entire problem [2] (for which also an $\Omega(\log\log m)$ lower bound was proved [3]) and a constant-time text search with expensive preprocessing that was dominated by the computation time of DS. Thus the DS computation has been the bottleneck.

In §2 we present a constant-time deterministic algorithm for computing a DS of logarithmic size. We also show how to compute a constant-size DS for $O(\log\log m)$ shifts in constant time. This constant-size DS will be crucial to our main result in §4. Since we compute deterministic samples for a part of the pattern, we can use more than linear number of processors.

§3 contains three applications of the constant-time algorithm for DS. The first application is that it allows us to have only one best string-matching algorithm with constant-time search and $O(\log\log m)$-time preprocessing. Our new algorithm achieves the best possible time in both preprocessing and text search. The second application is a deterministic $O(k)$-time string-matching algorithm using n processors for the case that $m = O(n^{1-2^{-k}})$, i.e., a constant-time string-matching algorithm using n processors when $n = \Omega(m^{1+\epsilon})$ for a constant $\epsilon > 0$. The third application is a simple string-matching algorithm that has constant time with high probability (and thus constant expected time) for random input.

In §4 we describe our main result. We present a constant expected time Las-Vegas algorithm for computing the witnesses. Together with the constant-time text search, we obtain a constant expected time Las-Vegas algorithm for string matching including preprocessing, solving the main open problem remaining in parallel string matching. Deterministically an $\Omega(\log\log m)$ lower bound is known for witness computation and string matching [3]. This algorithm is designed based on the lower bound argument; randomization is used to kill the argument. In the special case that the pattern is periodic and the period of the pattern has only a constant number of prime divisors, randomization is not needed.

Our algorithms will frequently use without mention the constant-time algorithm that finds the maximum (or minimum) position of a nonzero entry in an array [7]. Our algorithms will use the constant-time deterministic polynomial approximate compaction (PAC) of Ragde [11] and its improvement by Hagerup [9]. A d-PAC is an algorithm that compacts an array of size n with at most m nonzero elements into a prefix of size m^d (assuming that $m^d < n$). Ragde gave a $(4+\epsilon)$-PAC and Hagerup a $(1+\epsilon)$-PAC for any $\epsilon > 0$.

In many places where we use quantities such as $r/2$, $\log r$ or $\log\log r$ as integers we mean that any way of rounding them to the nearest integer will do.

2. Constant-Time Deterministic Sampling. Let x be a nonperiodic string of length r. We construct two kinds of DS's in constant time: a $\log k$-size DS for k shifts, $k \le r/2$, and a constant-size DS for $\log\log r$ shifts. We first show how to construct a $\log k$-size DS of x for k shifts, $k \le r/2$, in constant time using r^3 processors and r^2 space. This log-size DS was introduced by Vishkin [13], but its construction takes $O(\log^2 r/\log\log r)$ time using $O(r)$ operations, which is the bottleneck in the preprocessing of string matching [13,8]. Consider k-blocks starting at positions i for $0 \le i < k$. If two k-blocks are identical, we say that x *has a periodicity*. Note that x has a periodicity if and only if there are i and j for $i < j < k$ (the start positions of the two identical blocks) such that $x[i..j+k-1]$ has a period $p = j - i < k \le r/2$.

$$
\begin{array}{r|l|l}
\cline{2-2}
 & 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1 & 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\
1 & 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 & 1\ 0\ 1\ 0\ 0\ 1\ 1 \\
1\ 1 & 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 & 0\ 1\ 0\ 0\ 1\ 1 \\
1\ 1\ 0 & 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 & 1\ 0\ 0\ 1\ 1 \\
1\ 1\ 0\ 1 & 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 & 0\ 0\ 1\ 1 \\
1\ 1\ 0\ 1\ 0 & 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0 & 0\ 1\ 1 \\
1\ 1\ 0\ 1\ 0\ 1 & 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0 & 1\ 1 \\
x \rightarrow\ 1\ 1\ 0\ 1\ 0\ 1\ 0 & 1\ 0\ 1\ 0\ \underline{1}\ 0\ \underline{0}\ 1 & 1 \\
\cline{2-2}
\multicolumn{1}{r}{} & \multicolumn{1}{l}{\qquad\quad\ \ \uparrow\quad\ \uparrow} & \\
\multicolumn{1}{r}{} & \multicolumn{1}{l}{\qquad\quad\ \ q\quad q+p} &
\end{array}
$$

Fig. 2. *A DS with* $A = \{q, q+p\}$ *and* $f = 1$.

Using $k^3 < r^3$ processors, the algorithm checks in constant time if x has a periodicity. If it does, the algorithm finds such i, j and $p = j - i$. p is not necessarily the smallest such period.

Case 1. x has a periodicity in $x[i..j+k-1]$ with period p: Although p is a period of $x[i..j+k-1]$, p cannot be a period of x since x is nonperiodic. That is, the periodicity with period p cannot extend both all the way to the right and all the way to the left of $x[i..j+k-1]$. If it does not extend to the right, let $q > j$ be the smallest position such that $x[q] \neq x[q+p]$ (i.e., end of periodicity). The position q can be found in constant time with r processors. Now the DS is $A = \{q, q+p\}$ and $f = 1$ because in the first copy we have mismatching symbols at positions $q, q+p$ and in the next $k-1$ copies to the right we have matching symbols at the same positions. See Fig. 2, where $k = 8$, $i = 2$, and $j = 4$ ($p = j - i = 2$). If the periodicity extends all the way to the right, let $q < i$ be the largest position such that $x[q] \neq x[p+q]$. The DS is $A = \{q, q+p\}$ and $f = k$.

Case 2. x does not have a periodicity (i.e., all the k-blocks are distinct): Consider (for discussion only) the compacted prefix tree T of all the blocks (each path from the root of T to a leaf corresponds to a block and every internal node has degree at least two). Since T has k leaves, there is at least one leaf v of depth $\leq \log k$. Let B be the block corresponding to v. The path in T from the root to v hits at most $\log k$ nodes which define at most $\log k$ positions; B is different from each of the other blocks in at least one of these positions. Below we will find such a block B and the (at most $\log k$) positions in B in constant time using r^3 processors and r^2 space. Let b be the start position of B in x. The DS is the derived set of positions in B shifted by b and $f = k - b$. For example, consider Fig. 1. For the block $B = 01000101$, its start position in x is 2 and f is 6.

To find B and the positions in it, we compute a $k \times k$ 0-1 matrix: one row for each block. The matrix is set initially to 0. With r processors per each pair (i, j), $1 \leq i, j \leq k$, of blocks, find in constant time the smallest index ℓ such that the ith block and the jth block differ at position ℓ (a node in T). Note that we find in this way exactly all nodes of T (more than once). Set to 1 entry ℓ in the two rows i and j. Now we only have to find a row with no more than $s = \log k$ 1's and compress their positions to an array of size s. So we need to solve the following problem for each row of the matrix. Given a 0-1 array of size $k \leq r$ and r^2 processors, find whether it has at most $s = \log k$ 1's and if it does compress their positions into an array of size s.

Ragde designed a $(4+\epsilon)$-PAC [11] and then used it to compress an array of length k with at most s items (= nonzero entries) into an array of size s in time $O(\log s / \log\log k)$. In case the number of items is larger than s the algorithm fails.

Note that when $\log s = O(\log\log k)$ the time is $O(1)$. So to solve the problem above, first the j-th processor replaces a 1 in the j-th entry with j and then Ragde's compressions is applied in constant time. This compression will succeed with at least one of the rows of the matrix and will yield the desired DS.

THEOREM 1. *The deterministic algorithm above constructs a* $\log k$*-size DS for* k *shifts,* $k \le r/2$*, for a nonperiodic string of length* r *in constant time using* r^3 *processors and* r^2 *space.*

Although the DS computation used to be the bottleneck, the algorithm in [8] has another part of the preprocessing (the hitting set) that does not take constant time: the part that enables the algorithm to eliminate all but at most one candidate in every $\log m$-block. The hitting set can be constructed in $O(\log\log m)$ time. Using Theorem 1 and an $O(\log\log m)$-time construction of the hitting set, the algorithm in [8] can be transformed into a string-matching algorithm which takes constant time for text search and $O(\log\log m)$ time for preprocessing. However, in order to derive a randomized constant-time algorithm we cannot afford to compute a hitting set. Instead of the hitting set, we use the following constant-size DS for $O(\log\log m)$ shifts to design an alternative algorithm called CONST-MATCH in §3.

We now show how to construct a constant-size DS of x for $\log\log r$ shifts in constant time with $r^2\log\log r$ processors. This constant-size DS is new and is crucial for constant-time randomized string matching in §4 as discussed above.

Case 1. If there exists position i in x such that $x[i] \ne x[i+j]$ (or $x[i] \ne x[i-j]$) for every $1 \le j < \log r$, then we take $A = \{i\}$ and $f = \log r$ (or $A = \{i\}$ and $f = 1$) as the DS for $\log r$ shifts (and therefore for $\log\log r$ shifts as well).

Case 2. Otherwise, every symbol in x occurs very often (with distance shorter than $\log r$ between neighboring ones). So every symbol occurs at least $r/\log r$ times in x, which implies that there are at most $\log r$ different symbols in x. Consider all substrings of length $\log\log r$ in the first half of x. Since there are $(\log r)^{\log\log r}$ different strings of length $\log\log r$ over $\log r$ symbols, and $(\log r)^{\log\log r} < r/(2\log\log r)$, some substrings of length $\log\log r$ repeat without overlap in the first half of x. Find such a substring y in constant time using $r^2\log\log r$ processors. Let z be the substring between the two copies of y. The substring yzy has a period $p = |yz| < r/2$. Since x is nonperiodic, period p has a mismatch in x. Let q be the smallest (largest) position such that $x[q] \ne x[q+p]$ to the right (left) of the first copy of y. Then $A = \{q, q+p\}$ and $f = 1$ ($A = \{q, q+p\}$ and $f = \log\log r$) is a constant-size DS for $\log\log r$ shifts.

THEOREM 2. *The deterministic algorithm above constructs a constant-size DS for* $\log\log r$ *shifts for a nonperiodic string of length* r *in constant time using* $r^2\log\log r$ *processors.*

3. Applications of Constant-Time DS. In this section we present applications of the constant-time DS computation, and at the same time we build up procedures which will be used in the constant-time randomized string-matching algorithm in §4. Thanks to a well known reduction [2], we can assume without loss of generality that the pattern in a string matching problem is non-4-periodic.

The text search algorithm will maintain candidates, which can still be start positions of pattern occurrences. All other positions have gotten witnesses to non-occurrences. Consider two candidates $i < j$ such that w is a witness against $j - i$ (i.e., $P[w] \ne P[w-j+i]$).

(a) If $T[i+w] \ne P[w]$, $i+w$ is a witness to non-occurrence at i.

(b) If $T[i+w] \ne P[w-j+i]$, $i+w$ is a witness to non-occurrence at j.

The two tests above are called a *duel* between i and j [12]. By a duel we can remove one or both of the candidates. Given $h > 0$, we partition the text T into disjoint h-blocks in the obvious way. If every h-block has at most two candidates, we say that T is *h-good.*

LEMMA 1. *If T is h-good and an h-size DS for k shifts is given, then T can be made k-good in optimal constant time.*

Proof. Let A be the ordered set of the h-size DS. For each k-block, there are initially at most $2k/h$ candidates and we have $h/2$ processors per candidate. For each such candidate in the k-block, make h comparisons with the positions of A. If a candidate has a mismatch, it provides a witness against the candidate. Find the leftmost (ls) and rightmost (rs) survivors in the k-block. By the definition of DS, every survivor i between ls and rs has at least one mismatch in the DS positions of ls or rs. For each such i, make $2h$ comparisons with the DS positions of ls and rs and find a witness against it. ∎

LEMMA 2. *If T is $m^{1/k}$-good for $k > 1$ and the witnesses of the pattern are given, T can be made $m/4$-good in time $O(\log k)$ with $O(n \log k)$ operations.*

Proof. Run $\log k$ rounds of the following until T is $m/4$-good. In a round we start with at most two candidates per i-block (i-good) and end with at most one per i^2-block (i^2-good) by performing at most $4i^2$ duels in each i^2-block. Duels find witnesses to non-occurrences. (Note that we actually start each round after the first with at most one candidate per i-block.) ∎

Given a string x of length r and a number $\ell \leq 2\sqrt{r}$, FIND-SUB finds the first nonperiodic substring z of length ℓ and computes witnesses of z if such z exists, and otherwise computes the period p of x of length less than $\ell/2$ and witnesses against non-multiples of p in optimal constant time.

Procedure FIND-SUB:

1. Naively check if each of the first $\ell/2$ positions is a period of the prefix of x of length ℓ and compute witnesses against nonperiods.
2. If none is a period, z is the prefix of x of length ℓ. Stop.
3. If there are periods, find the shortest one p. Find the smallest prefix y of x such that p is not the period of y.
4. If p is the period of x (y does not exist), witnesses against non-multiples of p are easily computed from the witnesses of the first p-block. Stop.
5. Otherwise (y exists; i.e., there is a mismatch with period p), z is the suffix of y of length ℓ. (z is nonperiodic [8].) Naively compute the witnesses of z.

The first application of the constant-time DS computation is a simple string-matching algorithm with constant-time search and $O(\log \log m)$-time preprocessing called CONST-MATCH. In order to be used in §4, CONST-MATCH solves the following problem: Given a (non-4-periodic) pattern P of length m and its witnesses and text T, find all occurrences of P in T and witnesses to non-occurrences in optimal constant time. Initially, T is 2-good.

Procedure CONST-MATCH:

1. Find the first nonperiodic substring x of P of length $r = m^{1/3}$ and the first nonperiodic substring x' of x of length $2 \log r$ using FIND-SUB, which also computes witnesses of x and x'. Steps 2–4 use Lemma 1.
2. Use the constant-size DS of x for $\log \log r$ shifts to make T $\log \log r$-good.
3. Use the $\log \log r$-size DS of x' for $\log r$ shifts to make T $\log r$-good.
4. Use the $\log r$-size DS of x for $r/2$ shifts to make T $r/2$-good.
5. Perform two rounds of duels to make T $m/4$-good (Lemma 2). Then check the surviving candidates by naive comparisons.

THEOREM 3. *Given the witnesses of the pattern, Procedure CONST-MATCH performs string matching in optimal constant time.*

COROLLARY 1. *Using CONST-MATCH, we get a string-matching algorithm with constant-time text search and $O(\log\log m)$-time preprocessing.*

The second application is a deterministic $O(k)$-time algorithm for finding a subpattern of length $m^{1-2^{-k}}$ in the text. This application may seem somewhat contrived, but it is useful in the next two applications. We will twice use the case $k = 1$. Let P' be a subpattern of P of length $m^{1-2^{-k}}$. We first compute witnesses of P' in $O(k)$ time by the first k rounds of the preprocessing of [2] and then find all occurrences of P' in the text by CONST-MATCH. (In case P' is 4-periodic, we use the reduction of [2].)

COROLLARY 2. *Using CONST-MATCH, we get an $O(k)$-time algorithm using n processors for finding all occurrences of a given subpattern of length $m^{1-2^{-k}}$ in the text, or stated differently we get a deterministic $O(k)$-time algorithm using n processors for string matching in case $m = O(n^{1-2^{-k}})$.*

The third application is a simple string-matching algorithm that has constant time with high probability for random input. Let P' be the prefix of the pattern of length $\sqrt{m}$. Compute witnesses of P' in constant time. Find all occurrences of P' in the text by CONST-MATCH. Being able to match P' of length $\sqrt{m}$ in constant time gives us a constant expected time parallel algorithm for random text even if we sequentially check the remaining symbols. The probability that the time is larger than some small constant is exponentially small.

COROLLARY 3. *Using CONST-MATCH, we get a simple string-matching algorithm that has constant time with high probability for random input.*

4. Constant-Time Randomized String Matching. The main application of the constant-time DS computation is a constant expected time Las-Vegas algorithm for computing the witnesses of the pattern. Together with CONST-MATCH we obtain a constant expected time Las-Vegas algorithm for string matching including preprocessing, solving the main open problem remaining in string matching. Thus, randomization is used to 'beat' the deterministic $\Omega(\log\log m)$ lower bound for witness computation and string matching [3].

We introduce a notion of *pseudo period* (also used in [4]). It has an operational definition: Given a string x of length r, if we compute witnesses against all $i < r/4$ except for multiples of q we say that q is a pseudo period of x. It follows from this definition that if x is 4-periodic, q must divide the period of x. Procedure FIND-PSEUDO computes a large pseudo period of x.

Procedure FIND-PSEUDO:

1. Run FIND-SUB with x and $\ell = 2\sqrt{r}$. If the period q of x is $< \sqrt{r}$, stop. Otherwise, FIND-SUB finds the first nonperiodic substring z of x of length $2\sqrt{r}$ and computes witnesses of z.

2. Using CONST-MATCH, find all occurrences of z in x and witnesses to non-occurrences.

3. Construct the $(r - 2\sqrt{r})$-bit binary string x' such that for $0 \le i < r - 2\sqrt{r}$, $x'[i] = 1$ if i is an occurrence of z, $x'[i] = 0$ otherwise. Compute the period q of x' in case $q < r/4$ and in addition all witnesses of x' against non-periods of x' smaller than $r/4$. Note that if $q < r/4$ all periods of x' smaller than $r/4$ are multiples of q. This computation exploits the special form of x'; it contains at most $\sqrt{r}$ 1's with distance of at least $\sqrt{r}$ between them since z is nonperiodic and of length $2\sqrt{r}$. Thus, we can compute witnesses of x' by considering only the 1's.

3.1. Divide string x' into disjoint $\sqrt{r}$-blocks.

3.2. There is at most one 1 in every block. Record the position of the 1 in the given block in the first element of that block. (Now every processor can read from the first element of a block the position of 1 in that block.)

3.3. Let t be the position of the first 1 in x'. For position $i < t$, t is a witness against $t - i$. If $t \geq r/4$ this substep is done. Note that we already have all witnesses against $i < r/4$. Otherwise consider $i \geq 2t$ (i.e., $i - t \geq t$). If $x'[i] = 0$ then i is a witness against $i - t$. If $x'[i] = 1$, $i - t$ is a potential period of x' since it shifts the first 1 to a 1. Use the $\sqrt{r}$ processors of the block of i to check if $i - t$ is a real period of x' by checking for all k such that $x'[k] = 1$ whether ($x'[k+i-t] = 1$ or $k + i - t \geq r - 2\sqrt{r}$) and ($x'[k - i + t] = 1$ or $k - i + t < 0$). If all these tests succeed, $i - t$ is a period of x'. If the test with k fails, $k + i - t$ or k is a witness against $i - t$. Compute q, the smallest period of x'.

3.4. From witnesses of x' compute witnesses of x. Let w be the witness of x' against i. Assume $x'[w] = 0$ and $x'[w - i] = 1$. (The other case is similar.) Since $x'[w] = 0$, w is a non-occurrence of z in x. Let j be the witness to non-occurrence of z at w. One can verify that $w + j$ is a witness of x against i.

Procedure FIND-PSEUDO computes q which satisfies:

P1. If $q \leq \sqrt{r}$, q is the real period of x.

P2. If $q > \sqrt{r}$, then q is a pseudo period of x.

Given q integers, let $\mathrm{LCM}_{k,q}$ be the minimum of $k/4$ and the LCM (least common multiple) of the q numbers. Given a $k \times q$ array B of symbols and for every column c its pseudo period $q_c < k/4$ and witnesses against non-multiples of q_c, Procedure FIND-LCM computes $\mathrm{LCM}_{k,q}$ of the pseudo periods and witnesses against non-multiples of $\mathrm{LCM}_{k,q}$ smaller than $k/4$ in constant time with kq processors as follows.

Procedure FIND-LCM:

1. Construct a $(k/4 - 1) \times q$ array B': in the c-th column of B', write 1 in the multiples of the pseudo period q_c and 0 in other places.

2. For each row that is not all 1's, any entry 0 provides a witness against the row number.

3. If there is a row with all entries 1's, return the smallest such row, otherwise return $k/4$.

Let p be the period of the pattern P. Recall that the main problem is to compute $\min(p, m/4)$ and the witnesses against all non-periods i, $1 \leq i < m/4$. These non-periods are exactly all the non-multiples of p smaller than $m/4$. In the 4-periodic case witnesses against all i, $1 \leq i < p$, are sufficient. The other witnesses against $i < m/4$ can be computed from them in constant time.

We first describe a deterministic $O(\log\log m)$-time algorithm for the main problem. The algorithm consists of rounds and maintains a variable q. The invariant at the end of a round is that q is a pseudo period of x. Initially, $q = 1$. We describe one round of the algorithm. A witness against i found during the round is a witness of P against iq.

1. Divide P into blocks of size q and make an array B of $k = m/q$ rows and q columns, where column j contains all $P[i]$ for all $i \equiv j \bmod q$.

2. For each column c of B, find q_c, its pseudo period, and witnesses against non-multiples of q_c using FIND-PSEUDO.

3. If all pseudo periods are $\le \sqrt{k}$, all pseudo periods are real periods. Using FIND-LCM, compute $\mathrm{LCM}_{k,q}$ and witnesses against non-multiples of $\mathrm{LCM}_{k,q}$ smaller than $k/4$. The period of P that we compute is $q \cdot \mathrm{LCM}_{k,q}$. Stop.

4. Otherwise, choose a column c with $q_c > \sqrt{k}$. Witnesses against non-multiples of q_c were computed in Step 2. $q \leftarrow q \cdot q_c$.

5. If $q < m/4$, then go to the next round; else stop.

Note that in the first round we have one column and compute a pseudo period of P by FIND-PSEUDO. In subsequent rounds $q \cdot q_c$ is a pseudo period because we compute witnesses for all non-multiples of $q \cdot q_c$. Since the new value k is at most $\sqrt{k}$, there are at most $O(\log \log m)$ rounds.

This algorithm follows the structure of the lower bound proof [3]. That proof uses the notion of 'possible period length', which is the minimum number that can still be the period of the pattern based on the results of the comparisons so far. The lower bound argument maintains a possible period length $q \le m^{1-4^{-i}}$ in round i and forces any algorithm to have at least $\frac{1}{4} \log \log m$ rounds. Here, we compute a pseudo period q that may not be a period length, but must divide it in case the pattern is 4-periodic. $q > m^{1-2^{-i}}$ in round i and the algorithm finds the period in at most $\log \log m$ rounds.

COROLLARY 4. *If P is 4-periodic and its period has a constant number of prime divisors, we can compute witnesses and do string matching in optimal deterministic constant time.*

We now describe an $O(1)$ expected time randomized algorithm for the main problem. Execute the first three rounds of the deterministic algorithm and then execute Round 4 below until it stops. At the beginning of Round 4, q is a pseudo period of x, and B is the $k \times q$ array, $k = m/q$, created by Step 1 of the deterministic algorithm. We have $q > m^{7/8}$ and $k = m/q < m^{1/8}$.

Round 4:

1. Randomly choose a multiset of $s = m/k^2$ columns from B, i.e., each of s processors chooses a random column number from the set $\{1, \ldots, q\}$. Find the period of each chosen column naively with k^2 processors. Using naive comparisons also compute witnesses against nonperiods. Using FIND-LCM, compute $h = \mathrm{LCM}_{k,s}$ and witnesses against non-multiples of h.

2. If $h = k/4$, the pattern P is not 4-periodic. Stop.

3. Otherwise, check if h is a period of each column of B. If h is a period in all columns, qh is the period of P; Stop. Otherwise, let C be the set of columns where h is not a period.

4. Using Hagerup's $(1+\epsilon)$-PAC [9], try to compact C into the set C' of size $m^{3/4}$. If the compaction fails, try again Round 4 starting from Step 1.

5. If the compaction is successful, compute all periods of columns in C' naively (we have enough processors because $m^{3/4}k^2 < m$). Using naive comparisons also compute witnesses against nonperiods. Using FIND-LCM, compute $h' = \mathrm{LCM}_{k,m^{3/4}}$ of these periods and witnesses against non-multiples of h'. The period of P that we compute is $\min(m/4, q \cdot \mathrm{LCM}(h, h'))$.

LEMMA 3. *With a very high probability the size of C in Round 4 is smaller than $m^{1/2}$.*

Proof. Let Q be the multiset of the q real periods of the columns of B. We call a period good if it appears in Q at least $q^{1/3}$ times; otherwise call it bad.

For a good period, the probability that it is not chosen among the s random choices is at most $(1 - q^{1/3}/q)^s < e^{-m^{1/6}}$, since $s = m/k^2 = q^2/m > q^{2/3}m^{1/6}$.

Thus the probability that there is some good period that was not chosen is at most $qe^{-m^{1/6}} < me^{-m^{1/6}} << 1$.

We showed that with a very high probability only the bad periods will remain in C. Since there are only k different values for periods, the number of occurrences of bad periods in Q is at most $q^{1/3}k = m/q^{2/3} < m/m^{7/12} < m^{1/2}$. ☐

It follows from Lemma 3 that the PAC will fail (and as a result Round 4 will be repeated) with a very small probability and the expected number of rounds is smaller than 5.

LEMMA 4. *The randomized algorithm is an optimal Las-Vegas parallel algorithm for computing the period and the witnesses of the pattern P. It has constant time with high probability.*

THEOREM 4. *Together with CONST-MATCH we have a randomized optimal Las-Vegas parallel algorithm for string matching, including preprocessing. It has constant time with high probability (and thus constant expected time).*

5. Conclusion. We have shown how to compute deterministic samples for a part of the pattern in constant time, and obtained a deterministic string matching algorithm which is provably the best in both preprocessing and text search. We use them to obtain a simple constant expected time algorithm for random input and a more sophisticated randomized algorithm for string matching with constant expected time.

The randomized algorithm for string matching leads to constant expected time randomized algorithms for several related problems. If we solve $\log m$ (or even more) string matching problems at the same time, the expected time is still a constant. This converts the algorithms for finding all periods, squares and palindromes [1] into constant expected time randomized algorithms.

We believe that there may be more applications for our super fast deterministic sampling. DS can be considered as a deterministic fingerprint. No constant-time algorithm is known for the conventional fingerprint computation on a CRCW PRAM. On the EREW PRAM our algorithm can be translated into an optimal $O(\log m)$ time algorithm [6]; it works for any alphabet since it only performs comparisons on the input symbols. The conventional fingerprint has an optimal randomized $O(\log m)$ time algorithm on the EREW PRAM and it works only when the alphabet is given and is of small size [10]. On the other hand, our DS is only for a subpattern and while it is computed deterministically one still needs randomization for computing the witnesses. (The conventional fingerprint, while randomized, is applied deterministically.)

Acknowledgements. We thank Noga Alon and Yossi Matias for helpful suggestions.

REFERENCES

[1] A. Apostolico, D. Breslauer, and Z. Galil, *Optimal parallel algorithms for periods, palindromes and squares*, Proc. 19th Int. Colloq. Automata Languages and Programming, Lecture Notes in Computer Science, Vol. 623, 1992, pp. 296–307.

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

[3] ———, *A lower bound for parallel string matching*, SIAM J. Comput. 21 (1992), 856–862.

[4] B.S. Chlebus and L. Gasieniec, *Optimal pattern matching on meshes*, Proc. 11th Symp. Theoretical Aspects of Computer Science, 1994, 213–224.

[5] R. Cole, M. Crochemore, Z. Galil, L. Gasieniec, R. Hariharan, S. Muthukrishnan, K. Park, and W. Rytter, *Optimally fast parallel algorithms for preprocessing and pattern matching in one*

and two dimensions, Proc. 34th IEEE Symp. Found. Computer Science, 1993, 248–258.

[6] A. Czumaj, Z. Galil, L. Gasieniec, K. Park, and W. Plandowski, *Work-time optimal parallel algorithms for string problems*, Proc. 27th ACM Symp. Theory of Computing, 1994, 713–722.

[7] F.E. Fich, P. Ragde, and A. Wigderson, *Relations between concurrent-write models of parallel computation*, SIAM J. Comput. 17 (1988), 606–627.

[8] Z. Galil, *A constant-time optimal parallel string-matching algorithm*, J. Assoc. Comput. Mach. 42 (1995), 908–918.

[9] T. Hagerup, *On a compaction theorem of Ragde*, Inform. Process. Lett. 43 (1992), 335–340.

[10] R.M. Karp and M.O. Rabin, *Efficient randomized pattern-matching algorithms*, IBM J. Res. and Dev. (1987), 249–260.

[11] P. Ragde, *The parallel simplicity of compaction and chaining*, J. Algorithms 14 (1993), 371–380.

[12] U. Vishkin, *Optimal parallel pattern matching in strings*, Inform. and Control 67 (1985), 91–113.

[13] ———, *Deterministic sampling–a new technique for fast pattern matching*, SIAM J. Comput. 20 (1991), 22–40.