Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions

Richard Cole[1]
Courant Institute, NYU

Maxime Crochemore
Université de Marne la Vallée

Zvi Galil[2]
Columbia University and
Tel-Aviv University

Leszek Gąsieniec[3]
Warsaw University

Ramesh Hariharan[4]
Courant Institute, NYU

S. Muthukrishnan[5]
Courant Institute, NYU

Kunsoo Park[6]
King's College London

Wojciech Rytter[7]
Warsaw University

Abstract

All algorithms below are optimal alphabet-independent parallel CRCW PRAM algorithms.

In one dimension: Given a pattern string of length m for the string-matching problem, we design an algorithm that computes a deterministic sample of a sufficiently long substring in **constant time**. This problem used to be a 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.

1. Improving the preprocessing of the constant-time text search algorithm [12] from $O(\log^2 m/\log\log m)$ to $O(\log\log m)$, which is now best possible.
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 probabilistic string-matching algorithm that has constant time with high probability for random input.
4. A **constant expected time Las-Vegas** algorithm for computing the period of the pattern and all witnesses and thus string matching itself, solving the main open problem remaining in string matching.

In two dimensions: Given an $m \times m$ pattern, we present the first optimal alphabet-independent parallel algorithm for two-dimensional pattern matching which requires constant-time text processing following $O(\log\log m)$-time preprocessing of the pattern. This algorithm depends upon three new time- and work-optimal results, one of which is the constant-time deterministic sample computation above.

The second is a simple constant-time two-dimensional pattern matching algorithm, given a deterministic sample and two-dimensional witnesses. The best previous optimal two-dimensional text search takes $O(\log\log m)$ time [3].

The third is an $O(\log\log m)$-time algorithm that computes two-dimensional witnesses. No optimal parallel algorithm was known before. The best previous algorithm takes $O(\log m)$ time and uses $O(m^2 \log m)$ operations and more than linear space [1].

1. Introduction

The (one-dimensional) string-matching problem is: Given pattern $P[0..m-1]$ and text $T[0..n-1]$, find all occurrences of P in T. The two-dimensional pattern matching (2D matching) problem is: Given pattern $P[0..m-1, 0..m-1]$ and text $T[0..n-1, 0..n-1]$, find all occurrences of P in T. The pattern and the text contain symbols from an alphabet Σ. The two-dimensional problem and all algorithms mentioned below (including ours) can be easily generalized to rectangular arrays. An alphabet-dependent algorithm is one whose run-time depends on $|\Sigma|$ (typically via a factor of $\log(\min(|\Sigma|, m))$.

[1] Work supported by NSF grants CCR-8902221, CCR-8906949 and CCR-9202900.
[2] Partially supported by NSF grant CCR-90-14605 and CISE Institutional Infrastructure Grant CDA-90-24735.
[3] Partially supported by KBN grant 2-11-90-91-01 and EC Cooperative Action IC-1000 (project ALTEC).
[4] Work supported by NSF grants CCR-8902221, CCR-8906949 and CCR-9202900.
[5] Work supported by NSF/DARPA grant CCR-8906949 and NSF grant CCR-9103953.
[6] Partially supported by SERC grant GR/J17944.
[7] Work done while visiting UC Riverside. Supported by the project ALTEC.

Most one- and two-dimensional pattern 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 one-dimensional string matching a constant-time optimal parallel algorithm is known for the search [12]. On the other hand, an $\Omega(\log\log m)$ lower bound is known for the entire problem [9]. One can easily show that this lower bound applies to the two-dimensional problem.

All algorithms below are optimal alphabet-independent parallel algorithms on a CRCW PRAM. We use the weakest version (called Common in [11]) in which the only concurrent writes allowed are of the same value 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
    x -->    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
   0 1 0 1 0 0 0|1 0 1 1 1 0 1 0|1
```

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

We consider a nonperiodic pattern string x of length m. A deterministic sample (DS) of x for k shifts is an ordered set A of positions and a number f 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 x in the positions of the set A. The size of a DS is the size of the ordered set A. (See Figure 1.) Vishkin [21] introduced 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. Assume we can somehow reduce the number of candidates (for occurrences) to one in every $\log m$-block of the text. Then in one step we compare the set A of DS for all candidates. The immediate survivors are the candidates for which we have a match. Now consider an $m/2$-block of the text. By the definition of DS we can eliminate all but the leftmost and rightmost survivors in the block. Then in one more step we have enough processors to naively check the (at most two) remaining candidates per $m/2$-block. All these steps take constant time.

This method was used in a constant-time optimal text search [12]. However, the optimal algorithm suggested by Vishkin and used in [12] 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 [8] (for which also an $\Omega(\log\log m)$ lower bound was proved [9]) 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 Section 2 we present the first main result of this paper, a constant-time deterministic algorithm for computing deterministic samples. It is actually a fourth in a sequence of unpublished improvements to $O(\log m)$, $O(\log^{\epsilon} m)$ for any constant $\epsilon < 1$, $O(\log\log m)$ and $O(1)$. In Section 2 we also show how to compute in constant time a **constant-size** DS for $O(\log\log m)$ shifts. This DS will be crucial to our main application in Section 4.

Section 3 contains three applications of the constant-time algorithm for DS. The first 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 obtains the best possible time in both preprocessing and text search. Although the DS computation used to be the bottleneck, the text search algorithm [12] 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. Nevertheless, this first application is immediate using $O(\log\log m)$ time for constructing the hitting set.

In order to derive a randomized constant-time algorithm we cannot afford to compute a hitting set. Instead, we use the constant-size DS for $O(\log\log m)$ shifts of Section 2 to design an alternative algorithm called CONST-MATCH. Given the witnesses, CONST-MATCH performs constant-time string matching **without** a hitting set and is a somewhat simpler deterministic algorithm with both best preprocessing and text search. The two other applications follow almost immediately by using CONST-MATCH.

The second application is a deterministic string-matching algorithm of time $O(k)$ for a subpattern of size $m^{1-2^{-k}}$. Stated differently, we have a **deterministic** $O(k)$-time string-matching algorithm for the case that $m = O(n^{1-2^{-k}})$. This application is useful in the next two applications. We will twice use the case $k = 1$.

The third application is a simple probabilistic string-matching algorithm that has constant time with high probability and constant expected time for random input.

In Section 4 we describe in some detail our main application: A constant expected time Las-Vegas algorithm for computing the witnesses. Together with CONST-MATCH we obtain a constant expected time Las-Vegas algorithm for string matching **including pre-**

processing, solving the main open problem remaining in string matching. Deterministically an $\Omega(\log\log m)$ lower bound is known for witness computation and string matching [9]. This algorithm was designed based on the lower bound argument; randomization was 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.

We present a new algorithm for two-dimensional pattern matching which requires constant-time text processing following $O(\log\log m)$-time preprocessing of the pattern. This is the best possible. The above result uses the constant-time DS of Section 2 and two very different algorithms. The first is a simple constant-time optimal algorithm for the text search. We describe it in Section 5. The previous best result known was the optimal $O(\log\log m)$-time parallel algorithm of Amir, Benson and Farach [3]. The second algorithm is a more involved algorithm for computing the two-dimensional witnesses of the pattern. This is the most expensive part of the pattern processing and was needed also by the algorithm in [3]. There was no optimal parallel algorithm known for this problem. Designing this algorithm we had to go around some open problems and to extend the theory of two-dimensional periodicity. Due to space limitations we only sketch the witness computation in Section 6.

Unlike in one-dimensional string matching, the pattern processing in 2D matching (i.e., two-dimensional witness computation) presents greater challenge than the text search. In sequential computation, Baker [6] and Bird [7] gave an alphabet-dependent linear-time algorithm (i.e., the time is linear only in case of a small alphabet size). Then Amir, Benson and Farach [2] gave an alphabet-independent linear-time text-search algorithm, but its pattern processing depends on the alphabet and uses techniques developed for one-dimensional string matching. Very recently, an alphabet-independent linear-time pattern processing was developed [13]. It relies heavily on a new theory on two-dimensional periodicity [1,13].

A similar difficulty repeated in parallel computation; the text search turns out to be relatively easy, relying mostly on one-dimensional techniques. The best parallel algorithm for the pattern processing was an $O(\log m)$ time algorithm using $O(m^2)$ processors due to Amir and Benson [1] which uses suffix trees for strings. This solution has two problems: (1) it is not optimal, requiring $O(m^2 \log m)$ total operations even for alphabets of constant size; (2) the space used by the parallel suffix tree algorithm is not linear.

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 [11]. Our algorithms will use the constant-time deterministic polynomial approximate compaction (PAC) of Ragde [19] and its improvement by Hagerup [14]. 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 $\epsilon > 0$.

2. Constant-Time Deterministic Sampling

A string u is a *period* of a string x if x is a prefix of u^k for some k or equivalently if x is a prefix of ux. The shortest period of a string x is *the period* of x. If the period of x is shorter than a half of x, x is called *periodic*. A substring of x of length i is called an *i-block* of x. The positions of all strings start from 0.

When we say that we compute the period of a string x of length r, we will mean computing $r' = \min(r/2, p)$, where p is the period length of x. When we say that we compute witnesses of x, we will mean computing r' and for every $1 \le i < r'$ a position w such that $x[w] \ne x[w-i]$. Such a w is called a *witness* of x against (the periodicity of) i [20]. (It is possible to compute large periods and witnesses using techniques of [5]. But we will not need them here.) If position i of T is not an occurrence of P, a position w such that $T[w] \ne P[w-i]$ is called a *witness to* non-occurrence i.

Vishkin [21] introduced the notion of a deterministic sample, which is crucial for very fast optimal parallel search of the pattern in the text. Let x be a nonperiodic string of length r. A *deterministic sample* (DS) of x for k shifts is an ordered set A of positions and a number f 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 x in the positions of the set A. The *size* of a DS is the size of the ordered set A. Vishkin [21] proved the existence of a $\log r$-size DS for $r/2$ shifts. We construct two kinds of DS's in constant time: a $\log k$-size DS for k ($\le r/2$) shifts 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 [21], 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 [21,12]. 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

$p = j - i < k \leq r/2$. 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: The periodicity cannot extend both to the right and the left since x is nonperiodic. 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). q can be found in optimal constant time. Now the DS is $A = \{q, q+p\}$ and $f = 1$ because in the first copy we have an inequality in positions $q, q+p$ and in the next $k-1$ copies to the right we have equalities. 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 corresponding block. 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 $f = k - b$.

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 of blocks, find in constant time the smallest index ℓ such that the two blocks 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 [19] 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, $k \leq r/2$, for a nonperiodic string of size r in constant time using r^3 processors and r^2 space.

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 Section 4.

Case 1. If there exists position i in x such that $x[i] \neq x[i+j]$ (or $x[i] \neq x[i-j]$) for every $1 \leq j < \log r$, 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 $\log\log r$ size in the first half of x. Since there are $(\log r)^{\log\log r}$ different strings of size $\log\log r$ over $\log r$ symbols, and $(\log r)^{\log\log r} << r/2$, some substrings of $\log\log r$ size 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 $x[q] \neq x[q+p]$ 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 size 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 $O(1)$-time randomized string-matching algorithm in Section 4. Thanks to a well known reduction [8], we can assume without loss of generality that the pattern in a string matching problem is nonperiodic.

The algorithm will maintain a subset of text positions, referred to as *candidates*. A candidate can still start an occurrence of the pattern. All other positions have witnesses to non-occurrences. Consider two candidates $i < j$ such that w is a witness against $j - i$ (i.e., $P[w] \neq P[w - j + i]$).

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

The two tests above are called a *duel* between i and j. By a duel we can remove one or both of the candidates. If all h-blocks of the text T have 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. For each candidate in the k-block, make h comparisons with the positions of A using h processors. If a candidate has a mismatch, it provides a witness against the candidate. Find the leftmost and rightmost survivors in the k-block. By the definition of DS, every survivor i in the middle has at least one mismatch in the positions of the leftmost or rightmost survivor. Make $2h$ comparisons with the positions of the leftmost survivor and the rightmost survivor and find a witness against i. $\square$

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

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

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 in the prefix of x of length ℓ and compute witnesses against nonperiods.
2. If none is a period, z is the prefix of length ℓ. Stop.
3. If there are periods, find the shortest one p. Extend the period p to the end of x.
4. If p extends to the end of x, p is the period of x. Witnesses against non-multiples of p are easily computed from the witnesses of the first p-block. Stop.
5. Otherwise (there is a mismatch with period p), z is the nonperiodic substring of length ℓ ending with and including the first mismatch, which provides 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 Section 4, CONST-MATCH solves the following problem: Given a (nonperiodic) pattern P of size 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 duels to make T $m/2$-good (Lemma 2) and 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 string-matching algorithm of time $O(k)$ for a subpattern of size $m^{1-2^{-k}}$. Let P' be a subpattern of P of size $m^{1-2^{-k}}$. We first compute witnesses of P' in $O(k)$ time by the first k rounds of the preprocessing of [8] and then find all occurrence of P' in the text by CONST-MATCH. (In case P' is periodic, we use the reduction of [8].)

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

The third application is a simple probabilistic 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.

Corollary 3. Using CONST-MATCH, we get a probabilistic string-matching algorithm that has constant time with high probability and constant expected time 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 [9].

We introduce a notion of *pseudo period* (also used in [10]) as follows. FIND-PSEUDO solves the following problem in optimal constant time: Given a string x of length r, find a number q (called the pseudo period) and witnesses against non-multiples of q.

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. Let b be the start position of z in x.
2. Using CONST-MATCH, find all occurrences of z in x and witnesses to non-occurrences.
3. Construct a string x' of r bits: $x'[i] = 1$ if $i + b$ is an occurrence of z, $x'[i] = 0$ otherwise. Note that for all 0's of x', witnesses have been computed in Step 2. Compute the period q of x' and witnesses against non-multiples of q as follows. (The string x' has a special form. Because nonperiodic z has length $2\sqrt{r}$, there are at most $\sqrt{r}$ occurrences of z in x. So x' contains at most $\sqrt{r}$ 1's with distances $\geq \sqrt{r}$ between neighboring 1's.)
   - 3.1. Divide string x' into $\sqrt{r}$-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. Using $\sqrt{r}$ processors for each block, check if the 1 in the block is a period of x' by comparing positions of 1's in all blocks. The shortest one among the periods is the period q of x'.
   - 3.4. If the 1 in a block is not a period (i.e., not a multiple of q), find the first mismatched position of 1 (to 0) in x' and make $|s| = 2\sqrt{r}$ comparisons between the occurrence and the non-occurrence (b positions to the right of the mismatched 1) of z in x, which gives a witness against the 1.

Lemma 3. If the period p of x is $< r/2$, then the pseudo period q divides p.

Proof. The period p is also a pseudo period (period of x') and thus $q \leq p < r/2$. But then $\mathrm{GCD}(p, q) \leq q$ is a pseudo period and therefore q divides p. □

By FIND-PSEUDO and Lemma 3, we have the following properties of the pseudo period q of x.

- P1. If $q \leq \sqrt{r}$, q is the real period of x.
- P2. If $q > \sqrt{r}$ and $p < r/2$ where p is the period of x, p is a multiple of q.

Given q numbers each of which is $< k/2$, let $\mathrm{LCM}_{k,q}$ be the minimum of $k/2$ and the LCM of the q numbers. Given a $k \times q$ array B of symbols and for every column c its pseudo period q_c 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 $k/2$ in constant time with kq processors as follows.

Procedure FIND-LCM:

1. If one of the numbers is $\geq k/2$, we are done.
2. Otherwise, construct a $k \times q$ 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.
3. The smallest row $< k/2$ that has all 1's is the $\mathrm{LCM}_{k,q}$.
4. For each row that is not all 1's, any entry 0 provides a witness against the row number.

Recall that the main problem is to compute the period and the witnesses of the pattern P. 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 beginning of a round is:

A. If P is periodic then its period is a multiple of q.
B. Witnesses against all non-multiples of q have been computed.

Initially, $q = 1$. We describe one round of the algorithm. A witness against i found during the round is a witness against iq in P.

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 of B, find its pseudo period and witnesses against non-multiples of the pseudo period using FIND-PSEUDO. If all pseudo periods are $\leq \sqrt{k}$, all pseudo periods are real periods. Using FIND-LCM, computes $\mathrm{LCM}_{k,q}$ and witnesses against non-multiples of $\mathrm{LCM}_{k,q}$ smaller than $k/2$. The period of P that we compute is $q \cdot \mathrm{LCM}_{k,q}$. (Recall that by computing the period p we mean computing $\min(m/2, p)$.) Stop.
3. If there exists at least one column whose pseudo period is $> \sqrt{k}$, choose such a column c, and let h be the pseudo period of column c. For non-multiples of h, column c provides witnesses. Go to the next round with $q \leftarrow qh$.

Since the new value k is at most $\sqrt{k}$, there are at most $O(\log\log m)$ rounds. The lower bound argument maintains a potential period $q \leq m^{1-4^{-i}}$ in round i and forces any algorithm to have at least $\frac{1}{4}\log\log m$ rounds [9], while our algorithm maintains a potential period $q > m^{1-2^{-i}}$ in round i and finds the period in at most $\log\log m$ rounds.

Fact 1. If P is periodic and its period has a constant number of prime divisors, we can solve the main problem 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 divisor of the period p of P if $p < m/2$, 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 $s = m/k^2$ columns from B and find the period of each chosen column naively with k^2 processors. Naive comparisons also computes witnesses against nonperiods. Using FIND-LCM, compute $h = \mathrm{LCM}_{k,s}$ and witnesses against non-multiples of h.
2. If $h = k/2$, the pattern P is nonperiodic. 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. Otherwise, let C be the set of columns where h is not a period.
4. Using Hagerup's $(1+\epsilon)$-PAC [14], 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 $m^{3/4}k^2 < m$). Naive comparisons also computes 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 $\min(m/2, q \cdot \mathrm{LCM}(h, h'))$.

Lemma 4. 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, 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 4 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.

Theorem 4. The randomized algorithm is an optimal constant-time Las-Vegas parallel algorithm for computing the period and the witnesses of the pattern P.

Corollary 4. Together with CONST-MATCH we have an optimal constant-time Las-Vegas parallel algorithm for string matching, including preprocessing.

5. Fast Two-Dimensional Text Search

We show how to find all occurrences of a 2D pattern P of size $m \times m$ in a text T of size $n \times n$ in $O(1)$ time and $O(n^2)$ work following preprocessing of the pattern.

Previous sequential and parallel algorithms for 2D text processing either involve a reduction to multiple string matching [6,7,17] or use suffix trees [4]. By contrast, our approach essentially provides a constant time Turing-reduction from 2D matching to string matching, which itself can be solved optimally in $O(1)$ time [12]. We do not use any 2D periodicity properties used in previous sequential and parallel algorithms for 2D pattern matching [13,2,3]. Instead we use the weaker notion of h-periodicity (or horizontal periodicity), which is just string periodicity generalized to 2D along the horizontal dimension. We also use the notions of *witnesses* [20] and *deterministic samples* [21] for string matching, again suitably generalized to 2D.

Definitions. Let $m' = \lfloor \frac{m}{2} \rfloor$. The pattern instance P_c *associated* with text location c is a copy of P placed with top left corner at c. If P_c potentially matches the text then c is called a *source*. Source c is said to be *eliminated* when P_c is determined not to match the text; otherwise it *survives*. Sources c, d are *compatible* if P_c and P_d match wherever they overlap. A *duel* between non-compatible sources c and d is a comparison of the character $\bar{a}$ at location $a \in P_c$ with the aligned text character, where $\bar{a}$ differs from the aligned character in P_d; this comparison eliminates one of c, d. If c and d are separated by vector v then the pair of locations $(a, a+v)$ is a *witness* against v in P. v is a *period vector* of P if it does not have a witness in P. A period vector of P that is horizontal is a *horizontal period vector* of P. P is *h-aperiodic* if it has no horizontal period vector of length at most m', and *h-periodic* otherwise.

We assume that P is h-aperiodic; the algorithm can be generalized easily to handle the h-periodic case. The following lemmas are crucial.

Lemma 5. Some row r of P (considered as a string) has period at least $\frac{\log m'}{2}$.

Lemma 6. There exist numbers $e, f,\ e + f = m'$, and a set of at most $\log m'$ locations in P (called the *h-sample*) with the following property: if the characters at the locations in the h-sample for P_s match the aligned text characters then there are no occurrences of P with top left corner at those locations in the text that are in the same row as s and either at most $e - 1$ positions to its left or at most f positions to its right.

The algorithm starts with all text locations as sources. The text T is divided into disjoint blocks of size $m' \times m'$. Each text block is processed in parallel in two phases. Consider text block B. Phase 1 is survived only by a set of mutually compatible sources in B. In Phase 2, for each surviving source c, P_c is either verified to be a complete match or eliminated. Each phase takes $O(1)$ time and $O((m')^2)$ work, for a total of $O(n^2)$ work over all text blocks. Phase 1 has the following 3 steps.

Step 1. All rows R of B are processed in parallel. At most $\frac{2m'}{\log m'}$ sources in R survive this step.

Let r be the row of P given by Lemma 5. Using Galil's algorithm [12], all occurrences of r (considered as a string) beginning at some position $r-1$ rows below R in B are found in $O(1)$ time and $O(m')$ work. Since r has period at least $\frac{\log m'}{2}$, at most $\frac{2m'}{\log m'}$ sources in R survive.

Step 2. All rows R of B are processed in parallel. At most two sources in R survive this step.

For each surviving source c in R, all characters at locations in the h-sample of P_c are compared with the aligned text characters in parallel. Then the leftmost and rightmost sources c and d in R with the property that all characters at the locations in the h-samples of both P_c and P_d match the aligned text characters are found. This takes $O(1)$ time and $O(m')$ work for each row R. By Lemma 6, only sources c and d survive in R.

Step 3. B has $O(m')$ surviving sources following Step 2. All pairs of surviving non-compatible sources in B are duelled in parallel in $O(1)$ time and $O((m')^2)$ work. Now, all surviving sources in B are mutually compatible.

Phase 2 proceeds as follows. At the start of Phase 2, for all surviving sources in B, the pattern instances associated with these sources match wherever they overlap. Therefore, each text character d needs to be compared with the aligned pattern character in only one of the surviving pattern instances (the pattern instances associated with the surviving sources) which overlap d. Details are deferred to the full paper.

Preprocessing. The row r in Lemma 5 can be determined in $O(\log\log m)$ time optimally by computing the periods of each row of P [8]. The preprocessing needed to match r in $O(1)$ time by Galil's algorithm [12] can be accomplished in $O(\log\log m)$ time. We show in the next section that witnesses for P can be computed in $O(\log\log m)$ time optimally. Given witnesses, the h-sample for P with $m' = \lfloor m^{1/4} \rfloor$ instead of $m' = \lfloor \frac{m}{2} \rfloor$ can be computed in $O(1)$ time. As in Procedure CONST-MATCH, our text search algorithm can be modified to work with this h-sample in the same time and work bounds.

Theorem 5. All occurrences of P in T can be determined in $O(1)$ time and $O(n^2)$ work following preprocessing of P, which takes $O(\log\log m)$ time and $O(m^2)$ work.

6. Two-Dimensional Witness Computation

A natural approach to computing 2D witnesses is to follow the approach for computing string witnesses [8,5]. There, witnesses for a string s of length m are found in $O(\log\log m)$ stages. Stage i finds witnesses for a prefix of s of length l_i; these witnesses are used in Stage $i+1$ to find witnesses for the prefix of s of length l_{i+1} in constant time. The lower bound of [9] shows that $\Omega(\log\log m)$ stages are required when only $O(m)$ work is allowed.

A natural analog of a prefix of a string is a corner sub-block of the 2D pattern P. The string witness computation algorithm described above is generalized to 2D patterns by designing an algorithm which performs Stage i, i.e., finds witnesses against all non-period vectors of a corner sub-block L_{i+1} of P of size $l_{i+1} \times l_{i+1}$, given witnesses against all non-period vectors of the corner sub-block L_i of P of size $l_i \times l_i$, for appropriate l_i, l_{i+1}. This presents two problems.

Problem 1. Using a construction similar to the string matching lower bound of [9], it is possible to show that, if no additional information is at hand, Stage i takes $\Omega(\log\log l_i)$ time even for the simple case in which all period vectors of L_i are parallel. This remains true if L_i is a central rather than a corner sub-block of L_{i+1}, as in the sequential algorithm of [13]. Since there are $\Theta(\log\log m)$ stages, this approach leads to an algorithm which takes $\Omega((\log\log m)^2)$ time even for this simple case. But our goal is an $O(\log\log m)$ time algorithm.

Problem 2. For the general case, i.e., when the period vectors of L_i are non-parallel, it is not clear how to perform Stage i in $O(\log\log m)$ or even $O(\log m)$ time while preserving work optimality of the overall algorithm. The sequential algorithm of [13] performs Stage i by using a sequential algorithm for computing leftmost string witnesses [18]. Using a similar approach, Stage i can be performed in $O(\log^2 m(\log\log m)^3)$ time using the only known optimal parallel algorithm for computing leftmost string witnesses [15], but this is too slow for our purpose.

Our $O(\log\log m)$ algorithm has the following two key features.

First, Problem 2 is solved in $O(\log\log m)$ time and $O(|L_{i+1}|)$ work for the general case. The solution is based on a new periodicity property of 2D patterns which we prove and use. Note that this immediately gives an $O((\log\log m)^2)$ time algorithm.

Definitions. P' denotes the central sub-block of P of size $\lfloor\frac{m}{2}\rfloor \times \lfloor\frac{m}{2}\rfloor$. The *j-fringe* of P' comprises those locations in P' which are at most distance j from the boundary of P'.

Second, to achieve $O(\log\log m)$ time, we compute with fringes instead of corner or central blocks which circumvents Problem 1. In the main step, witnesses are found against all non-period vectors of P' in $O(\log\log m)$ stages. These stages computes witnesses against all non-period vectors of fringes of P' of successively increasing sizes. Using fringes allows each stage to be performed in $O(1)$ time while keeping the overall work linear. A notable feature of the algorithm is that the witnesses it computes need not be be within P'. Finally, using the solution to Problem 2, witnesses in P, if any, are found against the period vectors of P' in $O(\log\log m)$ time.

Definitions. The arrow end of a vector is called its *head* and the other end its *tail*. A *Quad I vector* has its head below and to the right of its tail. A vector $v \in S$ is said to be *eliminated* when either a witness against v is found or it is determined that there is no such witness; otherwise it *survives*. A *duel* between vectors u, v is a comparison between the characters at locations a and $a+v$, where $(a, a+v-u)$ is a witness against vector $v-u$; if such a witness exists and $a+v$ is within P then the duel eliminates one of u, v.

Solution to Problem 2. Suppose L_i is the top left corner sub-block of L_{i+1}. The following issue is the crux of Problem 2: given a subset S of the Quad I period vectors of L_i of length at most $\frac{l_i}{16}$, to find witnesses in L_{i+1}, if any, against vectors in S in $O(\log\log m)$ time and $O(|L_{i+1}|)$ work.

If all vectors in S are parallel to a line l, say, then L_{i+1} is partitioned into lines parallel to l and witnesses against vectors in S are found in each line independently, using the string witness finding algorithm of [5]; this takes $O(\log\log m)$ time and $O(|L_{i+1}|)$ work. The difficult case is when there are two non-parallel vectors, v_1 and v_2, in S.

Definitions. Consider the lattice formed by the vectors v_1, v_2, with e, the top left corner of L_i, being a vertex in this lattice. A location $a \in P$ is called a *defect* if the character at a differs from the character at location c in the lattice cell bounded by the points $e, e+v_1, e+v_2, e+v_1+v_2$, where $c-a$ is a integer linear combination of v_1, v_2.

Lemma 7. There are no defects in the top left quadrant of L_i.

Lemma 8. Let a, b be locations in P separated by vector $v \in S$. If a is a defect and b is a non-defect then (a, b) is a witness against vector v. If a and b are both non-defects then (a, b) is not a witness against vector v.

Our solution to Problem 2 is obtained by exploiting the distribution of defects in L_{i+1} and using Lemma 8. First, we find witnesses in A, the $\lfloor \frac{l_i}{2} \rfloor \times l_{i+1}$ sized block with top left corner e, against vectors in S in $O(\log\log m)$ time and $O(|A|)$ work. Second, using a similar procedure, we find witnesses in B, the $l_{i+1} \times \lfloor \frac{l_i}{2} \rfloor$ sized block with top left corner e, against vectors in S. We then show that either there are no defects in the bottom half of A and the right half of B or all surviving vectors in S are parallel. In the former case, by Lemma 8 and the length of the vectors in S, $(a-v, a)$ is a witness against all $v \in S$, where a is the leftmost of the topmost defects in $L_{i+1} - (A \cup B)$. The latter case is solved in $O(\log\log m)$ time and $O(|L_{i+1}|)$ work as described earlier. Next, we outline the three main steps in the processing of A.

Step 1. All but l_i of the vectors in S are eliminated in $O(1)$ time and $O(|A|)$ work using the scheme implied by Lemma 9.

Lemma 9. Let α be the leftmost among the bottommost defects in A. Let β be the leftmost defect in A. Let v be a vector in S. If $\alpha - v \in A$ and $\alpha - v$ is not the leftmost defect in its row then either $(\alpha - v, \alpha)$ or $(\gamma, \gamma + v)$ is a witness against v, where γ is the leftmost defect in the row containing $\alpha - v$. If $\alpha - v \notin A$ but $\beta + v \in A$ then $(\beta, \beta + v)$ is a witness against v. If $\alpha - v \notin A$ and $\beta + v \notin A$ then there is no witness against v in A.

Step 2. For each surviving vector $v \in S$ and each location $a \in A$ which is the leftmost defect in its row, if either $(a, a+v)$ or $(a-v, a)$ is a witness against v then v is eliminated. This takes $O(1)$ time and $O(l_i^2) = O(|A|)$ work.

Lemma 10. Let br be the index of the row containing α, defined in Lemma 9. The surviving vectors in S can be partitioned into sets $S_0, \ldots, S_{\log br}$ with the following properties: (1) All the difference vectors of vectors in each set are parallel; (2) Any witness against a vector in set S_j has its lower endpoint in the contiguous $\frac{br}{2^j}$ rows of A above and including row br.

Step 3. Witnesses in A are found against the surviving vectors in sets $S_0, \ldots, S_{\log br}$ in parallel. Lemma 10 is crucial in performing this step efficiently. For set S_j, the property of parallelism of difference vectors and the restriction on the region containing the lower endpoints of witnesses against vectors in S_j enables this step to be accomplished in $O(\log\log m)$ time and $O(\frac{br}{2^j} \times l_{i+1})$ work. We defer the details of this step to the full paper. Clearly, the work done over all sets S_j is $O(|A|)$.

The New Block Structure. As stated earlier, witnesses are found against all non-period vectors of fringes of P' of successively increasing sizes in $O(\log\log m)$ stages. In Stage $i+1$, witnesses are sought against all non-period vectors of the l_{i+1}-fringe F_{i+1} of length less than $\frac{l_{i+1}}{16}$, given witnesses against all non-period vectors of the l_i-fringe F_i of length less than $\frac{l_i}{16}$, for appropriate l_i, l_{i+1}. There are two crucial steps in the computation in Stage $i+1$; each step uses $O(1)$ time and $O(|F_{i+1}|)$ work.

The first step computes witnesses in F_{i+1} against a given subset S of the Quad I period vectors of F_i of length less than $\frac{l_i}{16}$. We describe the case in which there are at least two non-parallel vectors, v_1 and v_2, in S. The other cases are handled similarly. We redefine the term defect as follows. A location $a \in F_{i+1}$ is called a *defect* if the character at a differs from the character at location c in the lattice cell bounded by the points $e, e+v_1, e+v_2, e+v_1+v_2$, where $c - a$ is a integer linear combination of v_1, v_2, and e is the top left corner of P'. Lemma 8 holds here too. Using Lemma 11, the above computation is accomplished in $O(1)$ time and $O(|F_{i+1}|)$ work.

Lemma 11. There are no defects in the inner half of F_i. If there are no defects in F_{i+1} then there are no witnesses in F_{i+1} against vectors in S. Otherwise, let α be the outermost defect in F_{i+1}. If α is in the upper or the left half of F_{i+1} then $\alpha - v$ is in F_{i+1} and is a non-defect; also $(\alpha - v, \alpha)$ is a witness against all $v \in S$. Otherwise, if α is in the lower or the right half of F_{i+1} then $\alpha + v$ is in F_{i+1} and is a non-defect; also $(\alpha, \alpha + v)$ is a witness against all $v \in S$.

The second step computes witnesses against the set of non-period vectors of F_{i+1} of length less than $\frac{l_{i+1}}{16}$, given witnesses against the set W of non-period vectors of F_{i+1} of length less than $\frac{l_i}{16}$. A vector v is said to *fall* in block Z if its head is in Z when its tail is at the top left corner of P'. The main problem here is to accomplish the following in $O(1)$ time and $O(|F_{i+1}|)$ work: given a set V of vectors which fall in some $\frac{l_i}{16} \times \frac{l_i}{16}$ subblock Z of F_{i+1}, eliminate enough vectors in V so that if $u, v \in V$ survive then $v - u$ is a period vector of F_{i+1}. Clearly, if at most one vector in V falls in each row of Z then this is easily done by duelling all pairs of vectors in V in parallel using the witnesses computed against vectors in W. By forcing these witnesses to be near the center of P, we ensure that the locations $a + v$ (see definition of duel) needed for these duels lie within P. If more than one vector falls in some row of Z then we show that after a preprocessing stage, a slight modification of this scheme works in the same time and work bounds. This preprocessing stage, which is performed before the current sequence of stages, is survived by vectors u, v only if either $v-u$ is not horizontal or $v-u$ is a period vector of P'; it takes $O(\log\log m)$ time and $O(|P'|)$ work.

We conclude with the following theorem.

Theorem 6. There exists a CRCW-PRAM algorithm which finds witnesses against all non-period vectors of an $m \times m$ pattern in $O(\log\log m)$ time using $O(m^2)$ work.

7. Conclusion

We obtain deterministic one- and two-dimensional pattern matching algorithms which are provably the best in both preprocessing and text search. We also show how to compute deterministic samples in constant time. We use it to obtain a simple probabilistic algorithm and a more sophisticated randomized algorithm for string matching with constant expected time.

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 translates into an optimal $O(\log m)$ time algorithm; 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 [16]. 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.)

Finally, is it possible to design a randomized algorithm for two-dimensional pattern matching with constant expected time? Perhaps one first needs to simplify our deterministic witness computation.

References

[1] A. Amir and G. Benson, Two-dimensional periodicity and its applications, *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, 1992, 440–452.

[2] A. Amir, G. Benson and M. Farach, Alphabet independent two dimensional matching, *Proc. 24th ACM Symp. Theory of Computing*, 1992, 59–68.

[3] A. Amir, G. Benson and M. Farach, Optimal parallel two dimensional pattern matching, *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 1993, 79–85.

[4] A. Amir and G.M. Landau, Fast parallel and serial multidimensional approximate array matching, *Theoret. Comput. Sci.* 81 (1991), 97–115.

[5] 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.

[6] T.J. Baker, A technique for extending rapid exact-match string matching to arrays of more than one dimension, *SIAM J. on Comput.* 7 (1978), 533–541.

[7] R.S. Bird, Two dimensional pattern matching, *Inform. Process. Lett.* 6 (1977), 168–170.

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

[9] D. Breslauer and Z. Galil, A lower bound for parallel string matching, *SIAM J. Comput.* 21 (1992), 856–862.

[10] B.S. Chlebus and L. Gąsieniec, Optimal pattern matching on meshes", Manuscript, 1993.

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

[12] Z. Galil, A constant-time optimal parallel string-matching algorithm, *Proc. 24th ACM Symp. Theory of Computing*, 1992, 69–76.

[13] Z. Galil and K. Park, Truly alphabet-independent two-dimensional pattern matching, *Proc. 33rd IEEE Symp. Found. Computer Science*, 1992, 247–256.

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

[15] R. Hariharan, An optimal parallel algorithm for computing leftmost string witnesses, Manuscript, 1992.

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

[17] Z. Kedem, G. Landau and K. Palem, Optimal parallel suffix-prefix matching algorithm and application, *Proc. 1st ACM Symp. on Parallel Algorithms and Architectures*, 1989, 388–398.

[18] M.G. Main and R.J. Lorentz, An $O(n\log n)$ algorithm for finding all repetitions in a string, *J. Algorithms* 5 (1984), 422–432.

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

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

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