A CRCW PRAM lower bound for problems on two dimensional arrays

Clive N. Galley *
Department of Computer Science
Kings College London
University of London

This paper is dedicated to the memory of Irene Galley

ABSTRACT. An array $A[i,j], 1 \leq i \leq n, 1 \leq j \leq n$ has a *period* $A[p,p]$ of dimension $p \times p$ if $A[i,j] = A[i+p, j+p]$ for $i,j = 1 \cdots n-p$. *The period* of $A[p,p]$ is the shortest such p. We study two dimensional pattern matching, and several other related problems all of which depend on finding the period of an array. In summary finding the period of an array in parallel using p processors for general alphabets has the following bound

$$\begin{cases} \Theta\left(\frac{n^2}{p}\right) & \text{if } p \leq \frac{n^2}{\log\log n}, n > 17^3 \quad (1.1) \\ \Theta(\log\log n) & \text{if } \frac{n^2}{\log\log n} < p < n^2, n > 17^3 \quad (1.2) \\ \Theta\left(\log\log_{\frac{2p}{n^2}} p\right) & \text{if } n^2 \leq p < n^2\log^2 n, n > 17^3 \quad (1.3) \\ \Theta\left(\log\log_{\frac{2p}{n^2}} p\right) & \text{if } n^2\log^2 n \leq p \leq n^4, n \text{ large enough} \quad (1.4) \end{cases}$$

1 Introduction

An *alphabet* denoted by, Σ, is a set of symbols. An unbounded or *general alphabet* is an alphabet where the set of symbols is variable. A *fixed alphabet* is an alphabet where the set of symbols is bounded by a constant. A *string* is a sequence of zero or more symbols from an alphabet Σ. A string x is of *length* n, where there are n symbols in x.

*The author was supported by the 91809000 Engineering and Physical Sciences Research Council Postgraduate Studentship. The author was also partially supported by a University of London Central Research Fund grant.

The string xy is a *concatenation* of two strings x and y. A string w is a *prefix* of x if $x = wu$ for $u \in \Sigma^*$, where Σ^* is the set of all strings over an alphabet Σ. Similarly a string u is a *suffix* of x if $x = wu$. *A period* u of the string x is a prefix of x constructed by concatenations of u. *The period* of the string x is the shortest such u. e.g. The period of *abaabaabaaba* is *aba*, and the period of *aabaaabaab* is *aaba*. If x has a period $u, x \neq u$ then x is *periodic*. A string $w_1 \dots w_m$ is a *substring* of $x = x_1 \dots x_n$ if $w_1 \dots w_m = x_i \dots x_{m-i-1}$ for some i.

A string x is said to be a *square* if there exists a substring u of x such that $x = uu$. e.g. *abab*, *abcabc*. A string is called *square free*, if it has no square as a substring. An array is *square free*, if the horizontal, and vertical strings making up that array also have no square as a substring.

We define the problem of *string pattern matching* as follows. Given a string p, of length m, called the pattern, and a string t, of length n, called the text, find all occurrences of the pattern in the text.

As yet, there is no established text book covering all the algorithms for the string pattern matching problem. For surveys the reader is referred to Aho's survey paper, see [A], which covers some efficient RAM algorithms. Also to Iliopoulos' survey, see [I] and [GS], for efficient parallel RAM algorithms. A history of string pattern matching is to be found in a leading paper on the problem by Knuth, Morris, and Pratt, see [KMP]. There also, is a linear time ($O(m+n)$) sequential RAM algorithm, which superseded an almost linear time algorithm by Karp, Miller, and Rosenberg, see [KMR]. This algorithm, however proved to be parallelizable, see [AILSV],[CR]. In another leading paper on string pattern matching, Boyer, and Moore gave a linear expected time algorithm, with $O(mn+|\Sigma|)$ worst case running time, in [BM].

Galil's [Ga] parallel algorithm for string matching was optimal, and required $O(\log n)$ time, for fixed alphabets. Vishkin in [V] introduced an algorithm with the same complexity, but applicable to general alphabets, using the *witness* idea.

Both of these were improved by Breslauer-Galil in [BG2], with an $O(\log \log n)$ time complexity for linear cost. In their analysis of the parallel computation of the string pattern matching problem in [BG], Breslauer-Galil had two phases of computation. Firstly the pattern is pre-processed with no access to the text required. (This pattern pre-processing phase is useful on its own, when one pattern is to be matched with several texts. Each text can be processed independently, in this case, thus reducing the overall cost, by pre-processing the single pattern only once.) Secondly the text is processed. The lower bound in [BG] refers to the pattern pre-processing. Hence our improvements also apply to this pattern pre-processing.

The *two dimensional pattern matching problem* is defined as follows.

Given a pattern P where $|P| = m \times m$ and a text T where $|T| = n \times n$ to which it is required to match P, where both are arrays of characters from an alphabet Σ, compute all occurrences of the pattern array in the text array.

In [G],[GI] we showed a lower bound on the problem of computing the period length of a string. This problem of computing the period length of a string is shown, in [ABG], to be equivalent to the problem of computing the initial palindromes of a string. Both problems have two-dimensional generalizations, which will be considered here. The following definitions are associated with these computational problems.

A string $S[0 \cdots k]$ is a *palindrome* if $S[i] = S[k - i]$ for $i = 0 \cdots k$. A string $S[1 \cdots n]$ is said to have an *initial palindrome* of length k if the prefix $S[0 \cdots k - 1]$ is a palindrome.

An array $A[i, j], 1 \leq i \leq n, 1 \leq j \leq n$ has a *period* $A[p, p]$ of dimension $p \times p$ if $A[i, j] = A[i + p, j + p]$ for $i, j = 1 \cdots n - p$. An array $A[k, k]$ is a *palindromic array* if $A[i, j] = A[k - i, k - j]$ for $i, j = 1 \cdots k$. An array is said to have an *initial palindrome* of dimension $k \times k$ if the prefixes in both directions $A[0 \cdots k - 1, 0 \cdots k - 1]$ are palindromes.

We define the problem of finding all repetitions in a string x as follows. Compute all the positions of squares within x. Main and Lorentz in [ML] provided an $O(n \log n)$ algorithm for finding all repetitions in a string, of length n, for general alphabets. Given that only comparisons of symbols are allowed by any algorithm, this complexity becomes $\Theta(n \log n)$. In [C] Crochemore found that all repetitions of a string, of length n, could be found, for fixed alphabets, also in $\Theta(n \log n)$. We define the problem of recognition of a square free string as follows. Given a string x, of length n, test if x contains any squares. If x does not, then it is said to be square free. For fixed alphabets, it was shown in [ML2] that recognition of a square free string, of length n, could be done in $O(n)$ time. For general alphabets in [ML2] an algorithm was provided for recognition of a square free string, of length n, in $O(n \log n)$ time.

Given a string, $S[1 \cdots n]$, we define the output of the computation of all periods of S, to be a Boolean array $P[1 \cdots n]$ such that $P[i] =$ true if and only if i is a period length of S. We further define the output of the computation of all initial palindromes of a string w, where w does not contain the symbol \$, to be the Boolean array P derived from the string $w\$w^R$. Here we denote as w^R, the string w reversed. An early result on initial palindromes can be found in [FP], where the equivalence of computation of palindromes, and periods was shown. The constant of the lower bound for the computation of all periods and initial palindromes in [ABG] is improved, by reducing the gap in the range of length of string, n, covered by that proof. This is achieved by a similar proof to that in [G],[GI].

The result is very similar to that in the improvement of the lower bound for pattern matching in parallel on the CRCW PRAM, (see [G],[GI]), except there are several new constant factors involved in the derivation of the bound.

In the following we refer to a function computed in the pattern processing step of the Knuth-Morris-Pratt algorithm, which is known as the *KMP failure function*, see [KMP]. This function represents an automaton containing separate branches, from a single root, for each substring of a string x with a common non-trivial prefix of x. We define the problem of *string prefix matching* to be as follows. Given a text string $T[1 \cdots n]$, and a pattern string $P[1 \cdots m]$, compute the longest prefix of P that occurs starting at any position of T.

Given the number of processors in the computation of the KMP failure function and string prefix matching of $O(m \log m / \log \log m)$ by Breslauer in [B], we can show a similar improvement to that shown here (in 2.13.1), for the lower bound of $\Omega(\log \log m)$ time in [B]. This computation of the KMP failure function, and string prefix matching by Breslauer in [B] requires the sub-computation of string pattern matching.

The existing lower bounds for periodicity related string problems as in [BG], and [ABG] left no gap in the parallel complexity of those problems, but there are possibilities resulting from generalizing the single dimensional case to two, or possibly more, dimensions. The first proof by Breslauer and Galil was ingenious in the use of induction, with invariants on any algorithm, and this proof convinced those working on parallel string matching at the time, of the fact that $O(\log \log m)$ pre-processing is the best possible.

Subsequent improvements to string matching algorithms concentrated on reaching constant time processing for $\Theta(\log \log m)$ pre-processing as in [Ga2], and [CC], and [GZ]. Some new open problems arose from the pattern matching proof in [BG], in that the range of length of text was restricted, for the constant of $1/4$. Also a condition of $n = 2m$ was required by their bound. With the new lower bounds in [ABG] only the range restriction remained relevant. For the proof in [ABG] to hold, the length of string required is vastly greater, than the already long string required in [BG].

In this paper we generalize the results obtained in [G],[GI] for one dimenssional pattern matching. Here we study the same problems in a two-dimensional setting. In particular we show, improved by a constant factor, lower bounds for two dimensional pattern matching. Moreover, we show a lower bound for recognition of square freeness, and computing all squares of an array. Similarly this bound applies to two dimensions.

The techniques used in this paper are similar to, or extensions of, the ones used in the previous paper, see [G],[GI]. There we showed it was possible to count multiples that forced a comparison in a way which allowed the

count to grow, depending on a variable k for k up to 3. This variable determined the number of partitions where we can form products of primes. To generalize the lower bound in [G],[GI] to two dimensional period lengths, $k = 3$ is enough. To further generalise this technique it would be necessary to show $k > 3$ forces a comparison for some function of k.

The results obtained are as follows :

(1) An improved constant factor, and smaller restriction on the size of array for which the proof holds, for the $\Omega(\log\log m)$ time, linear (in $O(m^2)$) work lower bound for two dimensional pattern matching. The constant is 0.26 for $m \geq 17^3 = 4913$.

(2) Lower bounds for finding all periods, and initial palindromes, the KMP failure function, and other problems related to finding the period of an array with the same complexity, to within a constant, as (1).

(3) Given an $n \times n$ array, a lower bound for recognition of square freeness, and computing all squares, of $\Omega(\log\log n)$ time for $O(n^2 \log^2 n)$ work.

In summary finding the period of an array in parallel using p processors for general alphabets has the following bound

$$\begin{cases} \Theta\left(\frac{n^2}{p}\right) & \text{if } p \leq \frac{n^2}{\log\log n}, n > 17^3 \quad (1.1) \\ \Theta(\log\log n) & \text{if } \frac{n^2}{\log\log n} < p < n^2, n > 17^3 \quad (1.2) \\ \Theta\left(\log\log_{\frac{2p}{n^2}} p\right) & \text{if } n^2 \leq p < n^2\log^2 n, n > 17^3 \quad (1.3) \\ \Theta\left(\log\log_{\frac{2p}{n^2}} p\right) & \text{if } n^2\log^2 n \leq p \leq n^4, n \text{ large enough} \quad (1.4) \end{cases}$$

(The algorithm in [CC] may be used in (1.2) directly, and in (1.3)-(1.4) with a redundant processors, and a in (1.1) with a slow down.)

In the next section, the proofs of the lower bounds for two-dimensional pattern matching for linear work are presented. These proofs also apply to finding all periods, and initial palindromes, and other problems on two dimensional arrays requiring only linear work to solve. In Section 3 we study the problem of recognition of square freeness, and computing all squares of two-dimensional arrays. In this section the proofs of the lower bounds from Section 2 are extended to $O(n^2 \log^2 n)$ work for two dimensional arrays.

2 Structure of the proof in two dimensions

The following definition replaces the relatively prime multiples used in [BG]. If either end of the vectors we call prime vectors, defined below, are projected on to either axis, the set of multiples generated is a subset of that

of relatively prime multiples from $n/2$ to n in [BG]. Hence it is possible to claim that the one dimensional proof is a special case of the two dimensional proof that follows.

Definition 2.1. *A vector* $(x_1, y_1) \rightarrow (x_2, y_2)$ *is a prime vector iff*

(i) $\{x_1, y_1\}, \{x_2, y_2\}$ *both force a comparison in one dimension (i.e. both sets satisfy the conditions for* p, q *in Lemma 2.2-2.4 below)*

(ii) $P \in \{x_1, y_1, x_2, y_2\}$ *is such that* $\forall\ P\ m/2 < P < m$ *where* m *is the length of one edge of the pattern array we are pre-processing.*

(iii) $P_1 \in \{x_1, y_1\}$ *is such that* $m/2 < P_1 < 3m/4$.

(iv) $P_2 \in \{x_2, y_2\}$ *is such that* $3m/4 < P_2 < m$. □

In the one dimensional case it was sufficient to denote the testing of equality, or non-equality, of two symbols, from an alphabet Σ, in a string, as a comparison. In the two-dimensional case, such a comparison is referred to as a *symbol comparison.* Given two symbols, from an alphabet, Σ, in an array, the testing for equality, or non-equality, of those symbols, is referred to as a symbol comparison. Moreover, the simultaneous comparison of the symbols at both ends of a vector, in an array, with the symbols at both ends of another vector, in the same array, is referred to as a *vector comparison.*

Consider the problem of finding the period length of an array, where *a period* u of the string x is a prefix of x constructed by concatenations of u, and *the period* of the string x is the shortest such u. In the two dimensional case, where an array $A[i,j], 1 \leq i \leq n, 1 \leq j \leq n$ has *a period* $A[p,p]$ of dimension $p \times p$ if $A[i,j] = A[i+p, j+p]$ for $i, j = 1 \ldots n-p$, and *the period* of the array is $A[p,p]$ for the shortest such p.

We use the following strategy, which is a generalization of that used in one dimension by Breslauer and Galil. Given an adversary, he could answer

$$\Omega(\log \log m)$$

iterations of vector comparisons in such a way that there is still a choice of *fixing* the input array A in two different ways. One way is that the output array has a period of length smaller than $\frac{m}{4}$. The other way is that the output array does not have such a period. This implies that any algorithm using less iterations than this will be erroneous.

We continue with some definitions used in [BG]. An integer k is a possible period length of A if the array A can be fixed consistently using the output from previous vector comparisons in such a way that k is a period length of A. For k to be a period length it is necessary that each residue equivalence class modulo k is fixed to the same symbol. Thus if $l_1 = j_1 \ mod\ k$, and

$l_2 = j_2 \ mod \ k$ then $A[l_1] = A[j_1]$, and $A[l_2] = A[j_2]$ where $A[i_1]$ is the x co-ordinate, and $A[i_2]$ is the y co-ordinate, of elements of a two dimensional array A.

At the start of iteration i Breslauer-Galil's adversary will maintain an integer, k_i, which is a possible period length. During iteration i this adversary answers vector comparisons such that some k_{i+1} is also a possible period length, and some symbols of A are fixed. Moreover, given a pattern array bounded by m^2, the following Lemma from the Breslauer-Galil proof is critical to what follows.

Lemma 2.2. [BG] *If* $p, q \geq \sqrt{\frac{m}{k_i}}$ *are relatively prime, then a symbol comparison is forced by at most one of the multiples* pk_i *and* qk_i. □

Lemma 2.3. *Let* $p = p_1 p_2, q = q_1 q_2$, *with*

$$\sqrt{\frac{m}{2k_i}} \leq p_j, q_j \leq \sqrt{\frac{m}{k_i}},$$

where $p_j, q_j, j = 1, 2$ *are prime numbers. A symbol comparison is forced by at most one of* pk_i *and* qk_i.

Proof: The lemma holds when $gcd(p, q) = 1$, see [BG].

Assume w.l.o.g. that $gcd(p, q) = p_1 = q_1$, $l \equiv k \bmod pk_i$ and $l \equiv k \bmod qk_i$ for $1 \leq l, k \leq m$. Then we have $l \equiv k \bmod p_2 q_2 p_1 k_i$ and $q_2 > 2$, which implies $p_2 q_2 p_1 k_i > m$; this implies $l = k$, a contradiction. □

Lemma 2.4. *Let* $p = p_1 p_2 p_3, q = q_1 q_2 q_3$, *with*

$$\left(\frac{m}{2k_i}\right)^{1/3} \leq p_j, q_j \leq \left(\frac{m}{k_i}\right)^{1/3}$$

where $p_j, q_j, i = 1, 2, 3$ *are prime numbers. A symbol comparison is forced by at most one of* pk_i *and* qk_i.

Proof: The lemma holds when the $gcd(p, q) = 1$, see [BG].

Assume $gcd(p, q) > 1$, $l \equiv k \bmod pk_i$ and $l \equiv k \bmod qk_i$ for $1 \leq l, k \leq m$.

If the $gcd(p, q)$ is just one prime number and w.l.o.g $gcd(p, q) = p_1 = q_1$, then we have $l \equiv k \bmod p_2 p_3 q_2 q_3 p_1 k_i$ and

$$p_2 p_3 q_2 q_3 p_1 k_i > m$$

this implies $l = k$, a contradiction.

If the $gcd(p, q)$ is a product of two primes and w.l.o.g let $gcd(p, q) = p_1 p_2 = q_1 q_2$ then we have $l \equiv k \bmod p_3 q_3 p_1 p_2 k_i$ and

$$p_3 q_3 p_1 p_2 k_i > m$$

this implies $l = k$, a contradiction. □

In the one dimensional case, as above, k_i is a candidate for the period length of a string of length n. However in the two dimensional case which follows, k_i is a candidate for the period length of an $n \times n$ array.

We also need the following three theorems from [GI] which improve the bound by counting comparison forcing multiples.

Theorem 2.5. [RS] *For $n \geq 17$ the number of prime integers between 1 and n denoted by $\pi(n)$ satisfies the following*

$$\frac{n}{\ln n} \leq \pi(n) \leq \frac{5}{4}\frac{n}{\ln n}$$

□

Corollary 2.6. *There are at least*

$$\rho_k = \left(\frac{1}{4}\frac{n^{\frac{1}{k}}}{\ln n^{\frac{1}{k}}}\right)$$

distinct prime integers in the range $[(n/2)^{\frac{1}{k}}, n^{\frac{1}{k}}]$ for $k > 1$.

Proof: Let $r = 1/k$ then from Theorem 2.5 we have

$$\pi(n^r) \geq \frac{n^r}{\ln n^r} \text{ and } \pi\left(\left(\frac{n}{2}\right)^r\right) \leq \frac{5}{4}\frac{\left(\frac{n}{2}\right)^r}{\ln\left(\frac{n}{2}\right)^r}$$

which implies that

$$\begin{aligned}\pi(n^r) - \pi\left(\left(\frac{n}{2}\right)^r\right) &\geq n^r\left(\frac{1}{r\ln n} - \frac{5}{2^r 4r\ln\left(\frac{n}{2}\right)}\right)\\ &= \frac{n^r}{r}\left(\frac{1}{\ln n} - \frac{5}{2^r 4(\ln n - 1)}\right)\\ &> \frac{n^r}{r}\left(\frac{1}{4\ln n}\right)\end{aligned}$$

□

Lemma 2.7. *There are at least*

$$\prod_{k=2}^{3}\binom{\rho_k}{k} > \frac{n^2}{15\log^5 n}$$

pairs (p, q) of distinct integers, with $p = p_1 p_2 ... p_k$, $q = q_1 q_2 ... q_k$ and, p_j, q_j are prime numbers in the range $[(n/2)^{\frac{1}{k}}, n^{\frac{1}{k}}]$, where

$$\rho_k = \frac{1}{4}\frac{n^{\frac{1}{k}}}{\ln(n^{\frac{1}{k}})}$$

Proof: From Corollary 2.6 there are ρ_k primes in the range $[(n/2)^{\frac{1}{k}}, n^{\frac{1}{k}}]$. One can choose k primes, $p_i, q_i \in [(n/2)^{\frac{1}{k}}, n^{\frac{1}{k}}], 1 \leq i \leq k$, then $p = p_1 p_2 .. p_k$, and $q = q_1 q_2 ... q_k$ seem to satisfy the conditions required to force a comparison. Unfortunately only p up to $p = p_1 p_2 p_3$ can be shown to do this (see Lemmata 2.3 and 2.4). Now considering $k = 2$ and $k = 3$, one can see that there are at least

$$\binom{\rho_2}{2}\binom{\rho_3}{3} \tag{2.7.1}$$

such pairs. By (2.7.1) it follows that

$$\begin{aligned}\binom{\rho_2}{2}\binom{\rho_3}{3} &= \frac{(\rho_2 - 1)\rho_2\rho_3(\rho_3 - 1)(\rho_3 - 2)}{12} \\ &\geq \frac{1}{12}(\rho_3^3\rho_2^2 - 3\rho_3^2\rho_2^2 + 2\rho_3\rho_2^2 - \rho_2\rho_3^3 + 3\rho_3^2\rho_2 - 2\rho_2\rho_3) \\ &\geq \frac{n^2}{15\log^5 n}\end{aligned}$$

□

In the two dimensional case we apply the same analysis as in [BG], but here p and q are prime vectors, and the lower bound, which follows in a similar fashion, requires developing a calculation based on the size of the set of these prime vectors.

Lemma 2.8. *Given an $m \times m$ array, and two arbitrary vectors $\vec{v} = (l_1, l_2) \rightarrow (l_3, l_4)$, and $\vec{u} = (j_1, j_2) \rightarrow (j_3, j_4)$, $1 \leq l_i \leq m$, and $1 \leq j_i \leq m$, and a prime vector, (see Definition 2.1), $\vec{w} = (x_1, y_1) \rightarrow (x_2, y_2)$, then a vector comparison $\vec{v} = \vec{u}$ is forced by the prime vector $\vec{w}$.*

Proof: Assume $j_1 \equiv l_1 \bmod (x_2)$ and $j_2 \equiv l_2 \bmod (y_2)$.

By two counts of Lemmata 2.2-2.4, a symbol comparison at (j_1, j_2) to (l_1, l_2) is forced.

Assume $j_3 \equiv l_3 \bmod (x_1)$ and $j_4 \equiv l_4 \bmod (y_1)$.

Again by two applications of Lemmata 2.2-2.4, a symbol comparison at (j_3, j_4) to (l_3, l_4) is forced.

As both ends of the prime vector, $\vec{w}$, force a symbol comparison at both ends of $\vec{v}$, and $\vec{u}$ a vector comparison is forced. In other words we force the symbol at (l_1, l_2) to be compared with the symbol at (j_1, j_2), and we force the symbol at (l_3, l_4) to be compared with the symbol at (j_3, j_4). That is

$$(j_1, j_2) \rightarrow (j_3, j_4)$$

is forced by

$$(l_1, l_2) \rightarrow (l_3, l_4)$$

The lemma follows □

Lemma 2.9. *There are at least*

$$\frac{1}{64}\left(\frac{c_1 n^2}{\log^5 n}\right)^4$$

prime vectors $\vec{w}$ *in the range* $n/2 < \vec{w} < n$ *for both co-ordinates, where*

$$c_1 = \frac{1}{15}$$

Proof: From Lemma 2.7 there are at least

$$\frac{c_1 n^2}{\log^5 n}$$

pairs (p, q) of distinct integers such that $n/2 < (p, q) < n$ so there are

$$\left(\frac{c_1 n^2}{\log^5 n}\right)^2 \tag{2.9.1}$$

of these pairs in both co-ordinates. To form the vectors we take

$$(x_1, y_1) \in \{m/2 \cdots 3m/4\} \text{ and } (x_2, y_2) \in \{3m/4 \cdots m\}$$

each giving

$$\frac{1}{8}\left(\frac{c_1 n^2}{\log^5 n}\right)$$

thus there are

$$\frac{1}{64}\left(\frac{c_1 n^2}{\log^5 n}\right)^2 \tag{2.9.2}$$

on consideration of both ends of each prime vector. Considering the product of 2.9.1 and 2.9.2 the lemma follows. □

The proof continues in much the same way as in [BG], but the invariants that are involved differ in the following ways:

- k_i is a prime vector multiple as in Lemma 2.8,
- modular residues are in both co-ordinates,
- K_i is the square of the function in [BG], that is

$$K_i = \left(m^{1-4^{-(i-1)}}\right)^2$$

Thus an adversary to any algorithm maintains the following invariants at iteration i:

(1) $\frac{1}{2}K_i \leq k_i \leq K_i$

(2) Given an $m \times m$ array, then for each symbol fixed to $(l_1, l_2) \rightarrow (l_3, l_4)$, for every $j_1 \equiv l_1 \bmod (x_2)$, and for every $j_2 \equiv l_2 \bmod (y_2)$, and for every $j_3 \equiv l_3 \bmod (x_1)$, and for every $j_4 \equiv l_4 \bmod (y_1)$;

$$(j_1, j_2) \rightarrow (j_3, j_4)$$

is fixed to the same symbol, where $x_1, x_2, y_1, and\ y_2$ are components of a prime vector.

(3) If a symbol comparison is equal then both symbols compared were fixed to the same symbol.

(4) If a symbol comparison is unequal then

(a) it is between different residues modulo $x_1, x_2, y_1, and\ y_2$

(b) if the symbols were fixed to values then those values are different

(5) The number of fixed symbols satisfies $f_i \leq K_i$.

A candidate denotes a possible new period length after an iteration of any algorithm. Next to be considered are the candidates for k_{i+1}, which are prime vector multiples of k_i, which also, as a check, satisfy the condition of Lemmata 2.2-2.4 in both co-ordinates. As mentioned earlier this enables us to claim that the one dimensional proof is a special case of our two dimensional version.

Lemma 2.10. *There are the following candidates for k_{i+1} which are prime vector multiples of k_i, and satisfy the invariant held by the adversary that $\frac{1}{2}K_{i+1} < k_{i+1} < K_{i+1}$, and the pair (k_i, k_{i+1}) satisfy the conditions of Lemmata 2.3-2.4 :*

$$\frac{1}{64}\left(\frac{c_1 K_{i+1}^2}{\log^5 m K_i^2}\right)^4, \quad K_i = \left(m^{1-4^{-(i-1)}}\right)^2$$

(Where we define a vector multiple as the conjunction of the co-ordinate scalar multiples.)

Proof: These candidates are of the form $\vec{r}k_i$ for a prime vector $\vec{r}$. The count of these follows from Lemma 2.9. □

Lemma 2.11. *Each such candidate satisfies the condition of Lemmata 2.2-2.4 for both co-ordinates.*

Proof: As $x_1 y_1 k_i \geq K_{i+1}/2$, $x_2 y_2 k_i \geq K_{i+1}$, and $k_i \leq K_i$ then

$$(x_1 y_1)^2 \geq \frac{1}{k_i}\frac{K_{i+1}^2}{4K_i} = \frac{1}{k_i}\frac{\left(m^{1-4^{-i}}\right)^4}{4\left(m^{1-4^{-(i-1)}}\right)^2} = \frac{1}{k_i}\frac{1}{4}m^2 m^{4 \cdot 4^{-i}} \geq \frac{m^2}{k_i}$$

$$(x_2y_2)^2 \geq \frac{1}{k_i}\frac{K_{i+1}^2}{4K_i} = \frac{1}{k_i}\frac{\left(m^{1-4^{-i}}\right)^4}{4\left(m^{1-4^{-(i-1)}}\right)^2} = \frac{1}{k_i}\frac{1}{4}m^2m^{4\cdot 4^{-i}} \geq \frac{m^2}{k_i}$$

□

Lemma 2.12. *There exists a candidate for k_{i+1} in the range $\frac{1}{2}K_{i+1}\cdots K_{i+1}$ that forces at most*

$$64m^4\left(\frac{K_i^2\log^5 m}{c_1K_{i+1}^2}\right)^4$$

vector comparisons. (Recall a vector comparison is defined as the simultaneous comparison of the symbols to be found at the co-ordinates of both ends of the vector.)

Proof: By Lemma 2.10 there are at least

$$\frac{1}{64}\left(\frac{c_1K_{i+1}^2}{\log^5 mK_i^2}\right)^4$$

such candidates that are prime vector multiples of k_i. By Lemma 2.8 each of the m^4 vector comparisons (m^2 for each end of each vector) is forced by at most one of them. □

Theorem 2.13. *Any comparison based algorithm for finding the period length, smaller than $(m/2)^2 = m^2/4$, of an array of size $m \times m$ using m^2 symbol comparisons in each iteration requires $0 \cdot 26(\log\log m)$ iterations.*

Proof: The proof is similar to that in [BG] except in the number of fixed elements which follows, and in that the invariants hold at iteration $i+1$ as follows :-

The basis for induction follows from the Breslauer-Galil proof, as there are no changes from their proof at that stage. By Lemma 2.12 k_{i+1} exists and it forces at most

$$64m^4\left(\frac{K_i^2\log^5 m}{c_1K_{i+1}^2}\right)^4$$

vector comparisons.

These vector comparisons are equal iff $x_1, x_2, y_1,$ *and* y_2 modulo k_{i+1} fix the residue class modulo k_{i+1} to the same symbol. All others are unequal. k_{i+1} is a prime vector multiple of k_i as in Lemma 2.8, so each residue class modulo k_i has four scalar residue classes modulo $x_2, x_1, y_2,$ *and* y_1 which split into

$$k_{i+1}/x_2, k_{i+1}/x_1, k_{i+1}/y_2, k_{i+1}/y_1$$

residue classes modulo k_{i+1}. If two indices are in different (respectively the same) residue classes modulo k_i, i.e. all four scalar residue classes,

then they are also in different (respectively the same) all four scalar residue classes modulo k_{i+1}.

The invariants can now be itemized

(1) By Lemma 2.10, $\frac{1}{2}K_{i+1} < k_{i+1} < K_{i+1}$, as the prime vector multiples are in the required range.

(2) By the argument above, each symbol is still fixed as before.

(3) Likewise, by the argument above, for equal symbols.

(4) (i) Residue classes of any of the four points may differ to produce an unequal answer.

(ii) Any previous differing class is maintained, by consistency of the calculation. Two relative prime multiples composed will not produce a common factor.

(5) By induction $f_{i+1} \leq k_{i+1}$. Here we must consider the induction in each co-ordinate in order to validate the proof. Where f^x_{i+1} is the x co-ordinate of f_{i+1}, and likewise f^y_{i+1} is the y co-ordinate of f_{i+1}, then let the prime vector $x_1 \to x_2$ fix f^x_{i+1}, and let the prime vector $y_1 \to y_2$ fix f^y_{i+1}, each having two scalar residue classes. Given that we show in the following proof that $f_{i+1} \leq k_{i+1} \Rightarrow f^{x,\text{ and } y}_{i+1} \leq k^{x,\text{ and } y}_{i+1}$, and both sets of co-ordinate vectors are of the same size, this implies that $f^x_{i+1} \leq k^x_{i+1}$ and $f^y_{i+1} \leq k^y_{i+1}$.

Each residue class modulo k_{i+1} has, at most,

$$(\lceil m \rceil)^4/4k_{i+1} < (2m)^4/k_{i+1} = 16m^4/k_{i+1} \leq 32m^4/K_{i+1}$$

elements and

$$f_{i+1} \leq K_i \left[1 + \frac{32 \cdot 64 m^4}{K_{i+1}} m^4 \left(\frac{K_i^2 \log^5 m}{c_1 K_{i+1}^2}\right)^4\right]$$

$$\leq K_i \left[1 + \frac{32 \cdot 64}{c_1^4} \frac{K_i^8}{K_{i+1}^9} m^8 \log^{20} m\right] \leq K_{i+1}$$

$$1 + \frac{32 \cdot 64}{c_1^4} m^{-46 \cdot 4^{-i}} m^6 \log^{20} m \leq \frac{K_{i+1}}{K_i}$$

$$(1+B)^{(1/52)} = \left(1 + m^6 \log^{20} m \frac{32 \cdot 64}{c_1^4}\right)^{(1/52)} \leq m^{4^{-i}}$$

and finally

$$\log\log((1+B)^{1/52}) \leq (1 - 0 \cdot 48) \log\log m$$

$$\log((1+B)^{1/52}) \leq 4^{0.26 \log\log m} \log m \qquad (2.13.1)$$

$$(1+B)^{1/52} \leq m^{4^{-0.26(\log\log m)}} \leq m^{4^{-i}}$$

so after $0 \cdot 26 \log\log m$ iterations

$$f_{i+1}, k_{i+1} \leq \left(m^{1-4^{-0.26\log\log m}}\right)^2 \leq \frac{m^2}{4}$$

and the theorem follows.

□

Note: The inequality in 2.13.1 is true for $m \geq 4913 = 17^3$.

We can also improve the lower bound for finding all periods and initial palindromes of an string, of length n, in parallel, from [ABG], with similar results to Theorem 2.13. There is a factor of four difference in the approximation to the linear work $\Omega(\log\log n)$ time lower bound for the parallel computation of all periods, and initial palindromes of a string of length n ([ABG]), from that in the string matching lower bound, ([BG]). In other words

$$\log\log(1+64\log n) \textit{ replaces } \log\log(1+16\log m)$$

(Here n, and m are equivalent for the purposes of comparison.)

3 Ramifications of the structure

Following the previous proofs we now present our improvements to Apostolico, Breslauer, and Galil's lower bound for recognition of square freeness, and hence for computing all squares, and its generalization to two dimensions. We begin with a reminder of some definitions.

A string x is said to be a *square* if there exists a substring u of x such that $x = uu$. e.g. *abab*, *abcabc*. A string is called *square-free* if it has no square as a substring. An array has no square, if the horizontal and vertical strings making up that array also have no square as a substring. Similarly to compute all squares in an array, requires the computation of all squares in the horizontal, and vertical strings which make up that array.

The best possible sequential algorithm for computing all squares in a string by Main and Lorentz, see [ML], runs in $O(n)$ time for fixed alphabets. For general alphabets $O(n \log n)$ time is required. Hence any optimal parallelization, for general alphabets, has an $O(n \log n)$ cost, as in [ABG]. In [ABG] it is shown that the string generated by the adversary in [BG] has a period smaller than half of its length if and only if it has a square. Hence, Apostolico et al show, the time complexity of $\Omega(\log\log n)$ is required for $O(n \log n)$ cost.

A summary of the changes to the proof from those already provided are given here. We are using essentially the same techniques, with some

substitutions for the extra $O(\log n)$ factor used in the case of recognition of square freeness, and computing all squares. The results are unchanged even with the extra $O(\log n)$ processors required in this case, and the lower bounds for computing the period length are still strong enough to cover an extra $O(\log n)$ factor to the number of processors for each dimension, which is needed in recognition of square freeness, and computing all squares. The first change to the previous proof occurs at Lemma 2.12, which is now as follows.

Lemma 3.1. *There exists a candidate that forces at most*

$$64n^4 \log^4 n \left(\frac{K_i^2 \log^5 n}{c_1 K_{i+1}^2} \right)^4$$

vector comparisons.

Proof: By Lemma 2.8 and Lemma 2.10, as earlier, except there now are $n^4 \log^4 n$ vector comparisons. □

Consider the one dimensional case. In [ABG] it is shown that the lower bound in [BG] extends to $O(n \log n)$ comparisons, and their proof at this stage is such that:-

$$\log\log(1 + 16\log^3 n) < \frac{1}{2}\log\log n$$

which only holds when $n > 10^{360}$. Here we improve this, by counting $n \log n$ elements in each residue equivalence class modulo k_{i+1}, before the fixed elements are bounded, and substituting $n \log n$ comparisons in the one dimensional case of Lemma 2.12, so the function at this point is

$$\left(\frac{4\log^7 n}{c_1} \right)^{1/3} < \left(n^{4^{-\frac{1}{4}\log\log n}} \right)^c < n^{4^{-\frac{1}{4}\log\log n}} \qquad (3.1.1)$$

Below we include the lower bound approximations for pattern matching from [BG] in (3.1.2), and from [GI] in (3.1.3) for completeness.

$$\log\log(1 + 16\log m) \leq \frac{1}{2}\log\log m \qquad (3.1.2.)$$

$$\log\log \left(\frac{4\log^5 m}{c_1} \right)^{1/3} \leq \frac{1}{2}\log\log m \qquad (3.1.3)$$

The two-dimensional proof has similar improvements. Consider the problem of finding the period length of an array. The array we consider is bounded by n^2, and we use $n^2 \log^2 n$ symbol comparisons in each iteration. The lower bound, and its improvement, holds for testing whether such an array has a period of length smaller than $\frac{n}{4}$.

We use the same strategy as before, a generalization of that used in one dimension by Breslauer and Galil. Given an adversary, he could answer

$$\Omega(\log\log n)$$

iterations of symbol comparisons in such a way that there is still a choice of *fixing* the input array A in two different ways. One way is that the output array has a period of length smaller than $\frac{n}{4}$. The other way is that the output array does not have such a period. This implies that any algorithm using less iterations than this will be erroneous, even with the extra symbol comparisons in each iteration.

We repeat some definitions used in [BG]. An integer k is a possible period length of A if the array A can be fixed consistently using the output from previous vector comparisons in such a way that k is a period length of A. For k to be a period length it is necessary that each residue equivalence class modulo k is fixed to the same symbol. Thus if $l_1 = j_1 \ mod \ k$, and $l_2 = j_2 \ mod \ k$ then $A[l_1] = A[j_1]$, and $A[l_2] = A[j_2]$ where $A[i_1]$ is the x co-ordinate, and $A[i_2]$ is the y co-ordinate, of elements of a two dimensional array A.

At the start of iteration i Breslauer-Galil's adversary will maintain an integer, k_i, which is a possible period length. During iteration i this adversary answers vector comparisons such that some k_{i+1} is also a possible period length, and some symbols of A are fixed.

The two-dimensional case includes $n^2\log^2 n$ symbol comparisons, and $n^2\log^2 n$ elements in each residue equivalence class modulo k_{i+1}, before the fixed elements are bounded.

Given a constant of (1/6) the details of the calculations for a 1024×1024 array are as follows: Here we have

$$i = \frac{1}{6}\log\log n$$

and the figures obtained for $n = 1024$ in relation 2.13.1 (with an extra factor of $\log^{8/52} n$) were $21.29 < 24.96$. We have therefore the following theorem.

Theorem 3.2. *Any comparison based algorithm for finding the period length, smaller than* $(n/2)^2 = n^2/4$, *of an* $n \times n$ *array using* $O(n^2\log^2 n)$ *symbol comparisons in each iteration requires*

$$\frac{1}{6}(\log\log n)$$

iterations. □

At the time of writing it was an open problem whether there is an algorithm which matches this complexity in Theorem 3.2, for the computation

of all squares, or recognition of square freeness. The result in one dimension in [ABG] had yet to be generalized to many dimensions on the PRAM.

A (finite) *Fibonacci string*, F_n is defined as follows.

$$F_0 = b,\ F_1 = a,\ F_n = F_{n-1}F_{n-2}\ \textit{for every integer}\ n \geq 2$$

A *primitive square* is a square, which itself is not repetitive. Theorem 3.2 is not surprising when we consider that there are

$$\Omega(n^2 \log^2 n)$$

primitive squares in an array of single shifted Fibonacci strings. The array consists of Fibonacci strings in both directions, see Figure 1. For details of the one dimensional case of this, where it is shown there are $\Omega(n \log n)$ squares in the Fibonacci string, see [C].

```
a b a b a a b a a b a
  a b a b a a b a a b a
    a b a b a a b a a b a
      a b a b a a b a a b a
        a b a b a a b a a b a
          a b a b a a b a a b a
            a b a b a a b a a b a
```

Figure 1. A Fibonacci array

References

[A] A. Aho, Pattern matching in strings, In R. Book ed. *Formal language theory perspectives and open problems*, Academic press, (1980), 325–347.

[ABG] A. Apostolico, D. Breslauer and Z. Galil, Optimal parallel algorithms for periods, palindromes, and squares, *Proc. ICALP* (1992), 296–307.

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

[B] D. Breslauer, Fast parallel string prefix matching, CWI report CS-R9239 (1992).

[BG] D. Breslauer and Z. Galil, A lower bound for parallel string matching, *SIAM Journal on Computing* **21** (1992), 856–862.

[BG2] D. Breslauer and Z. Galil, An optimal $O(\log \log n)$ parallel string matching algorithm, *SIAM Journal on Computing* **19** (1990), 1051–1058.

[BM] R. Boyer and J. Moore, A fast string searching algorithm, *Comm. ACM* **20:10** (1977), 262–272.

[C] M. Crochemore, An optimal algorithm for computing the repetitions in a word, *Information processing letters* **12** (1981), 244–250.

[CC] R. Cole, M. Crochemore, Z. Galil, L. Gasieniel, R. Hariharan, S. Muthukrishnan, K. Park and W. Rytter, Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions, *Proc. IEEE FOCS* (1993), 248–258.

[CR] M. Crochemore and W. Rytter, Usefulness of the Karp-Miller- Rosenberg algorithm in parallel computations on strings and arrays, *Theoretical Computer Science* **88** (1991), 59–82.

[FP] M. Fischer and M. Paterson, String matching and other products, *SIAM-AMS proceedings* **7** (1974), 113–125.

[Ga] Z. Galil, Optimal parallel algorithms for string matching, *Information and Control* **67** (1985), 69–76.

[Ga2] Z. Galil, A constant-time optimal parallel string-matching algorithm, *Proc. ACM STOC* (1992), 69–76.

[G] C. Galley, On the lower bound for parallel string matching, *Intern. J. Computer Maths* **53** (1994), 129–133.

[GI] C. Galley and C. Iliopoulos, On the lower bound for parallel string matching, submitted to *Intern. J. Computer Maths* (1996).

[GS] A. Gibbons, P. Spirakis, (Eds), *Lectures on parallel computation*, Cambridge University Press, (1993), 74.

[GZ] T. Goldberg and U. Zwick, Faster parallel string matching via larger deterministic samples, *Journal of Algorithms* **16** (1994), 295–308.

[I] C. Iliopoulos, Parallel algorithms for string pattern matching, *King's College TR 91/16*, (1991).

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

[KMR] R. Karp, R. Miller and A. Rosenberg, Rapid identification of repeated patterns in strings, arrays, and trees, *Proc. ACM STOC* (1972), 125–136.

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

[ML2] G. Main and R. Lorentz, Linear time recognition of square free strings, In A. Apostolico, Z. Galil editors, *Combinatorial algorithms on words*, Springer Verlag, (1985), 271–278.

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

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