Optimal Parallel Two Dimensional Pattern Matching

Amihood Amir* Gary Benson† Martin Farach‡
Georgia Tech U. of Southern California DIMACS

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; (404) 853-0083; amir@cc.gatech.edu; Partially supported by NSF grant IRI-90-13055.

†Department of Mathematics, University of Southern California, DRB 155, 1024 W. 36th Pl., Los Angeles, CA 90089-1113; (213) 740-2404; gbenson@hto-03.usc.edu; Partially supported by NSF grant IRI-90-13055.

‡DIMACS, Box 1179, Rutgers University, Piscataway, NJ 08855; (908) 932-5928; farach@dimacs.rutgers.edu; Supported by DIMACS under NSF contract STC-88-09648.

Proposed running head: Parallel Two Dimensional Matching

Address correspondence to:
Gary Benson
Department of Mathematics
University of Southern California
DRB 155
1024 W. 36th Pl.
Los Angeles, CA 90089-1113
(213) 740-2404
gbenson@hto-03.usc.edu

Abstract

We present a parallel algorithm for two dimensional matching. This algorithm is optimal in two ways. First, the total number of operations on the text is linear. Second, the algorithm takes time $O(\log m)$ on a CREW PRAM, thus matching the lower bound for string matching on a PRAM without concurrent writes. On a CRCW, the algorithm runs in time $O(\log \log m)$. Finding such an algorithm was a problem posed in 1985 and has been open since.

Key words: Analysis of algorithms, multidimensional matching, period, parallel algorithms, string

1 Introduction

In this paper, we present what is, to our knowledge, the first efficient parallel algorithm for two dimensional pattern matching over a general alphabet. Our text processing phase, which has been traditionally emphasized in pattern matching algorithms (see e.g. [Vis91, Gal92]) is optimal in that its work is linear and its running time is optimal, i.e. it matches the lower bound for CREW PRAMs. On a CRCW PRAM, our algorithm runs optimally in $O(\log\log m)$ time. Furthermore, it makes no assumptions on the character alphabet. The design of such an algorithm was posed as an open problem by Vishkin [Vis85] as early as 1985. Note that the preprocessing work is not optimal in this algorithm. It remains an open problem to have an algorithm that is optimal in both the preprocessing and the text processing.

Before describing our algorithm we present some background on sequential and parallel string matching.

The classical *string matching problem* has as its input a *text* string T of length n and a *pattern* string P of length m. The elements in the text and pattern are taken from an alphabet set Σ. The output is all text locations i where there is a character-by-character match with the pattern, i.e. $T[i+j-1] = P[j],\ j = 1, ..., m$.

String matching is one of the most widely studied problems in computer science [Gal85] and has many linear time solutions, starting with Knuth, Morris and Pratt [KMP77] and Boyer and Moore[BM77]. Parallel algorithms for string matching have also been extensively studied. After a long series of papers by Vishkin, Galil and others, Galil gave a constant time optimal algorithm

string matching on the CRCW [Gal92]. Breslauer and Galil [BG90] gave an $\Omega(\log\log m)$ time lower bound for CRCW string which Galil avoided in his $O(1)$ algorithm by ignoring preprocessing time. An $\Omega(\log m)$ time lower bound on the CREW PRAM follows from the lower bound for computing the `OR` of m bits [CDR86].

In recent years there has been growing interest in multidimensional pattern matching, largely motivated by problems in low-level image processing [RK82]. Various algorithms exist for the *exact two dimensional matching* problem. The exact two dimensional matching problem is defined similarly to the string matching problem but the text and pattern are rectangular matrices rather than strings. For simplicity's sake we assume that T is an $n \times n$ matrix and P is an $m \times m$ matrix, although all the stated results apply to rectangular matrices as well.

Baker [Bak78] and, independently, Bird [Bir77] used the Aho and Corasick [AC75] dictionary matching algorithm to obtain a $O(n^2 \log|\Sigma|)$ algorithm for the exact two dimensional matching problem. In [ABF92], we showed a two dimensional matching algorithm which was linear in the text size. In [GP92], the preprocessing for our algorithm was also improved to linear, thus matching the bounds for one dimensional string matching. The first work optimal parallel algorithm for two dimensional matching was given by Kedem, Landau and Palem [KLP89]. Their algorithm runs in time $O(\log m)$ on a CRCW PRAM. However, this algorithm assumes a fixed alphabet and requires space quadratic in the input size.

Our contribution is to give an work optimal algorithm that runs in $O(\log m)$ time on a CREW PRAM which works for general alphabets using linear space. The CRCW version of this algorithm

runs optimally in $O(\log\log m)$ time. The pattern preprocessing runs in $O(\log m)$ time with $O(m^2)$ CRCW processors, or in $O(\log^2 m)$ time with $O(m^2/\log m)$ CREW processors. Recently, [MR92] and [CGR92] have extended Galil's algorithm to two dimensions to produce an optimal constant time CRCW algorithm for two dimensions at the cost of some extra preprocessing.

Our approach has some similarities with the fast parallel string matching algorithms. Most of those algorithms use some notion of a *witness*, an idea which was introduced in [Vis85]. We also make use of witnesses, though in a different way. The string algorithms rely on that fact that periodic strings can be broken down into aperiodic substrings. The algorithm can then proceed by finding occurences of the smaller aperiodic string. We take a different approach. Amir and Benson showed in [AB92] that periodicity in two dimensions is much more complicated that in strings. In particular, there are four classes of periodicity in rectangular arrays. We achieve our efficient algorithm by dividing the four classes into two groups and showing how each group is handled.

The paper is constructed as follows. In section 2, we give a brief overview of two dimensional periodicity. In section 3, we give an outline of the algorithm. In section 4, we describe the pattern preprocessing. In section 5, we describe the first phase of the text processing in which we reduce the number of potential locations in which the pattern may occur. Finally, in section 6 we describe how to verify which of the surviving candidates represents an actual occurence of the pattern in the text.

2 Preliminaries

In [AB92], periodicity in two-dimensional arrays is defined. The pattern is divided into four quadrants. Any location in a quadrant where a second copy of the pattern could originate is termed a *source*. In quadrant I (upper left) for example, the origin or upper left corner is a source. If other sources exist in quadrant I, then a periodicity vector is defined from the origin to the nearest such source. Once the lengths of the periodic vectors (or *basis vectors*) in quadrants I and II (lower left) have been determined, the periodicity class of the pattern is known. The basis vectors for quadrants I and II are, respectively, $\vec{v}_1 = r_1\vec{y}+c_1\vec{x}$ and $\vec{v}_2 = r_2\vec{y}+c_2\vec{x}$, where $\vec{y}$ is the unit vector in the direction of increasing row index and $\vec{x}$ is the unit vector in the direction of increasing column index. It was shown that there are four classes of periodicity for rectangular arrays. For this paper, we define a new subclass of patterns which we call *dense lattice periodic*. The optimality of our text scanning algorithm depends on knowing whether or not the pattern is dense lattice periodic.

Definition: A pattern is *dense lattice periodic* if the coefficients of its basis vectors meet the following two restrictions:

1. All of $|r_1|, |r_2|, |c_1|, |c_2| < \lceil \frac{m}{2} \rceil$.

2. One of $|r_1| + |r_2| < \lceil \frac{m}{2} \rceil$ or $|c_1| + |c_2| < \lceil \frac{m}{2} \rceil$.

Essentially, a pattern is dense lattice periodic if it is lattice periodic and an $\frac{m}{2} \times \frac{m}{2}$ block of *text* can contain at least three non-colinear sources.

3 Two Dimensional Matching

The *Exact Two-dimensional Matching Problem* is defined as follows:

Input: $n \times n$ square text matrix $T[1 \ldots n, 1 \ldots n]$ and $m \times m$ square pattern matrix $P[0 \ldots m-1, 0 \ldots m-1]$.

Output: All locations in T where P occurs.

For simplicity, we have defined the problem in terms of square arrays, but *our algorithm works for any rectangular array.* We present below an overview of our algorithm. In the discussion that follows, the term *candidate source* or merely *source* refers to a location in the text that *may* correspond to the origin of a pattern occurrence. Our use of two dimensional periodicity will be apparent from the overview.

Algorithm Overview:

As in many pattern matching algorithms our algorithm consists of a *pattern preprocessing* part and a *text scanning* part.

Pattern Preprocessing: The pattern is analyzed for periodicity as described in [AB92]. From this analysis we obtain the periodicity class of the pattern and a witness table which indicates, for any offset of two copies of the pattern, if the copies can overlap without mismatch or, if not, the location of a mismatch.

Text Processing: Performed in two phases:

1) *Block Compatibility:* The text is partitioned into disjoint blocks of size $\frac{m}{2} \times \frac{m}{2}$. Within any block, only *compatible* candidate sources remain. A set of sources is compatible if for any two sources in the set, the two copies of the pattern originating at the sources can overlap without any mismatches. This phase differs in its details depending upon whether or not the pattern is lattice periodic.

2) *Candidate Verification:* We select a constant number of pattern elements with which to compare each text element. Each text element is then compared with its assigned pattern elements and we propogate the results of the tests to the candidates, thereby verifying which of the candidates are actual occurrences of the pattern.

4 Pattern Preprocessing

We begin with the preprocessing part of the algorithm. Its input is the $m \times m$ pattern P. The output of this step is 1) the pattern's periodicity class and 2) a table $Witness[-\frac{m}{2} \ldots \frac{m}{2}, 0 \ldots \frac{m}{2}]$. $Witness$ indicates whether two potential sources are compatible or if not, gives a location that witnesses a mismatch in the overlap of the patterns. Specifically, given a source S_1 at text location $T[r, c]$ and a source S_2 at text location $T[r+i, c+j]$, if $Witness[i, j] = [m, m]$ then the sources are compatible. Otherwise, if $Witness[i, j] = [a, b]$, then text location $T[r+i+a, c+j+b]$ matches at most one of pattern locations $P[a, b]$ (the pattern for S_2) or $P[i+a, j+b]$ (the pattern for S_1), i.e. $P[a, b] \neq P[i+a, j+b]$.

The witness table construction can be done in time $O(\log m)$ using $O(m^2)$ CRCW processors or in

$O(\log^2 m)$ time using $O(m^2/\log m)$ CREW processors. For full details of this algorithm see [AB92].

5 Block Compatibility

Initially, we assume that every location in the text is a candidate source. In the *block compatiblity step*, we partition the text into disjoint blocks of size $\frac{m}{2} \times \frac{m}{2}$ and reduce the number of candidates so that any two candidate sources within a block are compatible. We eliminate candidates only when they are inconsistent with the text. If the pattern is *dense lattice periodic*, the blocks are built up in stages from smaller blocks. As we combine blocks, we maintain the property of compatibility of sources within a block. If the pattern is *not* dense lattice periodic, then we treat the columns of the text independently, dividing them into disjoint strips of length $\frac{m}{2}$, again by joining smaller strips and maintaining compatibility of the candidates within the strips. Finally, $\frac{m}{2}$ of the strips of length $\frac{m}{2}$ are joined into a block, and all the candidates within the block are simultaneously tested for compatibility with each other.

Definition: A *k-block* $B[i,j]$ for $k = 0, \ldots, \log m - 1$, $i, j = 0, \ldots, \frac{n}{2^k} - 1$ is:

$$T[i \cdot 2^k + 1 \ldots (i+1) \cdot 2^k, j \cdot 2^k + 1 \ldots (j+1) \cdot 2^k]$$

Algorithm A *Block Compatibility*

Input: Text T, pattern P and $Witness$ table.

Output: $Cand[i,j]$ where $Cand[i,j] =$ `T` if $T[i,j]$ is a candidate source and $Cand[i,j] =$ `F` otherwise, such that for each $(\log m - 1)$-*block* of text, the set of candidate sources within

the block are compatible and no candidate has been eliminated except when it is inconsistent with the text.

In the next two sections, we describe the different procedures for Algorithm A based on whether or not the pattern is dense lattice periodic.

5.1 Patterns That Are Not Dense Lattice Periodic

For this section, the following lemma is essential:

Lemma 1 *If, within any* $\frac{m}{2} \times \frac{m}{2}$ *block of text, two or more copies of the pattern originate within a single row* and *two or more copies originate within a single column, then the pattern is dense lattice periodic.*

Proof: If two sources occur within the same row, then there is a quadrant I periodicity vector $\vec{v}_1 = r_1\vec{y} + c_1\vec{x}$ with $r_1 = 0$ and $c_1 < \lceil \frac{m}{2} \rceil$. Similarly, if two sources occur within the same column, then there is a quadrant II periodicity vector $\vec{v}_2 = r_2\vec{y} + c_2\vec{x}$ with $r_2 < \lceil \frac{m}{2} \rceil$ and $c_2 = 0$. The vectors meet the restricitions for dense lattice periodic patterns. $\diamond$

By lemma 1, patterns that are not dense lattice periodic cannot be periodic in both the first column and the first row. This means either that overlapping patterns aligned on their first column with a vertical separation of less than $\frac{m}{2}$, or overlapping patterns aligned on their first row with a horizontal separation of less than $\frac{m}{2}$, have a mismatch. From the pattern preprocessing, we can determine in which direction, either row or column, the pattern is not periodic. For the remainder of this section we will assume that the pattern is not column periodic.

Procedure A1 *Column Sparseness*

Breslauer and Galil [BG90] showed that for a non-periodic string of length m, finding the surviving candidate within a text segment of length $m/2$ can be done in linear work in $O(\log\log m)$ time on a CRCW PRAM since it is analagous to computing the maximum of $m/2$ numbers. Using a similar method, we find a single surviving candidate in each subcolumn in $O(\log m)$ time with $O(m/\log m)$ CREW processors. Note that in the two dimensional case the witness need not be within the processed column (but the time for witness verification is still constant).

Procedure A2 *Join Strips*

There is now one suvivor per subcolumn, so we have $\frac{m}{2}$ candidates in each $(\log m - 1)$-*block*. In constant time, we now perform all pairwise consistency checks within each block. Finally, we find the candidates that survive all such comparisons.

Theorem 1 *Procedures Column Sparseness and Join Strips are correct and run in linear work and space in time $O(\log m)$ on a CREW PRAM and in time $O(\log\log m)$ on a CRCW PRAM.*

Proof: The correctness of the procedures follows from the preceding comments. As pointed out above, Procedure Column Sparseness executes within the desired bounds. In Procedure Join Strips, there are $\frac{m}{2}$ candidates per block of size $\frac{m}{2}\times\frac{m}{2}$. We perform all $O(m^2)$ duels in work and space which is linear in the $O(m^2)$ block. Since each duel takes $O(1)$ time, all duels can be in time $O(\log m)$ on the CREW PRAM and time $O(\log\log m)$ on the CRCW PRAM. Finally, each candidate must

perform an OR over the answers of its duels to see if it is eliminated. Once again, the OR is computable optimally within the desired bounds. $\diamond$

5.2 Dense Lattice Periodic Patterns

When the pattern is dense lattice periodic, a text block of size $k \times k$ may contain as many as k^2 candidates so the column sparseness will not help us here. We use a different technique. As in Procedure Column Sparseness, the method is analagous to computing the maximum. Our goal is to eliminate enough candidates so that within each $(\log m - 1)$-*block*, all candidates are compatible.

Procedure A3 *Blocks Merge*

At each stage, we will do all pairwise duels between members of some set of blocks. The size of the set of merged blocks will be noted later, but depends on the type of processor used. The algorithm for finding the MAX works by finding the maximum of larger and larger blocks and relies on the fact that each block has a single maximal value. This implies that if we adapt the MAX algorithm, we must be able to both compare two blocks and eliminate a block in constant time. Since each block may contain more than one candidate, we need to show how to compare two blocks in constant work and how to mark blocks for elimination.

Having presented these subprocedures, we can then simply apply the standard MAX algorithm which, on a CREW PRAM combines 2 blocks per stage, giving a running time of $O(\log m)$, and on a CRCW PRAM combines 2^{2^i} blocks in stage i, giving a running time of $O(\log\log m)$.

5.2.1 Comparing Blocks in constant work

For each $k \times k$ block, we have only $\frac{k^2}{\log m}$ CREW or $\frac{k^2}{\log \log m}$ CRCW processors available. Therefore, if we require that every candidate look at a witness, we would need super-constant time per stage. Instead, we use a representative candidate from each block and find the witness (if one exists) for those candidates. We will treat the representative as the `MAX` value of this computation. Initially, the representative is the single candidate in its 0-*block*. We define $>$ over two representatives as follows:

Definition: For two representatives X and Y, let $X \gg Y$ mean that X occurs first in a lexicographic ordering of the indices, first by row and then by column. Then, $X > Y$ if one of the following three conditions holds:

1. X and Y are compatible and $X \gg Y$.
2. X is incompatible with Y and X is found compatible with the text at the witness and Y is not.
3. X is incompatible with Y, neither is found compatible with the text at the witness and $X \gg Y$.

We want a witness that can eliminate all the candidates in either of the two blocks. But, the witness w we get from the witness table may not fall within the common overlap of all the candidates in the two blocks. In this case, it is *always* possible to find an alternate witness w' that *does* fall within that overlap. In the following two lemmas, we show that the region of common overlap of

all the candidates for the two blocks has dimensions at least $\lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil$, the witness w lies within $\lceil \frac{m}{2} \rceil$ rows and or columns of this region and that such a region must always contain an alternate witness.

Lemma 2 *If two k-blocks, B_1 and B_2, $0 \leq k < \log m - 1$ lie within the same $(\log m - 1)$-block, then for any two candidates c_1 and c_2, $c_1 \in B_1, c_2 \in B_2$, the witness w for those two candidates (if it exists) lies within $\lceil \frac{m}{2} \rceil$ rows and/or columns of the common overlap of all the candidates in B_1 and B_2, which has size at least $\lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil$.*

Proof: Let *k-block* $B_1 = B(i,j)$ and $B_2 = B(s,t)$ and wolog, let $i \leq s$ and $j \leq t$, where $s - i < \lceil \frac{m}{2^{k+1}} \rceil$ and $t - j < \lceil \frac{m}{2^{k+1}} \rceil$ (so the B_i are in the same $(\log m - 1)$-*block*). The common overlap for the candidates in B_1 and B_2 contains the text region $T[(s+1)\cdot 2^k \ldots i \cdot 2^k + m, (t+1)\cdot 2^k \ldots j \cdot 2^k + m] = R_1$. Note that R_1 is at least as large as $\lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil$. The witness w for c_1 and c_2 occurs somewhere in the region $T[s \cdot 2^k + 1 \ldots (i+1) \cdot 2^k + m - 1, t \cdot 2^k + 1 \ldots (j+1) \cdot 2^k + m] = R_2$, depending upon the exact locations of c_1 and c_2 and their witness. Any point in R_2 is within $\lceil \frac{m}{2} \rceil$ rows and/or columns of R_1. $\diamond$

Lemma 3 *If $\vec{v}_1 = r_1\vec{y} + c_1\vec{x}$ and $\vec{v}_2 = r_2\vec{y} + c_2\vec{x}$ are the basis vectors of a dense lattice periodic pattern, and a lattice with such basis vectors is laid over the text T, then every block of size $\lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil$ must contain a lattice point.*

Proof: Consider a unit cell of the lattice on the text T. Label the nodes of the cell, in clockwise order, p_1, p_2, p_3 and p_4 where p_1 is the upper left corner of the cell. By the definition of basis

vectors from [AB92], $p_1\vec{p}_2 = \vec{v}_1 = r_1\vec{y} + c_1\vec{x}$ and $p_4\vec{p}_1 = \vec{v}_2 = r_2\vec{y} + c_2\vec{x}$. Because the pattern is lattice periodic, all of $|r_1|, |r_2|, |c_1|, |c_2| < \lceil \frac{m}{2} \rceil$. Therefore, for an $\lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil$ block of text to fit without containing a lattice point, it must fit within the rectangular area bounded by such a cell. (It can not, for example, squeeze between two adjacent nodes.) But, the largest such rectangular region has at most $\lceil \frac{m}{2} \rceil - 2$ columns if $|c_1| + |c_2| < \lceil \frac{m}{2} \rceil$ or at most $\lceil \frac{m}{2} \rceil - 2$ rows if $|r_1| + |r_2| < \lceil \frac{m}{2} \rceil$.

$\diamond$

By lemma 3, an alternate witness for w must lie within R_1. Additionally, it must lie on a point of the lattice defined by the basis vectors of the pattern, $\vec{v}_1$ and $\vec{v}_2$, and with origin at w. In order to find an alternate witness w', we will store the locations of the lattice points in an array *Alt*. The basic idea is as follows. For each possible region R_1, *Alt* will indicate the position of an alternate witness, permitting lookup in constant time. Since w always lies within $\lceil \frac{m}{2} \rceil$ rows and/or columns of R_1 (by lemma 2), the size of *Alt* is linear in the size of the pattern.

Let

$$r_{i,j} = i \cdot r_1 + j \cdot r_2$$

$$c_{i,j} = i \cdot c_1 + j \cdot c_2$$

be the coordinates of the lattice points reached from w along i vectors $\vec{v}_1$ and j vectors $\vec{v}_2$. We will compute the locations of all lattice points within an $m \times m$ block around w.

Procedure A3.1 *Alternate Witness Table*

For all $i, j = -m, \ldots, m$ pardo:

If $-m \leq r_{i,j} \leq m$ and $-m \leq c_{i,j} \leq m$

then $Alt[r_{i,j}, c_{i,j}] = [r_{i,j}, c_{i,j}]$.

The location of the lattice points with respect to w are now stored in Alt. In the remainder of this procedure, we fill out the remaining entries in Alt by propogating the locations of the lattice points up and left. The method is similar to the first and last "1" method described in section 6 and will not be further described here.

When we find the witness for two blocks $B_1 = B(i,j)$ and $B_2 = B(s,t)$, The upper left corner of region R_1 is at $T[(max(i,s)+1) \cdot 2^k, (max(j,t)+1) \cdot 2^k]$. If w is at $T[r_w, c_w]$ then the alternate witness w' is at $Alt[(max(i,s)+1) \cdot 2^k - r_w, (max(j,t)+1) \cdot 2^k - c_w]$.

5.2.2 Eliminating Candidates

When we run the `MAX` algorithm on the blocks, we only have constant time per comparison to eliminate all the surviving candidates per block. Instead, we simply keep a tree which mimics the computation tree, i.e. if two blocks are compared in some stage i, they are sibling nodes at height i from the leaves. Now, if some node needs deleting, by which we mean that all the candidates in the block represented by the node do not survive, we mark the node for deletion.

Such a deletion tree is of size $O(m^2)$. We need only start at the root and propogate the deletion bits down to the leaves. On either a CRCW or CREW PRAM, the processor allocation problem is staightforward and we can finish the computation in time proportional to the depth of the tree, i.e. within the same bounds as the `MAX` algorithm.

We summarize the discusion above with the following theorem.

Theorem 2 *Procedure Blocks Merge is correct and run in linear work and space in time* $O(\log m)$ *on a CREW PRAM and in time* $O(\log\log m)$ *on a CRCW PRAM.*

6 Candidate Verification

Within each $\log m$-*block*, all remaining candidates are now mutually compatible. Each text element $T[r,c]$ may be contained by several candidates, the *relevant* candidates. Let $B[b(r),b(c)]$, the *home block*, be the $\log m$-*block* containing $T[r,c]$, where the block index $b(i) = \lfloor i/\frac{m}{2} \rfloor$. Note that $T[r,c]$ may have relevant candidates in each of nine blocks, namely $B[b(r)-s, b(c)-t]$, $s,t = 0\ldots 2$. We call these nine blocks the *relavant* blocks.

Compatible candidates that are relevant to the same text element must agree on the expected character in that element. This leads to the following crucial observation: Every element $T[r,c]$ can be labeled with a matrix $M_{r,c}[s,t] \in \{\texttt{true},\texttt{false}\}$ where $M_{r,c}[s,t] = \texttt{true}$ means that $T[r,c]$ equals the unique pattern symbol expected by all relevant candidates in block $B[b(r)-s, b(c)-t]$ and $M_{r,c}[s,t] = \texttt{false}$ otherwise. Thus, every text element needs to be compared to a *single* pattern element per relevant block, and every candidate source that contains an element with the appropriate entry of M set to `false` is not a pattern appearance and can be discarded.

We proceed independently for each block. The following is the algorithm for processing each block.

Algorithm B *Candidate Verification for block* $B[i,j]$

Step B.1: Mark every text location $T[r,c]$ such that it has relevant candidates in $B[i,j]$ (*i.e.* such that $0 \leq b(r) - i \leq 2$ and $0 \leq b(c) - j \leq 2$,) with a *pattern coordinate pair* $\langle x, y \rangle$, where $\langle x, y \rangle$ are the coordinates of the pattern element $P[x,y]$ with which $T[r,c]$ should be compared.

There may be several options for some locations, namely, the position of the scanned text element relative to each of its relevant candidates. However, any will do since all candidate sources within block $B[i,j]$ are now compatible. If a location is not contained in any candidate source it is left unmarked. We will later see how this step is implemented.

Step B.2: Compare each text location $T[r,c]$ with $P[x,y]$, where $\langle x, y \rangle$ is the pattern coordinate pair assigned to $T[r,c]$ in the previous step. If $T[r,c] = P[x,y]$, then $M_{r,c}[b(r) - i, b(c) - j] \leftarrow$ `true`, else `false`.

Step B.3: Flag with a *discard* every candidate that contains a `false` location within its bounds.

This flagging is done by the same method as in step B.1.

Step B.4: Discard every candidate source flagged with a *discard*. The remaining candidates represent all pattern appearances.

Our only remaining task is to show how to mark the text elements with the appropriate pattern coordinate pairs. This implementation of this step relies on the following lemma.

Lemma 4 *[FRA88] The first and last 1 in a an array of length n can be determined in $O(1)$ time with $O(n)$ processors on a CRCW PRAM.*

Similarly, finding the first and last 1 in an array of lenght n can be accomplished optimally in $O(\log n)$ time on a CREW PRAM using a prefix sum.

For each block $B[i,j]$, we have an array $C_{i,j}[1 \ldots \frac{3m}{2}, 1 \ldots \frac{3m}{2}]$ in which we will do our computation. All ones in the array represent surviving candidates in $B[i,j]$ and all other positions contain zeros. The information in $C_{i,j}[r,c]$ applies to text element $T[i \cdot \frac{m}{2} + r, j \cdot \frac{m}{2} + c]$. For clarity, we now refer to this as text location $T[r,c]$ and drop the subscript on C.

The goal is to assign to each position $C[x,y]$ a pair (a,b) such that $C[a,b] = 1$ and both $0 \leq x - a < \frac{m}{2}$ and $0 \leq y - b < \frac{m}{2}$. We refer to such a relevant candidate as the *ruler* of $C[x,y]$. (The reality of the assumption that rulers are relavent is questionable.) We proceed first within columns and then within rows.

Independetly for each column c, if column c contains all zeros, do nothing. Otherwise, find the first and last occurence of a "1". Call the row position of those "1" r_f and r_l respectively. Now we have several cases:

$r < r_f$: Do nothing. There is no 1 in column c relevant to $C[c,r]$.

$r_f \leq r < r_l$: Set the ruler of $C[c,r]$ to be (c, r_f). Since $r_l - r_f \leq \frac{m}{2}$, then the candidate at $C[c,r_f]$ is close enough to $C[c,r]$ to be relevant.

$r_l \leq r \leq r_l + \frac{m}{2}$: Set the ruler of $C[c,r]$ to be (c, r_l). Similarly, the candidate at $C[c,r_l]$ is close enough to $C[c,r]$ to be relevant.

$r_l + \frac{m}{2} < r$: Do nothing. $C[c,r]$ is too far from any candidate in column c.

Now we propogate along the rows with the following modifications. Instead of finding the first and last candidate of each row, we find the the first and last position which has been assigned a ruler in the previous phase. Then if at some point we would set the ruler of $C[c, r]$ to be (c', r), we instead set it to be the ruler (c', r') of $C[c', r]$.

Theorem 3 *Algorithm B is correct and runs in time $O(\log m)$ with $O(m^2/\log m)$ CREW processors per block and linear space, thus $O(\log m)$ time, $O(n^2/\log m)$ CREW processors and linear space for all blocks. Similarly, the algorithm runs in optimally in constant time on a CRCW with linear space.*

Correctness: We need only show that if $C[a, b]$ is assigned ruler (x, y), then $T[x, y]$ is a candidate. But this follows directly from the case analysis above. The symmetric process of finding, for each candidate, if it has a relavent error can clearly be computed within the same bounds.

Time: The bottleneck in the compuation is finding the first and last "1" in each column and row. But by lemma 4, this step takes exactly the stated bounds. $\diamond$

Aknowledgements

The authors wish to thank S. Muthukrishnan and H. Ramesh for fruitful discussion.

References

[AB92] A. Amir and G. Benson. Two-dimensional periodicity and its application. *Proc. of the*

Third Ann. ACM-SIAM Symp. on Discrete Algorithms, Jan 1992.

[ABF92] A. Amir, G. Benson, and M. Farach. Alphabet independent two-dimensional matching. *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 59–68, 1992.

[AC75] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18:333–340, 1975.

[Bak78] T. J. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Comp*, 7:533–541, 1978.

[BG90] D. Breslauer and Z. Galil. An optimal $o(\log\log n)$ time parallel string matching algorithm. *SIAM J. Comput.*, 19:1051–1058, 1990.

[Bir77] R. S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6:168–170, 1977.

[BM77] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.

[CDR86] S.A. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneus writes. *SIAM Journal on Computing*, 15(1):87–97, 1986.

[CGR92] M. Crochemore, L. Gasieniec, and W. Rytter. Constant time optimal parallel algorithm for 2d-pattern matching. Manuscript, 1992.

[FRA88] F. Fich, R. Ragde, and A.Widgerson. Relations between concurrent-write models of parallel computation. *SIAM J. Comput.*, 17:606–627, 1988.

[Gal85] Z. Galil. Open problems in stringology. In Z. Galil A. Apostolico, editor, *Combinatorial Algorithms on Words*, volume 12, pages 1–8. NATO ASI Series F, 1985.

[Gal92] Z. Galil. A constant-time optimal parallel string-matching algorithm. *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 69–76, 1992.

[GP92] Z. Galil and K. Park. Truly alphabet independent two-dimensional pattern matching. *Proc. of the 33rd IEEE Annual Symp. on Foundation of Computer Science*, 1992.

[KLP89] Z. M. Kedem, G. M. Landau, and K. V. Palem. Optimal parallel suffix-prefix matching algorithm and application. *Proc. of the First Ann. ACM Symp. on Parallel Algorithms and Architectures*, pages 388–398, 1989.

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.

[MR92] S. Muthukrishnan and H. Ramesh. A constant time optimal parallel algorithm for two dimensional pattern matching. Manuscript, 1992.

[RK82] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, New York, 1982.

[Vis85] U. Vishkin. Optimal parallel pattern matching in strings. *Proc. 12th ICALP*, pages 91–113, 1985.

[Vis91] U. Vishkin. Deterministic sampling - a new technique for fast pattern matching. *SIAM J. Comp.*, 20:303–314, 1991.