PARALLEL PATTERN MATCHING WITH SCALING

H. Mongelli*
Universidade Federal do Mato Grosso do Sul
Departamento de Computação e Estatística
Campo Grande, MS 79070-900 - Brazil

and

S. W. Song†
Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação
São Paulo, SP 05508-900 - Brazil

ABSTRACT

Given a text and a pattern, the problem of pattern matching consists of determining all the positions of the text where the pattern occurs. When the text and the pattern are matrices, the matching is termed bidimensional. There are variations of this problem where we allow the matching using a somehow modified pattern. A modification that we will allow is that the pattern can be scaled. We propose a new parallel algorithm for this problem, under the CGM (Coarse Grained Multicomputer) model. This algorithm requires linear local computing time in the input, linear memory and uses only one communication round, during which at most a linear amount of data is exchanged. To the best of our knowledge, there are no known parallel algorithms for the bidimensional pattern matching problem with scaling in the literature. This proposed algorithm was implemented in C, using the PVM interface and was executed on a Parsytec PowerXplorer parallel machine. The experimental results obtained were very promising and showed significant speedups.

Keywords:

pattern matching, scaling, parallel algorithm, coarse-grained multicomputer.

1 Introduction

In this paper we present an efficient parallel algorithm for the bidimensional pattern matching problem with scaling. We use the CGM (*Coarse Grained Multicomputers*) model. Let T be a matrix called the *text* matrix and P a matrix called the *pattern* matrix. Assume T and P to be, respectively, $N \times N$ and $m \times m$ and their elements belonging to some finite alphabet Σ. The bidimensional pattern matching with scaling problem consists of determining the positions in T where P with scale k occurs, for all integer values of k, $1 \leq k \leq \lfloor N/m \rfloor$. When $k = 1$, we have a normal pattern matching problem (without scaling). An occurrence of the pattern with scale k (or k-occurrence) is the position (i, j) of the text matrix T where each

*Email: mongelli at dct. ufms. br
†Email: song at ime. usp. br

element of the submatrix $T[i+x, j+y]$, with $0 \leq x, y < km-1$, is equal to the element $P[\lfloor \frac{x}{k} \rfloor, \lfloor \frac{y}{k} \rfloor]$.

Pattern matching with scaling appears, for example, in some image processing applications where the source of the captured image can be at different distances [1].

Consider a problem with input size n. We use the CGM (Coarse-Grained Multicomputer) model [2,3], with p processors each with $O(n/p)$ local memory. Usually we have $n \gg p$. The processors are connected through an interconnection network that allows point-to-point communication among processors. A CGM algorithm is described as a number of alternating computing rounds and communication rounds. In a computing round, each processor computes independently and considers the data stored in its memory. In a communication round, data can be exchanged among processors. More specifically, in a communication round, each processor can send $O(n/p)$ data to other processors and receive $O(n/p)$ data from other processors. In the CGM model, the goal is to minimize the number of supersteps or rounds needed to solve the problem.

We propose a CGM algorithm for the bidimensional pattern matching problem with scaling that requires linear local computing time and only one communication round.

The straightforward idea of performing pattern matching for each scale would give an inefficient algorithm. Amir *et al* [1] present a sequential algorithm, based on the KMP algorithm [4], to determine all the occurrences for each scale k in linear time on the input size. Ferragina and Luccio [5] consider string searching (without scaling) in the BSP model and coarse-grained computing models. Boxer *et al* [6] present CGM algorithms for geometric pattern recognition. More recently, Amir *et al* [7] present a sequential algorithm for the bidimensional pattern matching problem with scaling where the scale is a real number.

To the best of our knowledge no coarse-grained algorithms are known for bidimensional pattern matching with or without scaling. Our algorithm is based on [1] and uses the range minima algorithm of Mongelli and Song [8,9]. The proposed algorithm uses $O(N^2/p)$ local computing time and $O(N^2/p)$ memory, and requires only one communication round in which at most $O(Nm/\sqrt{p})$ data are exchanged, with $m \leq N/\sqrt{p}$. We have implemented the proposed algorithm using C and PVM on the *PowerXPlorer* distributed memory machine with 16 processors. The efficiency of the algorithm results from the minimum amount of data transmitted (equal to the number of possible occurrences), in addition to linear local computing time. As a result, the experimental results obtained are very promising and show considerable speedups, even with the Ethernet-speed connection of this particular machine that does not excel in communication. For small patterns we get almost optimum speedups.

In this work, for simplicity, we consider T and P as square matrices. The algorithm however works for rectangular matrices. The sizes of the text and pattern matrices are therefore, respectively, N^2 and m^2.

The paper contains the main result of [10] which also presents CGM algorithms for the unidimensional pattern matching with and without scaling, and bidimen-

sional pattern matching without scaling.

In Section 2 some important definitions are presented. In Section 3 we present the bidimensional pattern matching problem. In Section 4 we show how to distribute the input data among the processors. The main algorithm is presented in Section 5. We obtain the time required by this algorithm in Section 6 and show experimental results in Section 7. Then we give concluding remarks in Section 8.

2 Preliminary Definitions

2.1 CGM (Coarse Grained Multicomputer) Model

The CGM model was proposed by Dehne [2,3]. It is similar to the BSP model (*Bulk-Synchronous Parallel*)[11]. However, it uses only two parameters: the input size n and the number of processors p, each with a local memory of size $O(n/p)$. The term *coarse grained* means the size of the local memory is much larger than $O(1)$. Each processor is connected by a router that can send messages in a point-to-point fashion. A CGM algorithm consists of an alternate sequence of computing and communication rounds separated by a barrier synchronization. A computing round is equivalent to a superstep in the BSP model. In a computing round, we usually use the best sequential algorithm in each processor to process its data locally.

In a communication round, each processor exchanges a maximum of $O(n/p)$ data with other processors. In the CGM model, the communication cost is modeled by the number of communication phases. Some algorithms for Computational Geometry and graph problems require a constant number or $O(\log p)$ communication rounds [12,2,3,13]. Contrary to a PRAM algorithm, that is often designed for $p = O(n^k)$, with $k \in \mathcal{N}$ and each processor receives a small number of input data, in the CGM model we consider the more realistic cases where $n \gg p$. The CGM model is particularly suitable in current parallel machines in which the global computing speed is considerably larger than the global communication speed.

2.2 Factorized Representation of a String

Definition 1 *Let $S = \sigma_1\sigma_2\cdots\sigma_n$ be a string over some alphabet Σ. The factorized representation [1] of string S is the string $\sigma'^{r_1}_1\sigma'^{r_2}_2\cdots\sigma'^{r_{\hat{n}}}_{\hat{n}}$, $\hat{n} \leq n$, such that:*

1. *$\sigma'_i \neq \sigma'_{i+1}$, for $1 \leq i < \hat{n}$; and*
2. *S can be described as the concatenation of the symbol σ'_1 repeated r_1 times, the symbol σ'_2 repeated r_2 times, ..., and the symbol $\sigma'_{\hat{n}}$ repeated $r_{\hat{n}}$ times. (The symbol σ'_i repeated r_i times is written as $\sigma'^{r_i}_i$.)*

2.3 Blocks and k-Blocks

Definition 2 *Let T be an $N \times N$ text matrix and P an $m \times m$ pattern matrix. A k-block, for a positive integer k, in position i_1 of a column j_1 of T, is the submatrix $T' = T[i_1, \ldots, i_2; j_1, \ldots, j_1 + km - 1]$ that satisfies the following conditions: all the rows of T' are equal; no row can be added to T' without violating the previous condition of all rows are equal, that is, $T[i_2; j_1, \ldots, j_1+km-1] \neq T[i_2+1; j_1, \ldots, j_1+$*

$km-1]$ *and* $T[i_1; j_1, \ldots, j_1 + km - 1] \neq T[i_1 - 1; j_1, \ldots j_1 + km - 1]$. *When* $k = 1$, *we say simply block.*

2.4 The lowest common ancestor problem (LCA)

Definition 3 *(Lowest common ancestor) The lowest common ancestor [14,15,16] of two vertices* u *and* v *of a rooted tree is the vertex* w *that is an ancestor of* u *and* v *and that is farthest from the root. We denote* $w = LCA(u, v)$.

Definition 4 *(Lowest common ancestor problem)*

The lowest common ancestor problem consists of pre-processing a rooted tree such that queries of the type $LCA(u, v)$, *for any vertices* u *and* v *of the tree, can be answered in constant sequential time.*

2.5 Range Minima Problem

Definition 5 *Given a vector of* n *real numbers* $A = (a_1, a_2, \ldots, a_n)$, *define* $MIN(i, j) = \min\{a_i, \ldots, a_j\}$. *The range minima problem [14,8,9] consists of pre-processing vector* A *such that queries* $MIN(i, j)$, *for any* $1 \leq i \leq j \leq n$, *can be answered in constant time.*

2.6 Cartesian Trees and k-height Cartesian Trees

Definition 6 *(Cartesian tree)*

Let $A_{0,n-1} = (a_0, a_1, \ldots, a_{n-1})$ *be an array of* n *distinct real numbers, the Cartesian tree [17] of* $A_{0,n-1}$ *is a rooted binary tree whose nodes are labeled with the elements of* A. *The tree root has label* $a_m = \min\{a_0, a_1, \ldots, a_{n-1}\}$. *The left child of the root is the Cartesian tree of* $A_{0,m-1} = (a_0, a_1, \ldots, a_{m-1})$, *and the right child is the Cartesian tree of* $A_{m+1,n-1} = (a_{m+1}, \ldots, a_{n-1})$. *The Cartesian tree of an empty array is the empty tree.*

Definition 7 *(k-height Cartesian tree)*

A k-height Cartesian tree [1] of $L = l_1, \ldots, l_N$ *is a rooted binary tree defined recursively as follows:*

Let $l_{root} = \max\{l_1, \ldots, l_N\}$ *(if* $\max\{l_1, \ldots, l_N\}$ *is obtained at more than one index then root is the maximum such index). Then,*

- l_{root} *is the root of the k-height Cartesian tree*
- *the left child of the root is the k-height Cartesian tree of* $[l_1, \ldots, l_{root-k}]$
- *the right child of the root is the k-height Cartesian tree of* $[l_{root+k}, \ldots, l_N]$

3 Bidimensional Pattern Matching with Scaling

The bidimensional pattern matching with scaling problem takes as input an $m \times m$ pattern matrix P and an $N \times N$ text matrix T, where $m \leq N$ and as output all the positions of T with k-occurrences of P, for all $k = 1, \ldots \lfloor N/m \rfloor$. Figure 1 shows an example of pattern matching with scaling. The text and pattern matrices have therefore N^2 and m^2 elements, respectively.

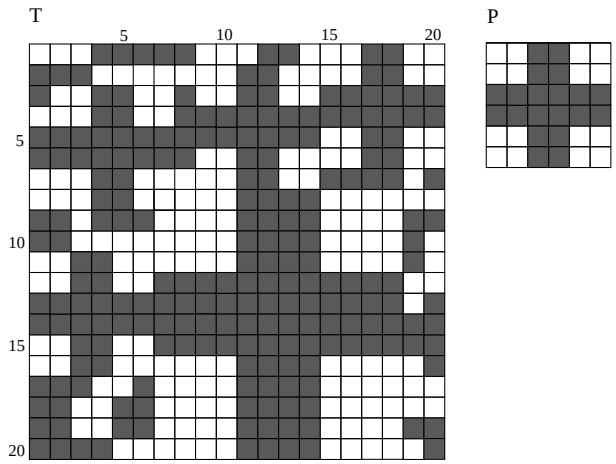

Fig. 1: Example of text matrix T and pattern matrix P with occurrences of P in T in positions $[1, 15]$, $[2, 9]$, $[3, 2]$, $[11, 1]$ with scale 1 and in position $[8, 7]$ with scale 2.

Amir *et al* [1] present a sequential algorithm, based on the KMP-algorithm [18,4,19], of time complexity $O(N^2)$ (thus linear in the input size N^2). It consists of two steps: pattern analysis and text analysis. The input data are preprocessed and stored in data structures that ensure the linear processing time of the algorithm. The k-occurrences are determined separately for each scale k. The linearity of the algorithm is obtained by reducing, for each k, the number of columns which may contain the beginning of a k-occurrence and in these columns reduce the amount of positions where there may exist a k-occurrence. The proofs of the time complexity and correctness of the algorithm can be seen in [1,10].

In Amir *et al*'s algorithm [1], the text and pattern are given in factorized representations. The factorized representation of a matrix corresponds to the matrix formed by the factored representation of each of its rows.

Our proposed CGM algorithm is based on Amir's algorithm. The CGM algorithm has as input the factorized representation FT of the text matrix T and the factorized representation FP of the pattern matrix P. The CGM algorithms has the following main steps.

CGM Algorithm:

1. Preprocessing of input data: construction of data structures (the C string and the suffix tree).
2. Pattern analysis.
3. Text analysis.

These steps are described in the next sections. We suppose the data are distributed among the processors. The way it is done is shown in the next section.

4 Data Distribution

Consider the text and pattern matrices T and P with N^2 and m^2 elements, respectively. Recall each of the p processors has $O(N^2/p)$ memory.

First we assume that the pattern is stored entirely in each processor. In other words, we assume $m^2 \leq N^2/p$, where $m \leq \frac{N}{\sqrt{p}}$.

We distribute the data such that no more than $O(N^2/p)$ data is transmitted in a communication round, as limited by the CGM model. Each processor will receive a submatrix of T of size $\frac{N}{\sqrt{p}} \times \frac{N}{\sqrt{p}}$. Without loss of generality, we assume p to be a perfect square and $\sqrt{p}$ to be a divisor of N. Define also $n = \frac{N}{\sqrt{p}}$. Each processor thus receives an $n \times n$ submatrix as input.

With such a data distribution in the processors, we have the following mapping that associates submatrices to processors: submatrix $S[(i-1)n+1, \ldots, in; (j-1)n+1, \ldots, jn]$ will be stored in processor of index k or, to make the notation simpler, processor k, with $k = (i-1)\sqrt{p} + j - 1$, where $1 \leq i, j \leq \sqrt{p}$ and $0 \leq k \leq p-1$ and will be called submatrix $S_{i,j}$.

The relation between neighbor processors is illustrated in Figure 2 for $p = 25$ processors. This relation can be extended to neighborhood relation in submatrices. So we can say that a submatrix $S_{i,j}$ stored in processor k has, for example, a Southeast neighbor $S_{i',j'}$ if this neighbor is stored in $k + \sqrt{p} + 1$ and we have $i' = i + 1$ and $j' = j + \sqrt{p}$. We will use the cardinal points (East, South, West and North) to identify the neighbors of a processor or submatrix.

The submatrices stored in processors k with $\sqrt{p} < k < p - \sqrt{p}$, $k \bmod \sqrt{p} > 0$ and $(k+1) \bmod \sqrt{p} > 0$, have the following adjacent submatrices: to the right, at processor $k+1$, called *East neighbor (E)*; to the left, at $k-1$, called *West neighbor (W)*; below, at $k + \sqrt{p}$, called *South neighbor (S)*; and above, stored at $k - \sqrt{p}$, called *North neighbor (N)*.

These submatrices are related to other diagonally adjacent submatrices as follows: the submatrix stored at processor $k + \sqrt{p} + 1$, called *Southeast neighbor (SE)*; the submatrix stored at $k + \sqrt{p} - 1$, called *Southwest neighbor (SW)*; the submatrix stored at $k - \sqrt{p} - 1$, called *Northwest neighbor (NW)*; and the submatrix stored at $k - \sqrt{p} + 1$, called *Northeast neighbor (NE)*.

Some submatrices, due to their location (for example at the border), present less neighbors. The submatrices stored at processors k: with $k \bmod \sqrt{p} = 0$, for

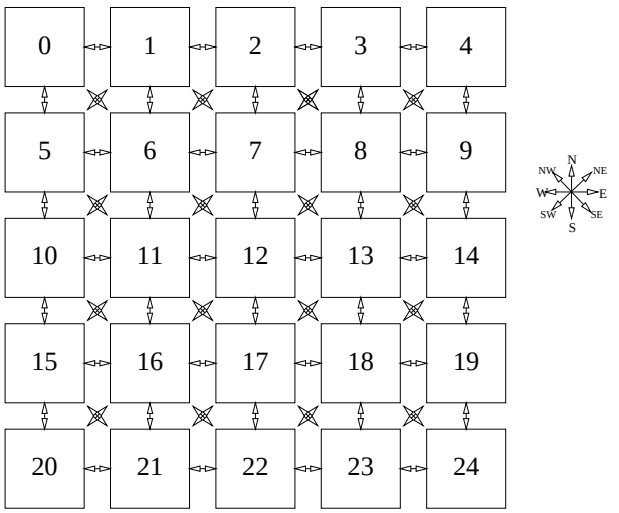

Fig. 2: Neighborhood relation between processors considering data distributed in $p = 25$ processors.

$0 < k < p - \sqrt{p} - 1$, have neighbors N, NE, E, SE and S; with $0 < k < \sqrt{p} - 1$, have neighbors E, SE, S, SW and W; with $(k+1) \bmod \sqrt{p} = 0$, for $\sqrt{p} < k < p - 1$, have neighbors S, SW, W, NW and N; with $p - \sqrt{p} < k < \sqrt{p} - 1$, have neighbors W, NW, N, NE and E; with $k = 0$, have neighbors E, SE and S; with $k = \sqrt{p} - 1$ have neighbors S, SW and W; with $k = p - 1$ have neighbors W, NW and N; and with $p - \sqrt{p}$ have neighbors N, NE and E.

In Figure 2, processors 6, 7, 8, 11, 12, 13, 16, 17 and 18 have neighbors in all directions; 0 has neighbors S, E and SE; 4 has neighbors S, W and SW; 20 has neighbors N, E and NE; 24 has neighbors N, W and NW; 1, 2 and 3 have neighbors E, SE, S, SW and W; and, 21, 22 and 23 have neighbors W, NW, N, NE and E.

When we look for occurrences of the pattern P in the text matrix T, we should search the entire pattern in the text as whole. Thus, the pattern can occur entirely in the submatrix stored locally in a processor or can start at a certain submatrix position of a processor and spread to submatrices of neighbor processors. In this case, we have an occurrence of the pattern in a boundary region. The *boundaries* will be submatrices of matrices stored locally and will be designated by the cardinal points. The existence of boundaries in a submatrix depends on the existence of a corresponding neighbor processor in that direction. For example, if a submatrix has a neighbor in NW direction, it will have a NW boundary. Thus, the *North and South boundaries* are horizontal bands or strips $[1, \ldots, m-1; 1, \ldots, n]$ and $[n-m+2, \ldots, n; 1, \ldots, n]$, respectively; the *East and West boundaries* are vertical bands or strips $[1, \ldots, n; n-m+2, \ldots, n]$ and $[1, \ldots, n; 1, \ldots, m-1]$, respectively; the *Northeast, Southeast, Southwest and Northwest boundaries* are square matrices $[n-m+2, \ldots, n; 1, \ldots m-1]$, $[n-m+2, \ldots n; n-m+2, \ldots, n]$, $[1, \ldots, m-1; n-m+2, \ldots, n]$ and $[1, \ldots, m-1; 1, \ldots m-1]$, respectively. Figure 3 illustrates these

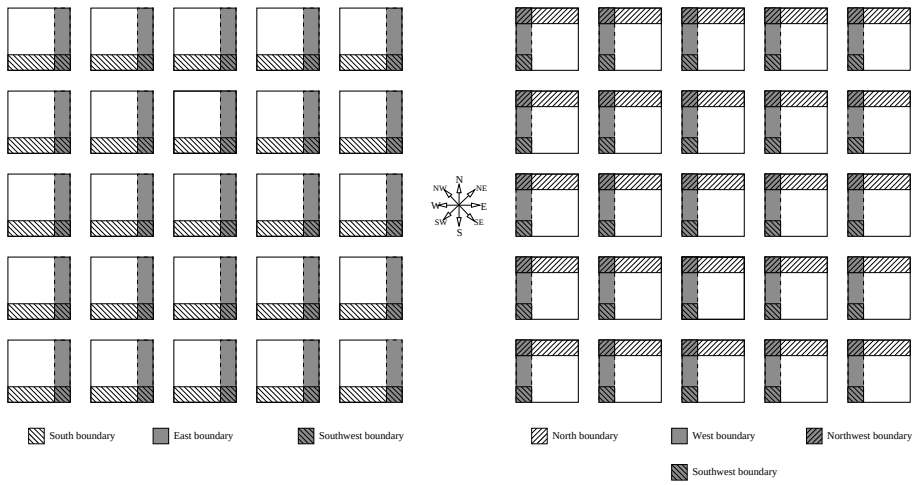

Fig. 3: Representing the boundaries South(S), East(E) and Southeast (SE) and boundaries North(N), West(W), Northwest(NW) and Southwest(SW).

boundaries.

5 The proposed CGM Algorithm

With the data distributed as discussed above, each processor receives an $n \times n$ submatrix S, where $n = N/\sqrt{p}$, as well as the entire $m \times m$ pattern P. We should be careful about the range of values the scale k can assume. In the sequential algorithm, the largest possible value for the scale k is $\lfloor N/m \rfloor$. In the CGM case, this value is $\lfloor n/m \rfloor = \lfloor N/(m\sqrt{p}) \rfloor$. The sequential algorithm uses the factorized representations of the text and pattern matrices. Thus, we first obtain the factorized representations, denoted FS and FP, of matrices S and P, respectively.

In the following sections we describe the steps of the CGM algorithm to be executed in each processor.

5.1 Pre-Processing of Input Data

Initially we obtain a string C formed by just concatenating all the rows of FS followed by $\lfloor n/m \rfloor$ strings: concatenation of the rows of FP and concatenation of the rows of P, for each scale k, with $2 \leq k \leq \lfloor n/m \rfloor$, in their factorized representations.

We then construct the suffix tree [20] ST of C and apply to it the LCA algorithm. With this, we will get constant time in the comparison of a substring of the text and a row of P^k, $1 \leq k \leq \lfloor n/m \rfloor$ [1].

We also construct auxiliary data structures to reduce the amount of positions to be analyzed as candidates to be the starting points of k-occurrences. These structures are obtained by going through the factorized rows of the text matrix and storing the needed information in arrays. The detailed description of these structures is not relevant to this paper, and can be found in [1,10].

5.2 Pattern Analysis

After preprocessing, we do a pattern analysis in each processor, where the pattern is stored. This analysis is described below.

Let $P_1, P_2, \ldots, P_m$ be the rows of P and $FP_1, FP_2, \ldots, FP_m$ the respective factorized representations of these rows. Consider each of these factorized representations FP_i as a *symbol*. We have then the sequence $FP_1, FP_2, \ldots, FP_m$ seen as a unidimensional string R. On the other hand, we can describe FP as row FP_{l_1} repeated r_1 times (the block in row l_1), followed by row $FP_{l_2} (\neq FP_{l_1})$ repeated r_2 times, followed by other blocks until, finally, we have row FP_{l_α} $(\neq FP_{l_{\alpha-1}})$ repeated r_α times, where $l_1 = 1$, $l_k = r_1 + r_2 + \cdots + r_{k-1}$ and $r_1 + r_2 + \cdots + r_\alpha = m$. Formally, a pair (FP_{l_k}, r_k) is a block of r_k rows in row l_k. So the factorized representation of R, called FR, is given by this sequence of pairs.

The obtention of the factorized representation FR of R is done in time proportional to $|FP_1| + |FP_2| + \cdots + |FP_m|$.

Assume $\alpha > 2$, that is, there exist at least 3 blocks in FR. For $\alpha \leq 2$ there exists a solution with the same complexity. Let $\hat{FR}$ be the subsequence of FR that starts at the second element and ends at the next to last element. That is, $\hat{FR} = (FP_{l_2}, r_2) \ldots (FP_{l_{\alpha-1}}, r_{\alpha-1})$.

We compute a prefix function for the string $\hat{FR}$, using the suffix tree ST. (This function is described for the KMP algorithm [18].) It consists of calculating for each element (FP_{l_i}, r_i), with $2 \leq i < \alpha$, of $\hat{FR}$, the length of the largest prefix of $\hat{FR}$ which is a proper suffix of (FP_{l_i}, r_i). This function is used to direct the searches of k-occurrences. Obtention of this function takes $O(\alpha)$ time, because we use LCA queries on the suffix tree ST.

Because of the boundary regions, we need to construct some matrices whose columns contain information to direct the search of *candidate positions* at boundary regions. These candidate positions are those that may be (or not) k-occurrences and that depend on data from neighbor processors and on *confirmation positions*. Confirmation positions are those that will be sent to neighbors to confirm if a candidate position is a global k-occurrence.

5.3 Text Analysis

This step considers two cases, depending on the pattern. Assume that there exist at least two rows, i_1 and i_2, of the pattern where $P[i_1; 1, \ldots, m] \neq P[i_1 + 1; 1, \ldots, m]$ and $P[i_2; 1, \ldots, m] \neq P[i_2 + 1; 1, \ldots, m]$. In case this assumption does not hold, the solution is easily obtained. Furthermore, this easy solution allows us to assume that at least one of the rows of the pattern has a factorized length larger than 1. That is, there exists at least one row whose symbols are not all equal. If each pattern row consists of a single repeating symbol, then we view the pattern in column major order (a 90° rotation) and thereby get a pattern with no row changes. The easy solution can then be applied to this case. Under this assumption, we consider two cases: case A, where in any $m/2$ successive rows of the pattern there exist at least one row of factorized length greater than 1 and case B, where there exist at least $m/2$ consecutive rows of the pattern with factorized length equal to 1.

The proposed CGM algorithm, as the sequential one, considers each scale k, $1 \leq k \leq \lfloor N/m \rfloor$, separately. The k-occurrences are determined comparing the pattern and the text, using as unit the k-blocks. Before searching the k-occurrences, we try to reduce the number of positions where these occurrences may exist.

Since we consider each scale k separately, the boundary limits depend on the value of k. Thus, the North and South boundaries are horizontal strips $[1, \ldots, km-1; 1, \ldots, n]$ and $[n-km+2, \ldots, n; 1, \ldots, n]$, respectively; the East and West boundaries are vertical strips $[1, \ldots, n; n-km+2, \ldots, n]$ and $[1, \ldots, n; 1, \ldots, km-1]$, respectively; the Northeast, Southeast, Southwest and Northwest boundaries are square matrices $[n-km+2, \ldots, n; 1, \ldots km-1]$, $[n-km+2, \ldots n; n-km+2, \ldots, n]$, $[1, \ldots, km-1; n-km+2, \ldots, n]$ and $[1, \ldots, km-1; 1, \ldots km-1]$, respectively.

5.4 Case A

For this case we reduce initially the regions of the text to search the pattern with scaling. For $k = 1$, all the columns are selected. For other values of k, only $\lfloor N/k \rfloor$ columns are selected. In these columns, determine lists of row index pairs, called *row intervals*, that can contain the end of k-blocks. These intervals are obtained from arrays constructed during the preprocessing, using range minima query.

From the lists of intervals, we do a second reduction where we obtain lists of the end of k-blocks, using k-height Cartesian trees that are constructed with the data stored in arryas obtained during preprocessing of the input. These lists are then used to obtain the k-occurrences. The objective of these reductions is to obtain, finally, the linear time to determine all the k-occurrences. We examine the lists of the end of k-blocks and, by comparing with the pattern, obtain the k-occurrences. More details can be obtained in [1,10].

In each processor, first we obtain the k-occurrences of the pattern in the locally stored text. Then, for each scale k, we determine the confirmation positions on North, Northeast, Northwest, West and Southwest boundaries, depending on the processor index. These confirmation positions, with the respective k value, are sent to the respective neighbor. Additional information that serves to obtain the candidate positions are also sent, when they exist, to the North neighbor. The candidate positions are determined during the next step, again for each scale k separately. Having these candidate and confirmation positions, these are combined together to determine the k-occurrences in the boundary regions.

5.5 Case B

In this case, we have a simpler pattern matrix. We use also a reduction in the search regions. In this reduction, we use the algorithm of unidimensional pattern matching with scaling (see [1,10]) to exclude positions where we cannot have k-occurrences of the pattern. We reduce the amount of positions to be tested as k-occurrences that can thus be obtained in linear time. The positions not eliminated are verified to see if they can or cannot be the beginning of a k-occurrence. This verification is done in constant time and consists of comparing the text submatrix starting from that position with the pattern.

The obtention of k-occurrences in the boundary regions for this case is a little different. Since we use the unidimensional pattern matching with scaling algorithm [1,10], additional information about the text submatrix rows should be sent to the West neighbor, together with the confirmation positions, to obtain the candidate positions of the West neighbor. The obtention of the confirmation positions at the North boundary and candidate positions at the South boundary is symmetric and uses only local data. For the other regions, the candidate or confirmation positions are obtained by combining the two. The confirmation positions are obtained for each scale k separately and then all the positions are sent together with the respective scale value. The candidate positions are also obtained separately for each scale k and kept together with the respective scale value. The candidate positions (computed locally) and the confirmation positions (received) can then be combined to determine the k-occurrences at the boundaries.

Observe that, in both case A and B, we use the same idea of reducing the amount of positions to search the k-occurrences before doing the search proper, in order to get the linear time complexity.

6 Discussion of the time complexity

The algorithm takes $O(n^2)$ local computing time. Requires only one communication round where at most $O(n^2)$ data are transmitted and requires $O(n^2)$ memory. Recall $n = N/\sqrt{p}$ and the input submatrix to each processor is of size $n \times n$. The time analysis and correctness proof of the algorithm is basically technical and can be found in [10].

In any of the two cases, the preprocessing of the input data takes $O(n^2)$ time. In case A, the pattern analysis step takes $O(n^2)$ time. This step does not exist in case B. This step is executed on the locally stored data and does not involve communication among processors.

In the text analysis step, in both cases, the determination of the k-occurrences in the text submatrix (except the boundary regions) takes linear time, since we use the algorithm of Amir *et al* [1]. The obtention of the k-occurrences in the boundary regions involves: linear local computing time to obtain the confirmation positions; the sending of at most $O(n^2)$ data to neighbors; again linear local computing time to obtain the candidate positions; and finally the combining of candidate positions (locally generated) and confirmation positions (received) to obtain the k-occurrences in the boundary regions.

7 Implementation Results

For the experimental part, we implemented the algorithm in C and used the PVM interface. We ran the experiments in a Parsytec PowerXplorer parallel machine. There are 16 nodes (each composed of a PowerPC 601 processor and a Transputer T805 and a local memory of 32 Mbytes) interconnected as a bidimensional grid.

To test the performance of the algorithm, we used instances of the problem that are unfavorable in the sense that they present many hits or occurrences, thus requiring a large amount of data exchanges. We obtained speedups by comparing

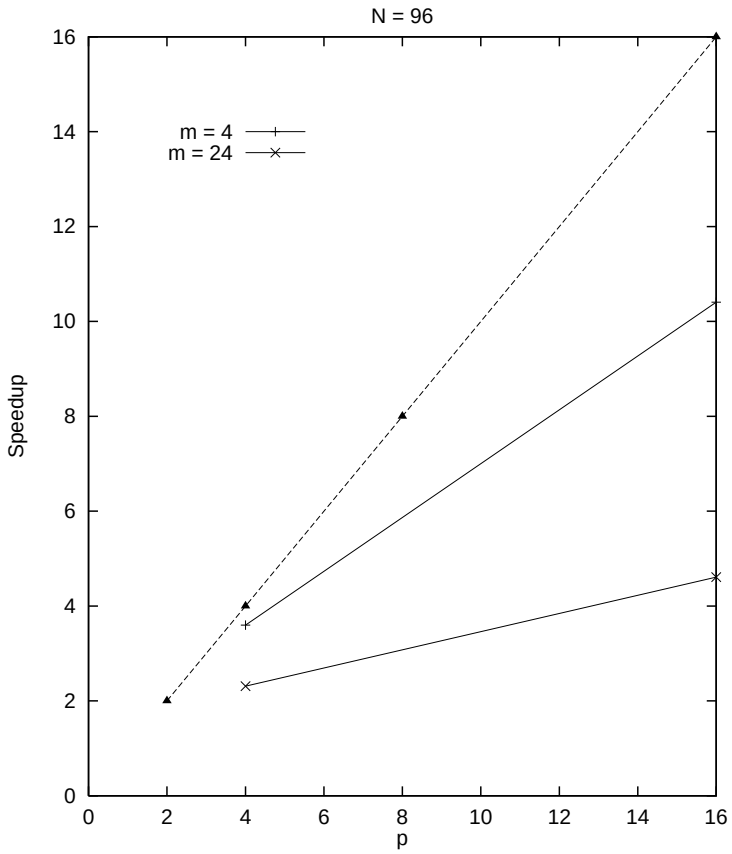

Fig. 4: Speedup for 4 and 16 processors with $N = 96$, $m = 4$ and $m = 24$. The dotted line represents the optimal speedup.

the parallel times with the sequential time. We used a $N = 96 \times 96$ text matrix and $m = 4 \times 4$ and $m = 24 \times 24$ pattern matrices. These matrices have symbols a and b. Each row is constituted by these symbols alternating as and bs and two successive rows do not begin with the same symbol.

In Figure 4 observe a very significant near-optimal speedup, for 4 processors and a 4×4 pattern. The speedup becomes smaller for 16 processors. For a 24×24 pattern, we obtain less significant speedups. It should be noted that the size of the input text matrix (96×96) is due to the memory limitation of the particular machine used. Ideally we should use larger input sizes. Nevertheless the results obtained indicate the efficiency of the proposed algorithm.

8 Conclusion

Amir *et al* [1] presented a sequential algorithm for bidimensional pattern matching with scaling. Their algorithm obtains all the occurrences for all possible integer scales, in sequential linear time in the size of the input. This problem appears

in many image applications where a scaled pattern is searched in a large image. To describe the algorithm, the authors give also algorithms for the unidimensional pattern matching with scaling and bidimensional pattern matching without scaling.

We propose a parallel algorithm for this problem under the CGM model, using p processors. We consider an $N \times N$ text matrix and an $m \times m$ pattern matrix. To solve the pattern matching problem with scaling, the proposed algorithm uses $O(N^2/p)$ local computing time and $O(N^2/p)$ memory, and requires only one communication round in which at most $O(Nm/\sqrt{p})$ data are exchanged, with $m \leq N/\sqrt{p}$.

To our knowledge this is the first parallel algorithm for distributed memory parallel machines known in the literature.

To design this algorithm, we consider boundary regions that correspond to submatrices of the text matrix, stored locally in each processor. The first idea that would work is to send the boundary regions to the respective neighbors and apply the sequential algorithm to the extended region (formed by local data plus the received boundary regions). This simple solution, however, would impose a fixed amount of data to be transmitted that is, in most cases, much more than the amount of occurrences. In our algorithm, neighbor processors receive only those data positions which are useful to determine the occurrences. This reduces the amount of data exchanged without increasing the local computing time excessively. Since the bottleneck is communication, we prefer a smaller amount of data exchanged. These considerations turn out to be crucial in obtaining an efficient algorithm, in theory and in practice, as shown with our experimental results.

Our algorithm is related to the common longest substring problem, which is used in the string comparisons. For this we used the suffix tree and LCA.

The parallel algorithm has been implemented in C, using the PVM interface and executed on a *Parsytec PowerXplorer* parallel computer. The results show very good performance for small patterns and a more discrete performance when using larger patterns. This is caused mainly by the quantity of data exchanged and also because we considered the worst case (by maximizing the amount of occurrences and of data transmitted) in choosing the text and pattern matrices for testing. Also we notice that the particular machine utilized does not excel in communication since it uses basically coaxial cables and Ethernet-like speed for interprocessor communication. Much better performances and speedups can be expected in practice where much less data is exchanged.

Acknowledgements

The authors would like to thank the referees for their comments. This research has been supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Proc. No. 98/06138-2, CNPq Proc. No. 52.3778/96-1 and 46.1230/00-3, and CNPq/NSF Collaborative Research Program Proc. No. 68.0037/99-3.

References

1. A. Amir, G. M. Landau and U. Vishkin, Efficient Pattern Matching with Scaling, *Journal do Algorithms* **13** (1992) 2–32.

2. F. Dehne, A. Fabri and C. Kenyon, Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time, in *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, 1994, 586–593.
3. F. Dehne, A. Fabri and A. Rau-Chaplin, Scalable Parallel Computational Geometry for Coarse Grained Multicomputers, in *Proc. ACM 9th Annual Computational Geometry*, 1993, 298–307.
4. D. E. Knuth, J. H. Morris Jr. and V. R. Pratt, Fast Pattern Matching in Strings, *SIAM J. Comput.* **6** (1977) 323–350.
5. P. Ferragina and F. Luccio, String Search in Coarse-Grained Parallel Computers, *Algorithmica* **24** (1999) 177–194.
6. L. Boxer, R. Miller and A. Rau-Chaplin, Scalable Parallel Algorithms for Geometric Pattern Recognition, *Journal of Parallel and Distributed Computing* **58** (1999) 466–486.
7. A. Amir, A. Butman and M. Lewenstein, Real Scaled Matching, *Information Processing Letters* **70** (1999) 185–190.
8. H. Mongelli and S. W. Song, Parallel Range Minima for Coarse Grained Multicomputers, *International Journal of Foundations of Computer Science* **10** (1999) 375–389.
9. H. Mongelli and S. W. Song, A Range Minima Parallel Algorithm for Coarse Grained Multicomputers, in *Workshop on Solving Irregularly Structured Problems in Parallel*, *Lecture Notes in Computer Science* **1586** (1999) 1075–1084.
10. H. Mongelli, *Algoritmos CGM para Busca Uni e Bidimensional de Padrões com e sem Escala*, Ph. D. Thesis, Inst. Matemática e Estatística - Universidade de São Paulo, 2000.
11. L. G. Valiant, A Bridging model for parallel computation, *Comm. of the ACM* **33** (1990) 103–111.
12. E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro and S. W. Song, Efficient Parallel Graph algorithms for Coarse Grained Multicomputers and BSP, in *Proc. 24th International Colloquium on Automata, Languages, and Programming (ICALP'97)*, *Lecture Notes in Computer Science* **1256** (1997) 390–400.
13. F. Dehne and S. W. Song, Randomized parallel list ranking for distributed memory multiprocessors, in *Proc. Second Asian Computing Science Conference (ASIAN'96)*, 1996, 1–10.
14. J. JáJá, *An Introduction to Parallel Algorithms* (Addison-Wesley, Reading, 1992).
15. J. H. Reif, *Synthesis of Parallel Algorithms* (Morgan Kaufmann Publishers, San Mateo, 1993).
16. , B. Schieber and U. Vishkin, On Finding Lowest Common Ancestors: simplification and par-allelization, *SIAM J. Comput.* **17** (1988) 1253–1262.
17. J. Vuillemin, A Unified Look at Data Structures, *Comm. of the ACM* **23** (1980) 229–239.
18. T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms* (The MIT Press, 1990).
19. E. M. Reingold, R. J. Urban and D. Gries, KMP String Matching revisited, *Information Processing Letters* **64** (1997) 217–223.
20. E. M. McCreight, A Space-Economical Suffix tree construction algorithm, *Journal of ACM* **23** (1976) 262–272.