Efficient Two-Dimensional Parallel Pattern Matching with Scaling

H. Mongelli
Universidade Federal do Mato Grosso do Sul
Depto. de Computação e Estatística
Campo Grande, MS, 79069, Brazil
email: mongelli at dct. ufms. br

S. W. Song
Universidade de São Paulo
Inst. de Matemática e Estatística
Depto. de Ciência da Computação
São Paulo, SP - 05508-900 - Brazil
email: song at ime. usp. br

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. In this paper we assume 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 on a parallel machine and the experimental results obtained were very promising and showed significant speedups.

KEY WORDS
Parallel algorithms, coarse-grained multicomputer, pattern matching with scaling, string matching with scaling

1. Introduction

In this paper we present an efficient parallel algorithm for the bidimensional pattern matching problem with scaling. Let T be the *text* matrix and P the *pattern* matrix. T and P are respectively, $N \times N$ and $m \times m$ matrices. 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). 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].

The CGM (Coarse-Grained Multicomputer) model [2] considers a problem with input size n, and solves it with p processors each with $O(n/p)$ local memory. Usually we have $n \gg p$. A CGM algorithm is described as a number of alternating computing rounds and communication rounds. 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 [3], to determine all the occurrences for each scale k in linear time on the input size. Ferragina and Luccio [4] consider string searching (without scaling) in the BSP model and coarse-grained computing models.

To the best of our knowledge no coarse-grained algorithms are known for bidimensional pattern matching with or without scaling. The proposed algorithm is based on [1] and uses the range minima algorithm of Mongelli and Song [5]. It constitutes the main result of [6]. We have implemented the proposed algorithm on a parallel machine and obtained very promising results. For small patterns we get almost optimum speedups.

2. The Computing Model and Some Definitions

The CGM model [2] is similar to the BSP (Bulk-Synchronous Parallel) model[7]. However, it uses only two parameters: the input size n and the number of processors p, each with a local memory of size $O(n/p)$. Usually we have $n \gg p$. 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. 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. The communication cost is modeled by the number of communication phases. This model is particularly suitable in current parallel machines in which the global computing speed is considerably larger than the global communication speed.

The proposed CGM algorithm uses some definitions that we summarize as follows.

Definition 1 *Factorized Representation of a String. 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 $\sigma_1'^{r_1}\sigma_2'^{r_2}\cdots\sigma_{\hat{n}}'^{r_{\hat{n}}}$, $\hat{n} \leq n$, such that:*

- $\sigma_i' \neq \sigma_{i+1}'$, for $1 \leq i < \hat{n}$; and
- 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 $\sigma_i'^{r_i}$.)

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.

Definition 3 *The* lowest common ancestor *[8] 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 *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.*

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 *[8, 5] consists of preprocessing vector A such that queries $MIN(i, j)$, for any $1 \leq i \leq j \leq n$, can be answered in constant time.*

Definition 6 *Let $A_{0,n-1} = (a_0, a_1, \ldots, a_{n-1})$ be an array of n distinct real numbers, the* Cartesian tree *[9] 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 *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 $k = 1, \ldots \lfloor N/m \rfloor$. The text and pattern matrices have therefore N^2 and m^2 elements, respectively. Figure 1 shows an example of pattern matching with scaling.

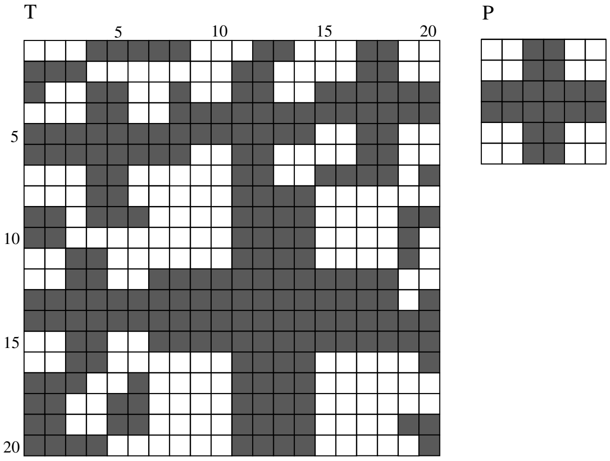

Figure 1. Example of text T and pattern P with occurrences of P in T in positions $[1, 15]$, $[2, 9]$, $[3, 2]$, $[11, 1]$ with scale 1 and in position $[8, 7]$ with scale 2.

Amir *et al* [1] present a linear sequential algorithm, based on the KMP-algorithm [3, 10], of time complexity $O(N^2)$. 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 proofs of the time complexity and correctness of the algorithm can be seen in [1, 6]. 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 has as input the factorized representation FT of the text matrix T and the factorized representation FP of the pattern matrix P. It has the following main steps. These steps will be described in the next sections.

CGM Algorithm:

1. Preprocessing of input data.
2. Pattern analysis.
3. Text analysis.

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. 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. 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.

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. We 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)*.

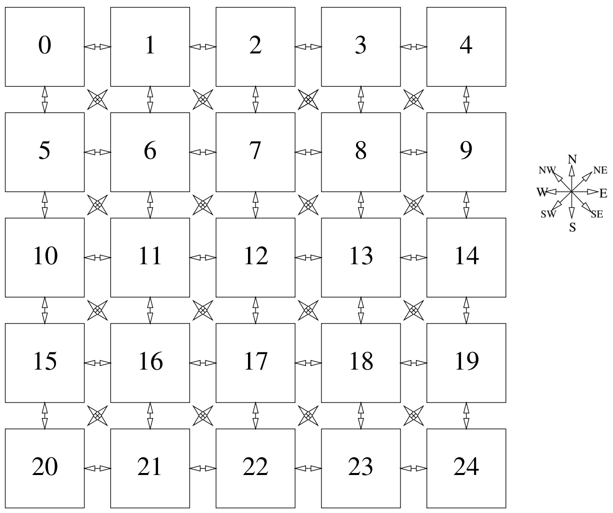

Figure 2. Neighborhood relation between processors considering data distributed in $p = 25$ processors.

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. 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.

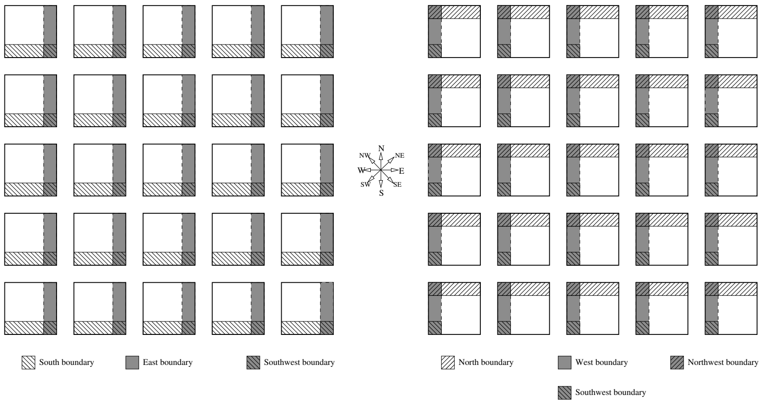

Figure 3. Representing the boundaries South(S), East(E) and Southeast (SE) and boundaries North(N), West(W), Northwest(NW) and Southwest(SW).

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 boundaries.

5. The proposed CGM Algorithm

Each processor receives an $n \times n$ submatrix S, where $n = N/\sqrt{p}$, as well as the entire $m \times m$ pattern P. 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

The preprocessing consists of the construction of the C string and the suffix tree and some other data strucutures such as the Cartesian trees. Range minima is used in this step.

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 [11] 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, 6].

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 $\hat{F}R$ be the subsequence of FR that starts at the second element and ends at the next to last element. That is, $\hat{F}R = (FP_{l_2}, r_2) \ldots (FP_{l_{\alpha-1}}, r_{\alpha-1})$.

We compute a prefix function for the string $\hat{F}R$, using the suffix tree ST. It consists of calculating for each element (FP_{l_i}, r_i), with $2 \leq i < \alpha$, of $\hat{F}R$, the length of the largest prefix of $\hat{F}R$ 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.

5.4 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.

6. Implementation Results

The algorithm has been implemented in C, using the PVM interface, on the Parsytec PowerXplorer parallel machine of 16 nodes.

We used instances of the problem that are very unfavorable, by maximizing the number of hits (occurrences) and requiring a large amount of data exchanges. We used a

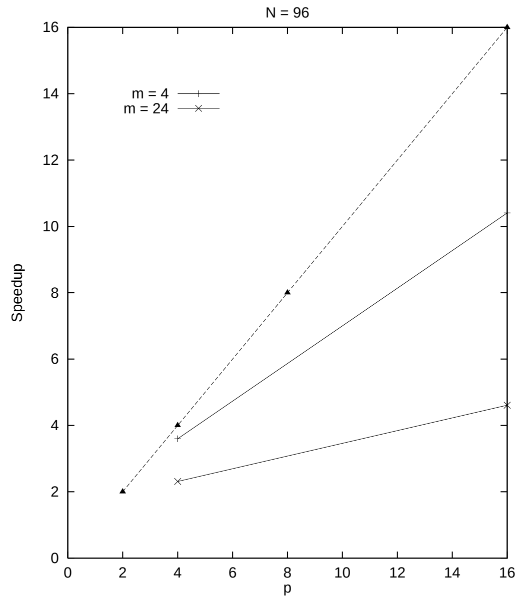

Figure 4. Speedup for 4 and 16 processors with $N = 96$, $m = 4$ and $m = 24$.

$N = 96 \times 96$ 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.

7. Conclusion

We propose a parallel algorithm for the two-dimensional patter matching problem with scaling 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.

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 an 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.

The experimental 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.

References

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

[2] F. Dehne, A. Fabri and A. Rau-Chaplin, Scalable Parallel Computational Geometry for Coarse Grained Multicomputers, *Proceedings of the ACM 9th Annual Computational Geometry*, 1993, 298–307.

[3] D. E. Knuth, J. H. Morris Jr. and V. R. Pratt, Fast Pattern Matching in Strings, *SIAM J. Comput.*, 6, 1977, 323–350.

[4] P. Ferragina and F. Luccio, String Search in Coarse-Grained Parallel Computers, *Journal of Algorithms*, 24(2), 1999, 177–194.

[5] H. Mongelli and S. W. Song, Parallel Range Minima for Coarse Grained Multicomputers, *International Journal of Foundations of Computer Science*, 10(4), 1999, 375–389.

[6] H. Mongelli, *Algoritmos CGM para Busca Uni e Bidimensional de Padrões com e sem Escala*, PhD. Thesis, Institute of Mathematics and Statistics, University of São Paulo, 2000.

[7] L. G. Valiant, A Bridging Model for Parallel Computation, *Comm. of the ACM*, 33, 1990, 103–111.

[8] J. JáJá, *An Introduction to Parallel Algorithms*, (Reading, Massachusetts, Addison-Wesley Publishing Company, 1992).

[9] J. Vuillemin, A Unified Look at Data Structures, *Comm. of the ACM*, 23, 1980, 229–239.

[10] E. M. Reingold, R. J. Urban and D. Gries, KMP String Matching Revisited, *Information Processing Letters*, 64(5), 1997, 217–223.

[11] E. M. McCreight, A Space-Economical Suffix Tree Construction Algorithm, *Journal of ACM*, 23(2), 1976, 262–272.