Parallel String Matching with Scaling*

H. Mongelli
Universidade Federal do Mato Grosso do Sul
Depto. de Computação e Estatística
Campo Grande, MS, 79069, Brazil

S. W. Song
Universidade de São Paulo
Inst. de Matemática e Estatística
Depto. 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 strings, the matching problem is said to be unidimensional. We examine a variation of this problem where we allow that the pattern can be scaled. We propose new parallel algorithms, under the Coarse Grained Multicomputer model, that require linear local computing time and only one communication round. The proposed algorithms were implemented on a Parsytec PowerXplorer parallel computer. The experimental results obtained are promising and show significant speedups.*

Keywords: Parallel algorithms, coarse-grained multicomputer, BSP, string matching with scaling

1 Introduction

We present new parallel algorithms, under the CGM (*Coarse Grained Multicomputers*) model, for the problem of searching unidimensional patterns in a text string without and with scaling. The same ideas can also be used for the bidimensional case as shown in [12].

Unidimensional pattern matching, which we refer to as *string matching*, is an important problem encountered in text processing, speech recognition, computer vision and molecular biology [8]. Bidimensional pattern matching without scaling is found in multimedia libraries, computer vision and heterogeneous data searching [1]. The case with scaling is found in image processing applications where the sources for capturing image can be at different distances [2].

Consider a text string $T = \tau_1 \dots \tau_n$, of length n, and a pattern string $P = \rho_1 \dots \rho_m$, of length m, formed by symbols of a given alphabet Σ. We say that the pattern P *occurs* in a position s, $1 \leq s \leq n - m + 1$, of T if $\tau_s = \rho_1, \tau_{s+1} = \rho_2, \dots, \tau_{s+m-1} = \rho_m$. Furthermore, we say that there is an occurrence of P at position s of T. The problem of string matching consists of determining all the positions s in T, $1 \leq s \leq n - m + 1$, where the pattern P occurs.

The most well-known sequential algorithm for string matching is the KMP (Knuth-Morris-Pratt) algorithm [10], that runs in linear time on the size of the input ($O(n + m)$). This time is obtained by reducing the number of comparisons between positions of the pattern and text, through previously computed information. These information enable us to discard beforehand those positions where the pattern cannot occur. Other sequential algorithms for this problem are presented in [3, 7, 9, 11, 13].

In string matching with scaling, we are interested in determining all the occurrences of the pattern in the text, scaled with any integer k, with $1 \leq k \leq \lfloor n/m \rfloor$. In other words, suppose the input of a pattern $P = \rho_1 \cdots \rho_m$ and a text $T = \tau_1 \cdots \tau_n$ with $n > m$, we

*Research 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 and CAPES.

want to determine all the positions of T where an occurrence of P scaled with k (called a k-occurrence) starts, for all $k = 1, \ldots, \lfloor n/m \rfloor$. The k-occurrences with $k = 1$ are those occurrences in normal string matching. Amir *et al* [2] present a linear time sequential algorithm that determines all the k-occurrences. It uses the KMP algorithm in the so-called *factored representations* of the text and pattern strings.

Let $S = \sigma_1\sigma_2\cdots\sigma_n$ be a string on an alphabet Σ. The *factored representation* of the 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 $1 \leq i < \hat{n}$; S can be described as a concatenation of the symbol σ_1' repeated r_1 times, the symbol σ_2' repeated r_2 times, $\ldots$, and the symbol $\sigma_{\hat{n}}'$ repeated $r_{\hat{n}}$ times. Denote by $S^\Sigma = \sigma_1'\sigma_2'\cdots\sigma_{\hat{n}}'$ the *part of symbols* of S and by $S^\#$ the array of integer numbers $r_1, r_2, \ldots, r_{\hat{n}}$, the *part of exponents* of S. We can also construct the *quotient strings* $T' = \tau_2/\tau_1, \tau_3/\tau_2, \ldots, \tau_{\hat{n}}/\tau_{\hat{n}-1}$ of $\hat{n}-1$ elements and $P' = \rho_3/\rho_2, \rho_4/\rho_3, \ldots, \rho_{\hat{m}-1}/\rho_{\hat{m}-2}$ of $\hat{m}-3$ elements.

The CGM model was proposed by Dehne *et al* [6, 5, 4] as a simpler model than the BSP model of Valiant [14]. For a problem of size n, the CGM model consists of a set of p processors, each with $O(n/p)$ local memory, that communicate through an interconnection network that allow point-to-point communication among processors.

An algorithm under the CGM model, or simply a CGM algorithm, for a problem of size n and using p processors is described as supersteps composed of local computing rounds and global communication rounds. In a computing round, data are manipulated locally in each processor. In a communication round, we allow the exchange of information among processors, where each processor can send and receive a total of $O(n/p)$ data. In the CGM model, the goal is to minimize the number of communication rounds and the quantity of local computation.

For both problems, without and with scaling, we consider p processors, a text of length N and a pattern of length m, with $m \leq N/p$. We present CGM algorithms requiring computation time of $O(N/p)$, memory of size $O(N/p)$ and only one communication round in which at most $O(m)$ data are exchanged.

We implemented these algorithms on a real parallel machine and obtained very promising results. The good performance is due to the effort of minimizing the number of communication rounds, the quantity of data transmitted, and the amount of local computing. We show that the speedups obtained are close to optimal for small patterns.

2 CGM Algorithms for String Matching

Consider a text string of length N and a pattern string of length m, $m \leq N/p$. We present CGM algorithms (for the cases without and with scaling) that execute in $O(N/p)$ local computing time, and use one communication communication round during which we exchange $O(m)$ data. Let us define $n = N/p$.

Two processors numbered i and j are said to be *neighbor processors* if their processor numbers differ by one. Given a processor i, its *left neighbor processor* is processor $i-1$, for $1 \leq i \leq p-1$, and its *right neighbor processor* is processor $i+1$, for $0 \leq i \leq p-2$.

The text will be initially divided into p *text subarrays* S, also known as *text segments*, each one of size $n = N/p$, that will be distributed in the processors such that the first and the last segments are respectively stored in processors 0 and $p-1$. Furthermore, consecutive segments are stored in processors with successive processor numbers, in a contiguous way. Thus, two neighbor processors will contain adjacent text segments. The pattern will be stored entirely in each processor. Figure 1 illustrates the distribution of the text in the processors.

For the case with scaling, we determine locally the arrays S^Σ and $S^\#$, P^Σ and $P^\#$, and the arrays of quotients S' and P'.

As the pattern must be compared with the text as if this text were contiguously stored, we must consider the case in which the pattern is compared with text subarrays stored in two

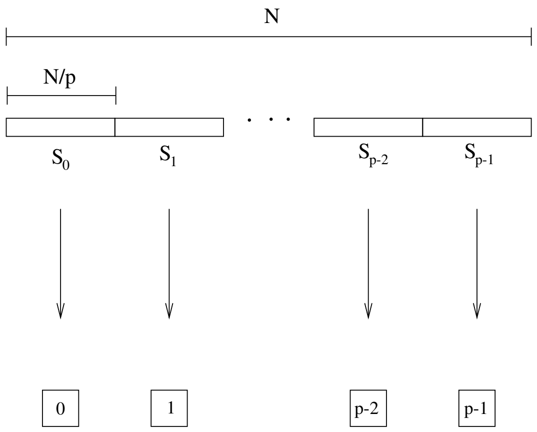

Figure 1: Distribution of a Text of Length N in the p Processors.

neighbor processors. In this case, we say that the pattern is being compared at the *boundary* of these text subarrays. In this way, the boundaries are constituted by prefixes and suffixes of length $m-1$ of the subarrays in each processor, except the processors that store the subarrays of the extremities, that present only one boundary. The boundary corresponding to the prefix of a text subarray will be denominated *left boundary* and, analogously, the boundary corresponding to the suffix of a text subarray will be called *right boundary*. For the case with scaling, the boundaries will be determined em relation to the arrays S^{Σ} and P^{Σ}.

The output of the algorithm without scaling is a list R of positions of occurrences of P in T, stored in each processor. For the case with scaling, the list includes the value of k.

In both algorithms, the first step consists of determining sequentially the positions of occurrence (respectively, k-occurrence) of the pattern in the locally stored text segment. For the case without scaling, we use the KMP algorithm [10] and for the case with scaling, the algorithm of Amir *et al* [2].

The next step consists of determining, for each segment, the possibility of an occurrence (respectively, k-occurrence) of the pattern at the right boundary. Each possible occurrence is stored in a list called C_D, that contains the so-called *candidate positions*. To this end we use the slightly adapted sequential algorithms.

Each processor obtains a list of positions that will serve to confirm if the candidate positions are really positions of occurrence (respectively, k-occurrence) of the pattern in the text. These positions will be denominated *confirmation positions* e will be stored in a list called C_E. In both cases, the idea consists of applying the algorithm at the left boundaries in a symmetric way as what has been done for the obtention of the candidate positions.

We then send the confirmation positions to the left neighbors, which will use them to validate or not the candidate positions as occurrences (respectively, k-occurrences). There are some peculiar characteristics in the case with scaling, because of the way the sequential algorithm works, that implies the sending of the two initial characters and exponents, in factored form, together with the positions in C_E and the respective scales. The details can be found in [12].

With the information thus received, each processor can validate or not the candidate positions. This is done by traversing the local lists C_D and received lists C_E as if they were interleaved and observe the sum of the distances of the candidate and confirmation positions relative, respectively, to the end and beginning of the text segments.

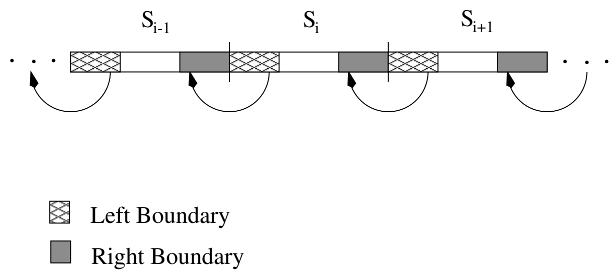

Figure 2: Boundaries and Communication between Neighbors.

The obtention of the list R in the first step takes $O(n/p)$ time. The determination of the lists C_D and C_E in each processor takes $O(m)$ time. The confirmation of the candidate positions takes $O(|C_D| + |C_E|) = O(m)$ time.

2.1 Implementation Results

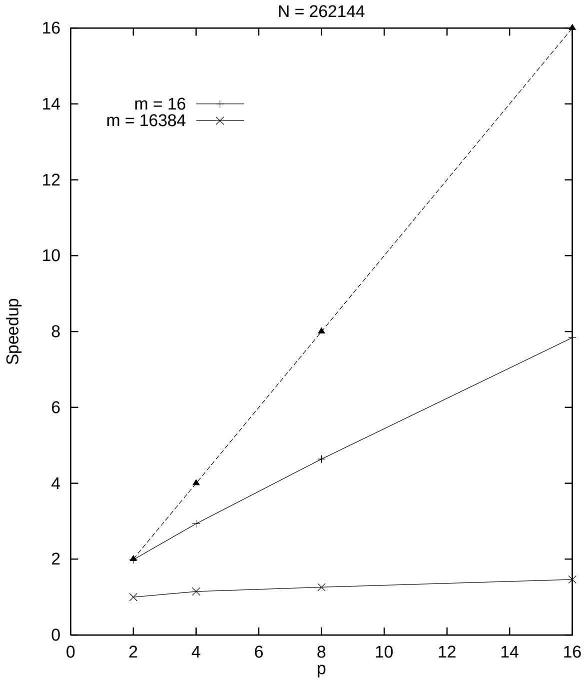

Figure 3: Speedups - Case without Scaling.

For the case without scale, we use the same symbol a to form the text and pattern strings. This is the worst possible case because it is the most time consuming. The CGM algorithm is executed for two examples, and in each we use a text string of length $N = 2^{18}$ and two patterns of lengths $m = 2^4$ and $m = 2^{14}$. The occurrences appear in all the positions of the text string except those positions starting at the last $m - 1$ positions.

In Figure 3 we show the speedups for two different pattern lengths. The dash line with small triangles corresponds to the optimal speedup. We notice a small speedup for the case $m = 2^{14}$. As the sequential algorithm is very fast, the communication ends up dominating the total computing time. This fact is confirmed by the better performance for the pattern of length $m = 2^4$.

For the case with scaling, we use a text string of length $N = 2^{18}$ and a pattern strings of length $m = 2^4$ and $m = 2^{14}$. These strings are made up by alternating the symbols a and b starting with a symbol a. In this case, the

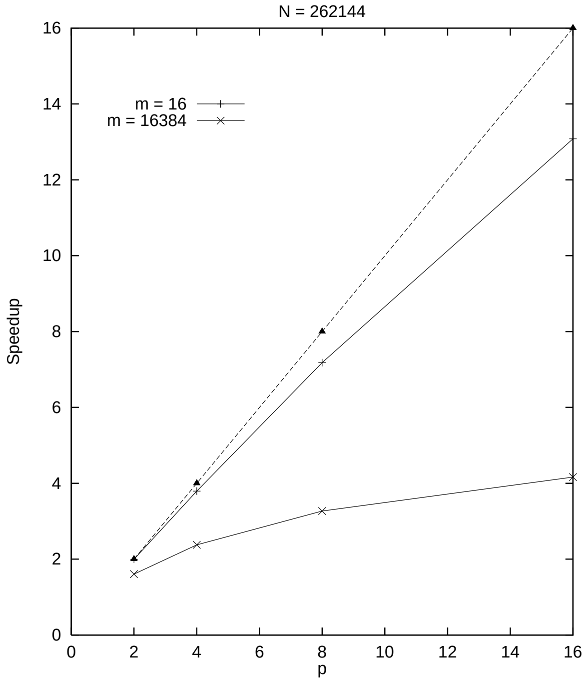

Figure 4: Speedups - Case with Scaling.

occurrences with scale 1 appear in $O(m/2)$ positions, alternately. There do not exist occurrences with other scales. We consider this case because it presents the largest number of occurrences with $\hat{m} > 3$. Figure 4 shows the speedups obtained. The result is very good for $m = 2^4$. It is more modest in the case $m = 2^{14}$ because of the dominance of the communication time on the local computing time.

3 Conclusion

We propose new CGM algorithms for string matching without and with scaling. We have implemented these algorithms and the experimental results are very promising showing the efficiency and scalability of the proposed algorithms, with speedups close to optimal for small patterns.

References

[1] A. Amir, A. Butman, and M. Lewenstein. Real scaled matching. *Information Pro-*

cessing Letters, 70(4):185–190, 1999.

[2] A. Amir, G. M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13:2–32, 1992.

[3] L. Colussi. Fastest pattern matching in strings. *Journal of Algorithms*, 16:163–189, 1994.

[4] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A. Kokhar. A randomized parallel $3D$ convex hull algorithm for coarse grained multicomputers. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)*, pages 27–33, 1995.

[5] F. Dehne, A. Fabri, and C. Kenyon. Scalable and architecture independent parallel geometric algorithms with high probability optimal time. In *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, pages 586–593, 1994.

[6] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. In *Proceedings of the ACM 9th Annual Computational Geometry*, pages 298–307, 1993.

[7] P. Ferragina and R. Grossi. Optimal on-line search and sublinear time update in string search. *SIAM J. Comput.*, 27(3):713–736, 1998.

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

[9] R. Karp and M. Rabin. Efficient randomized pattern-matching algorithms. Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University, 1981.

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

[11] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

[12] H. Mongelli. *CGM Algorithms for Uni and Bidimensional Pattern Matching with Scaling (in Portuguese).* Phd. thesis, Inst. Matemática e Estatística - Universidade de São Paulo, São Paulo, April 2000.

[13] E. M. Reingold, R. J. Urban, and D. Gries. KMP string matching revisited. *Information Processing Letters*, 64(5):217–223, 1997.

[14] L. G. Valiant. A bridging model for parallel computation. *Comm. of the ACM*, 33:103–111, 1990.