String Search in Coarse-Grained Parallel Computers

P. Ferragina 2 and F. Luccio 2

Abstract. Given a text string \( T[1, n] \), the multistring search problem consists of determining which of \( k \) pattern strings \( \{X_1[1, m], X_2[1, m], \ldots, X_k[1, m]\} \), provided on-line, occur in \( T \). We study this problem in the BSP model [27], and then extend our analysis to other coarse-grained parallel computational models. We refer to the relevant and difficult case of long patterns, that is \( m > p \), where \( p \) is the number of available processors, and provide an optimal result with respect to both computation and communication times, attaining a constant number of supersteps. We then study single string search (i.e., \( k = 1 \)), and show how the multistring search algorithm can be employed to speed up the process and balance the communication cost. The proposed solution takes a constant number of supersteps, and achieves optimal communication time if the string to be searched is longer than \( p^2 \). Our results are based on the distribution of a proper data structure among the \( p \) processors, to reduce and balance the communication cost. We also indicate how short patterns can be efficiently dealt with, through a completely different algorithmic approach.

Key Words. String matching, Distributed data structures, BSP model, Parallel computing, Computational complexity.

1. Introduction. The availability of very large text databases makes the design of efficient methods for string processing more and more critical [2]. Important applications on digital libraries [15], biological and textual databases [26], make use of very large text collections requiring specialized nontrivial search operations. Parallelism offers a strong hope for meeting such a challenge. A large amount of research has been recently directed to designing data structures and algorithms for string processing that exploit the parallelism inherent in the external storage devices (multidisks) [4], [8], [12], [13]. A natural and related question is how computing with \( p \) processors can speed up the search operations. Several techniques for PRAM fast string search have been proposed (e.g., see [1], [3], [16], [19], and [25]). As known, however, in this model of computation the number of processors is treated as an unbounded resource, and the cost of communication is not taken into account. In this paper we study the worst-case complexity of some string searching problems in a coarse-grained parallel environment, by developing simple distributed data structures that reduce and balance the communication cost.

The multistring search problem considered here can be regarded as the problem of performing a number of string search processes on a given set of text strings. For simplicity, we can restrict our attention to just one text string that is obtained by concatenating all available texts, separated by a special character that does not occur elsewhere. In formal terms, a multistring search on a text \( T[1, n] \) consists of determining, for a set of \( k \) pattern strings \( \{X_1[1, m], X_2[1, m], \ldots, X_k[1, m]\} \) provided on-line, which \( X_j \)'s

1 This research was supported in part by NATO Project No. CRG971467. Part of this work has been presented as a preliminary conference version in [13].

2 Dipartimento di Informatica, Università di Pisa, Pisa, Italy. ferragin,luccio@di.unipi.it.

Received June 1, 1997; revised March 10, 1998. Communicated by F. Dehne.
occurs in $T$. Since the text $T$ is given in advance, the goal is to preprocess properly it in such a way that the $k$ queried patterns can be searched into $T$ with a complexity that depends on their total length $km$, and is as independent as possible of the text length $n$. This problem naturally generalizes classical on-line string matching [24] to a set of pattern strings. Although for simplicity we refer to a boolean query that establishes “if a pattern occurs” and not “where it occurs,” our approach can be immediately extended to the more general “reporting” case. Furthermore, our assumption that all the pattern strings have the same length $m$ can be easily relaxed.

To study the communication and computation performance of our solutions we first adopt the Bulk Synchronous Parallel (BSP) model [27], and then consider some of its recent extensions. A BSP($p, g, L$) computer consists of $p$ processors, each provided with a local memory (of size $O(n/p)$) and communicating through a network of bandwidth $g$ and latency $L$. The computation of this machine is organized as a sequence of supersteps. In a superstep, the processors operate independently performing local computation and generating a number of point-to-point messages. At the end of the superstep, the messages are delivered to their destinations and a global barrier synchronization is enforced. If each processor performs at most $w$ local operations and sends/receives at most $h$ messages (this is denoted as an $h$-relation), the superstep requires $w + gh + L$ time, where $\max(w, L)$ is the computation time and $\max(gh, L)$ is the communication time. Some optimality criteria proposed in [17] characterize the possible speed-up on real machines. In this paper we say that a BSP algorithm is $c$-optimal if its speed-up is close to $p/c$ for both the communication and computation time.

Some variations of the BSP model have been subsequently introduced (e.g., see [5], [10], [9], and [20]) to encourage the use of spatial locality, and to measure I/O cost in addition to communication cost. These concepts can be combined in several ways, to depict different families of physical architectures. We consider here the models BSP* of [5] and EM-BSP* of [9], as paradigms respectively based on spatial locality and I/O complexity. In BSP*, blockwise communication is accounted for by a new parameter $b$ which measures the packet size that yields best throughput. In EM-BSP* each processor has a memory of size $M$, and $D$ disk drives that can be accessed concurrently. If $B$ denotes the disk block size, then a processor can transfer $D \times B$ data from disks to local memory in a single I/O operation, at a cost $G$. It is assumed that $M \geq DB$. As a limit EM-BSP* can contain one processor, thereby resembling the one-processor version of the Parallel Disk Model introduced in [28]. In this case parallelism occurs at disk (I/O) level.

In this paper we study the multistring search problem in the BSP($p, g, L$) model, assuming that the patterns are long, that is $m \geq p$, and provide an optimal result with respect to both the computation and communication times (the number of supersteps is constant and thus also optimal). Note that the condition $m \geq p$ is fairly natural in practice. A typical case is the indexing of WEB servers which usually consist of few powerful commodity workstations connected by a high-speed local network. We then consider the problem of single string search (i.e., $k = 1$), and show how to exploit the previous algorithms for multisearch in order to speed up the process and balance the communication cost. In this way we achieve optimal communication time if the string to be searched is sufficiently long (i.e., longer than $p^2$), using a constant (optimal) number of supersteps. The study conducted on BSP is then extended to BSP*, and to EM-BSP* with one processor, through simulation between models. This also leads to