On the Multisearching Problem for Hypercubes*

Mikhail J. Atallah† Andreas Fabri‡

Abstract

In this paper we give improved bounds for the multisearch problem on a hypercube. This is a parallel search problem where the elements in the structure $S$ to be searched are totally ordered, but where it is not possible to compare in constant time any two given queries $q$ and $q'$. This problem is fundamental in computational geometry, for example it models planar point location in a slab. More precisely, we are given on a $n$-processor hypercube a sorted $n$-element sequence $S$, and a set $Q$ of $n$ queries, and we need to find for each query $q \in Q$ its location in the sorted $S$. Note that one cannot solve this problem by sorting $S \cup Q$, because every comparison-based parallel sorting algorithm needs to compare a pair $q, q' \in Q$ in constant time. We present an improved algorithm for the multisearch problem, one that takes $O(\log n (\log \log n)^3)$ time on a $n$-processor hypercube. This essentially replaces a logarithmic factor in the time complexities of previous schemes by a $(\log \log n)^3$ factor. The hypercube model for which we claim our bounds is the standard one, SIMD, with $O(1)$ memory registers per processor, and with one-port communication. Each register can store $O(\log n)$ bits, so that a processor knows its ID.

1 Introduction

Consider the situation depicted in Figure 1: We have a horizontal slab partitioned by a set $S$ of $n$ nonintersecting segments. For a set $Q$ of $n$ points, we need to determine for each point which region of the slab it belongs to. Both the segments and the points are initially stored in a $n$ processor hypercube.

This problem would be trivial, if the partitioning segments were vertical, but the fact that they are slanted makes it impossible to solve the problem by (e.g.) simply mergesorting $S \cup Q$ according to $x$-coordinates. The method we give for solving this multisearch problem works for more general versions of this problem: The basic assumption is that any pair $x, y$ in a processor can be compared in constant time if $x \in S \cup Q$ and $y \in S$, but not so if both $x$ and $y$ are

---

*This work was supported in part by the National Science Foundation under Grant CCR-9202807, and by the ESPRIT Basic Research Action Nr. 7141 (ALCOM II).
†Department of Computer Science, Purdue University, West Lafayette, IN 47906, USA. Email: mj a@cs.purdue.edu
‡INRIA, BP 93, 06902 Sophia-Antipolis Cedex, France. Email: fabri@sophia.inria.fr.
in Q. In [3, 5] \(O(\log^2 n)\) time algorithms for this problem are given. They are easy to implement and thus of practical interest, and the algorithm of the first paper was later generalized for doing fractional cascading on a hypercube [4]. A randomized \(O(\log n)\) expected time scheme for multisearching was given by Reif and Sen [8]. Since searching is related to sorting and there is a deterministic \(O(\log n \log \log n)\) time sorting algorithm [2], the question was open, if there exists an algorithm for the multisearch problem that runs faster than \(O(\log^2 n)\). This paper gives a step in the right direction, by presenting an algorithm with time complexity \(O(\log n (\log \log n)^3)\) for a \(n\) processor hypercube. Our result is more of theoretical than of practical interest, because it uses the sorting algorithm of [2] as a subroutine. However, any practical improvement to sorting would immediately make our algorithm more practical.

The paper is organized as follows. In Section 2 we review the definition of a hypercube interconnection network and some basic algorithms for this parallel machine. Then in Section 3 we sketch a very preliminary solution that is worse than the one we claim, but that serves as a "warmup" for the later improved algorithms. Section 4 gives an algorithm that is almost as good as what we claim, except that it requires each processor to have \(\Omega(\log \log n)\) memory registers (rather than \(O(1)\) registers). Section 5 gives the algorithm that achieves the bounds we claim. Section 6 concludes by discussing some implementation issues and details.

2 The Model of Computation

This section is a brief review of the model, and in particular of some operations on that model that we will make use of.

The hypercube model for which we claim our bounds is the standard one, with \(O(1)\) memory registers per processor, and with one-port communication. Each register can store \(O(\log n)\) bits, so that a processor knows its ID. Recall that a hypercube of dimension \(d\) consists of \(n = 2^d\) processors which are uniquely labeled with bitstrings of length \(d\). Two processors are connected along dimension \(i\), iff their labels differ in exactly the \(i^{th}\) bit. In this paper we are interested in SIMD (Single Instruction Multiple Data) machines, that is, all processors execute the same instruction simultaneously. An instruction is either an operation on data in the local memory, or a communication step with a processor adjacent along a particular dimension. An instruction takes time \(O(1)\).

We shall use as subroutines certain operations on sequences of size \(n\), with