The Buffer Tree:  
A New Technique for Optimal I/O-Algorithms *

(Extended Abstract)

Lars Arge
large@daimi.aau.dk
BRICS**  
Department of Computer Science, University of Aarhus, Denmark***

Abstract. In this paper we develop a technique for transforming an internal memory tree data structure into an external storage structure. We show how the technique can be used to develop a search-tree-like structure, a priority-queue, a (one-dimensional) range-tree and a segment-tree, and give examples of how these structures can be used to develop efficient I/O-algorithms. All our algorithms are either extremely simple or straightforward generalizations of known internal memory algorithms — given the developed external data structures.

1 Introduction

In the last few years, more and more attention has been given to Input/Output (I/O) complexity of existing algorithms and to the development of new I/O-efficient algorithms. This is due to the fact that communication between fast internal memory and slower external storage is the bottleneck in many large-scale computations. The significance of this bottleneck is increasing as internal computation gets faster, and especially as parallel computing gains popularity [16]. Currently, technological advances are increasing CPU speed at an annual rate of 40-60% while disk transfer rates are only increasing by 7-10% annually [18].

A lot of work has already been done on designing I/O-variants of known internal memory data structures (e.g. [10, 11, 12, 14, 15, 19]), but practically all these data structures are designed to be used in on-line settings, where queries should be answered immediately and within a good worst case number of I/O’s. This effectively means that using these structures to solve off-line problems yields non-optimal algorithms, because they are not able to take full advantage of the large internal memory. Therefore a number of researchers have developed

* This work was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II) and by Aarhus University Research Foundation.
** Acronym for Basic Research in Computer Science, a Center of the Danish National Research Foundation.
*** Part of this work was done at the Department of Computer Science, Duke University.
techniques and algorithms for solving large-scale off-line problems without using external storage data structures [1, 7, 8].

In this paper we develop external storage data structures that take advantage of the large main memory. This is done by only requiring good amortized performance of the operations on the structures, and by allowing search operations to be batched. The data structures developed can then be used in simple and effective algorithms for computational geometry and graph problems. As pointed out in [7] and [8] problems from these two areas arise in many large-scale computations, e.g. in object-oriented, deductive and spatial databases, VLSI design and simulation programs, geographic information systems, constraint logic programming, statistics, virtual reality systems, and computer graphics.

1.1 I/O-Model and Previous Results

We will be working in an I/O-model introduced by Aggarwal and Vitter [1]. The I/O-model has the following parameters:

- \( N = \# \) of elements in the problem instance
- \( M = \# \) of elements that can fit into main memory
- \( B = \# \) of elements per block

An I/O-operation in the model is a swap of \( B \) elements from internal memory with \( B \) consecutive elements from external storage. The measure of performance we consider is the number of such I/Os needed to solve a given problem. Internal computation is for free. The model captures the essential parameters of many of the I/O-systems in use today.\(^4\)

In [22] the I/O-model is extended with a parameter \( D \). Here the secondary storage is partitioned into \( D \) distinct disk drives, and if no two blocks come from the same disk, \( D \) blocks can be transferred per I/O. Furthermore, the model can be extended such that we have more than one internal processor (see e.g. [13]), and a number of authors have considered further extended models, with so-called multilevel hierarchical memories (see e.g. [13] or [21]), which aim to capture the fact that large-scale computer systems contain many levels of memory.

Early work on I/O-algorithms concentrated on algorithms for sorting and permutation-related problems in the single disk model [1] as well as in the extended versions of the I/O-model [13, 21, 22]. External sorting requires \( \Theta(n \log_m n) \) I/Os,\(^5\) which is the external storage equivalent of the well-known \( \Theta(N \log N) \) time bound for sorting in internal memory. Note that this means that \( O\left(\frac{\log_m n}{B}\right) \) is the I/O-bound corresponding to the \( O(\log N) \) bound on the operations on many internal memory data structures. More recently researchers have designed

\(^4\) The quotients \( N/B \) (the number of blocks in the problem) and \( M/B \) (the number of blocks that fit into internal memory) play an important role in the study of I/O-complexity. Therefore, we will use \( n \) as shorthand for \( N/B \) and \( m \) for \( M/B \). Furthermore, we will say that an algorithm uses a linear number of I/O-operations if it uses at most \( O(n) \) I/Os to solve a problem of size \( N \).

\(^5\) We define for convenience \( \log_m n = \max\{1, (\log n)/(\log m)\} \).