Chapter 10

EXTERNAL MEMORY ALGORITHMS

Jeffrey Scott Vitter *
Department of Computer Science, Duke University, Durham, NC 27708-0129, USA
jsv@cs.duke.edu

Abstract  Data sets in large applications are often too massive to fit completely inside the computer’s internal memory. The resulting input/output communication (or I/O) between fast internal memory and slower external memory (such as disks) can be a major performance bottleneck. In this paper we survey the state of the art in the design and analysis of external memory (or EM) algorithms, where the goal is to exploit locality in order to reduce the I/O costs.

For sorting and related problems like permuting and fast Fourier transform, the key paradigms include distribution and merging. The paradigm of disk striping offers an elegant way to use multiple disks in parallel. For sorting, however, disk striping can be nonoptimal with respect to I/O, so to gain further improvements we discuss distribution and merging techniques for using the disks independently. We consider EM paradigms for computations involving matrices, geometric data, and graphs, and we look at problems caused by dynamic memory allocation. We report on some experiments in the domain of spatial databases using the TPIE system (Transparent Parallel I/O programming Environment). The newly developed EM algorithms and data structures that incorporate the paradigms we discuss in this chapter are significantly faster than methods currently used in practice.

Keywords:  External memory, Disk block, I/O complexity, Out-of-core, Batched sorting, Fourier transform, Parallel disk model, Locality and load balancing, Disk striping, RAID, Batched computational geometry, Batched graph problems, Indisibility assumption.

*Supported in part by Army Research Office MURI grant DAAH04–96–1–0013 and by National Science Foundation research grants CCR–9522047, EIA–9870734, and CCR–9877133. Part of this work was done at INRIA, Sophia Antipolis, France.

*Earlier versions of some of the material in this chapter appeared in (Vitter 1999a; 1998; 1999b;c; 2001).

J. Abello et al. (eds.), Handbook of Massive Data Sets, 359–416.
1. Introduction

In large applications, data sets are often too massive to fit completely inside the computer's internal memory, and they must be stored on larger but slower external memory devices, such as magnetic disks. The Input/Output communication (or simply I/O) between the fast internal memory and the slow external memory can be a major bottleneck.

Modern programming languages and operating systems support a virtual memory abstraction, in which all of memory is presented as one large uniform address space. In order to cope with the I/O bottleneck, they employ general-purpose caching and prefetching mechanisms, in which the more frequently used data are kept locally in internal memory. If the user or program requests access to data that are not cached, a page fault is generated, and I/O is done to retrieve the data and bring them into internal memory. By their general-purpose nature, caching and prefetching methods cannot be expected to take full advantage of the locality present in every computation. Some computations themselves are inherently non-local, and even with omniscient cache management decisions they are doomed to perform large amounts of I/O and suffer poor performance. Substantial gains in performance may be possible by bypassing the virtual memory system and incorporating locality directly into the algorithm design. We refer to such algorithms as external memory (or EM) algorithms. Some authors use the equivalent terms I/O algorithms or out-of-core algorithms.

In this chapter we discuss several techniques and challenges for how to exploit locality and reduce I/O costs when solving problems in external memory. We concentrate on generic batched problems, in which no preprocessing is done and the entire file of data items must be processed, often by streaming the data through the internal memory in one or more passes. For discussion of online problems we refer the reader to the survey by Arge (2001) in this volume and by Vitter (2001). Other chapters in this volume discuss batched and online problems in specific application domains.

We base our approach upon the parallel disk model (PDM) described in the next section. PDM provides an elegant and reasonably accurate model for analyzing the relative performance of EM algorithms and data structures. The three main performance measures of PDM are the number of I/O operations, the disk space usage, and the CPU time. For reasons of brevity, we focus on the first two measures. Most of the algorithms we consider are also efficient in terms of CPU time. We also mention other memory models and practical considerations.