The Power of Duality for Prefetching and Sorting with Parallel Disks

David A. Hutchinson*, Peter Sanders†, Jeffrey Scott Vitter‡

1. Introduction

External memory (EM) algorithms are designed to be efficient when the problem data do not fit into the high-speed random access memory (RAM) of a computer and must instead reside on external devices such as disk drives [7]. Because of the high latency of accessing data on such devices, data are transferred in units of *blocks* of B contiguous data items, and efficient EM algorithms exploit locality in their design in order to maximize the amount of useful data transferred in each input/output (I/O) step.

But even with blocked access, a single disk provides much lower bandwidth than the internal memory of a computer. This I/O bottleneck can be mitigated by using multiple disks in parallel. In an I/O operation, each of D disks simultaneously transfers a block to or from internal memory. The algorithm thus transfers D blocks at the cost of a single-disk access delay.

In this paper we consider parallel disk input and output separately, in particular as the *prefetch scheduling problem* and the *output scheduling problem*, respectively.

The (online) *queued writing (or output scheduling) problem* takes as input a fixed size pool of m (empty) memory buffers for storing blocks, and the sequence $\langle w_0, w_1, \ldots, w_{L-1}\rangle$ of block *write requests* as they are issued. Each write request is labeled with the disk it will use. The output is an optimal *output schedule*: an ordered sequence of parallel output steps, of minimal length for the given buffer pool capacity.

The (offline) *prefetch scheduling problem* takes as input a fixed size pool of m (empty) memory buffers for storing blocks, and the sequence $\langle r_0, r_1, \ldots, r_{L-1}\rangle$ of distinct block *read requests* that will be issued. Each read request is labeled with the disk it will use. The output is an optimal *prefetch schedule*: an ordered sequence of parallel input steps that allows the requested blocks to be delivered in the required order and is of minimal length for the given buffer pool capacity.

The central theme in this paper is the newly discovered duality between these two problems. We illustrate how applications in one domain can be analyzed via duality with applications in the other domain.

1.1 New Results and Overview

Algorithm *greedyWriting*, which uses a FIFO queue for each disk, is a straightforward online algorithm for output scheduling. It is fed the blocks b_i of a write request sequence Σ one at a time. Block b_i, labeled with an associated disk identifier, is put into the buffer pool and queued behind the other requests buffered for the same disk. One output step, consisting of the leading blocks (if any) of each of the disk queues, is scheduled each time the pool gets completely full.

Whereas the optimality of *greedyWriting* is intuitive and easy to prove, the prefetch scheduling problem seems much more difficult. Given a sequence Σ of write requests, we show a natural *duality* (see Figure 1) between the online output scheduling problem for Σ and the offline prefetching schedul-

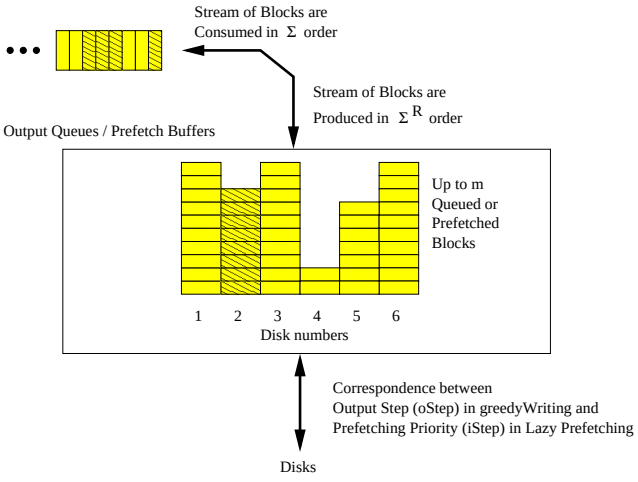

Figure 1: Duality between the prefetching priority and the output step. The hashed blocks illustrate how the blocks of disk 2 might be distributed.

*hutchins@cs.duke.edu. Department of Computer Science, Duke University, Durham, NC 27708–0129. Supported in part by the NSF through research grant CCR–0082986.

†sanders@mpi-sb.mpg.de. Max-Planck-Institute for Computer Science, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany. Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT)

‡jsv@cs.duke.edu. Department of Computer Science, Duke University, Durham, NC 27708–0129. Supported in part by the NSF through research grants CCR–9877133 and EIA–9870734 and by the ARO through MURI grant DAAH04–96–1–0013

ing problem for the reverse sequence Σ^R of read requests. In particular, there is a direct one-to-one correspondence between online write schedules of a sequence Σ and offline prefetch schedules for the reverse sequence Σ^R. This duality immediately implies an optimal, linear time algorithm for prefetching: for read sequences consisting of distinct blocks *greedyWriting* mutates into *lazyPrefetching* via the duality, which also implies its optimality as a prefetching algorithm. Algorithm *lazyPrefetching* is equivalent to the *reverse aggressive* algorithm [5] for the case of read-once sequences and proves that the latter is optimal in the case where the failure penalty F goes to infinity.[1]

A practical disadvantage of *lazyPrefetching* is its laziness; it delays fetching a block as long as possible. Barve et al. [1] and Kallahalla and Varman [3] give scheduling algorithms without this behavior. However the algorithm of [3] is a factor M/D slower than ours and that of [1] has the undesirable behavior that some blocks may be fetched several times. We therefore introduce an optimal variant called *prudentPrefetching* that runs in linear time, tries to fetch blocks as early as possible, and never needs to refetch them.

An optimal writing/prefetching algorithm on its own may be of little practical use if all requests happen to go to the same disk. We apply the concept of duality to transfer previous results on writing [6, 8] to prefetching. The results apply to arbitrarily interleaved accesses to streams allocated fully randomly (FR) or using randomized cycling (RC).[2] Using m buffers, L such block read or write requests can be serviced using $(1 + \mathcal{O}(D/m))L/D$ (expected) parallel I/O steps for sufficiently large L. For FR allocation this is also true for an arbitrary sequence of L requests, not just a sequence representing interleaved accesses to streams.

We again use duality to derive and analyze a very natural algorithm for integrated caching and prefetching, in which the read sequence has multiple instances of blocks. The optimality of the algorithm follows by its duality to an optimal algorithm for the dual write problem. In the write algorithm, the queues are no longer FIFO, but instead follow an ordering motivated by the MIN algorithm of Belady [2]. Kallahalla and Varman [4] recently discovered an optimal algorithm for this problem independently.

Prefetching read-once sequences is an important subproblem of parallel disk merge sort, and writing such sequences is an important subproblem for distribution sort on parallel disks. However, the interleave order for the merge and the partitioning pattern for the distribution are arbitrary and not known in advance. We show that the two problems are dual to one another and can be reduced in a natural way to the problems of read-once prefetching and queued writing, as follows: When the last (largest) key in a block enters the output stream, the block becomes empty and the next block from the run needs to be input. We say that the trigger value for a block is the largest key value in the preceding block. The access order Σ of the blocks being merged is given by the sorted order of the trigger values.[3] If we run the distribution method on the reversed sequence Σ^R, we get an optimal set of priority assignments for use in the merging problem. As a result, we get new and improved external merge sort algorithms.

The distribution sort algorithm RCD from [8] translates into RCM, the first asymptotically optimal variant of striped merge sort. For large N, this algorithm can sort N elements using a multiplicative factor γ more I/Os than the lower bound if $M/B = \Omega(D/\gamma)$ for arbitrarily small γ. Our results are the first to approach the lower bound for $M = \mathcal{O}(BD)$. Together with a improved algorithm for distribution sort, we obtain almost unbeatable algorithms for parallel disk sorting. This provides at least a partial answer to an open question of Knuth regarding a tight analysis of SRM. RCM uses a more elegant prefetching algorithm than the variant of SRM described in [1] and RC is an allocation discipline with more randomness than SR. Otherwise SRM and RCM are identical.

We also discuss how RCD and RCM can be adapted to attain a similar level of performance in a more detailed machine model with finite speed CPUs, variable block length, nonuniform block access times, and rotational delays and seek times.

[1] A *read-once* (resp., *write-once*) sequence is a sequence of distinct block requests that is read (resp., written) in the implied order.

[2] A *stream* is read-once or write-once sequence of blocks. The *allocation discipline* of a sequence of blocks specifies how consecutive blocks are allocated to the disks.

Fully Randomized (FR): Each block is allocated to a random disk.

Simple Randomized (SR): Consecutive blocks of a stream are allocated to consecutive disks in a simple, round-robin manner. The disk selected for the first block is chosen randomly.

Randomized Cycling (RC): Each stream i chooses a random permutation π_i of disk numbers and allocates the jth block of stream i to disk $\pi_i(j \bmod D)$.

Consistent with [1, 8] we denote merge sort variants which use the SR, FR, or RC allocation disciplines as SRM, FRM, or RCM, respectively. Similarly, distribution sort variants using SR, FR, or RC are called SRD, FRD, or RCD, respectively.

[3] If we use forecasting, the trigger definitions change appropriately. For example, if we store in each block of a run the (forecasting) value of the smallest key in the next block, then we define a block's trigger to be the block's smallest key value.

2. References

[1] Barve, R. D., Grove, E. F., and Vitter, J. S. Simple randomized mergesort on parallel disks. *Parallel Computing 23*, 4 (1997), 601–631.

[2] Belady, A. L. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal 5* (1966), 78–101.

[3] Kallahalla, M., and J.Varman, P. Optimal read-once parallel disk scheduling. In *IOPADS* (1999), pp. 68–77.

[4] Kallahalla, M., and Varman, P. Optimal prefetching and caching for parallel I/O systems. In *Proc. of the ACM Symposium on Parallel Algorithms and Architectures* (2001). To appear.

[5] Kimbrel, T., and Karlin, A. R. Near-optimal parallel prefetching and caching. *SIAM Journal on Computing 29*, 4 (2000), 1051–1082.

[6] Sanders, P., Egner, S., and Korst, J. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium on Discrete Algorithms* (2000), pp. 849–858.

[7] Vitter, J. S. External memory algorithms and data structures: Dealing with MASSIVE data. *ACM Computing Surveys* (in press). Available online at `http://www.cs.duke.edu/~jsv/`.

[8] Vitter, J. S., and Hutchinson, D. A. Distribution sort with randomized cycling. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms* (Washington, January 2001).