6.895 Theory of Parallel Systems Project Proposal

Cache-Oblivious Lock-Free Data Structures

Seth Gilbert

1 Introduction

As memory hierarchies become increasingly complicated, it has become difficult both to develop theoretical models that correctly predict the behavior of algorithms with respect to the memory subsystem and to design algorithms that make efficient use of memory. This has led to the development of *cache-oblivious* algorithms. A cache-oblivious algorithm is designed to be optimal, regardless of the underlying memory hierarchy.

1.1 Cache-Oblivious Algorithms

The cache-oblivious model was first introduced by Frigo, Leiserson, Prokop, and Ramachandran [9, 15]. In this model, the algorithm is analyzed with respect to a two-level memory hierarchy. The block size, however, is unknown to the algorithm. It is shown, then, that an algorithm that is optimal in this model is, in fact, optimal for any multi-level memory hierarchy.

Frigo et al. [9, 15] develop cache-oblivious algorithms for matrix multiplication, sorting, and Fast Fourier Transform that are optimal with respect to memory transfers. Since then, a number of other cache-oblivious algorithms have been developed: B-trees, priority queues, tries, etc. [2, 5, 6, 7, 8].

1.2 Lock-Free Algorithms

There has been no study, however, of cache-oblivious algorithms for parallel systems. None of the cache-oblivious algorithms that have been developed address the issues of concurrency.

The most common method of supporting concurrency is to introduce locks. Introducing locks, however, can significantly decrease performance, especially under heavy contention, and can lead to other scalability and fault-tolerance problems (see [10]). We prefer, therefore, to consider algorithms that do not depend on locks.

We focus, then, on wait-free and lock-free algorithms. In a *wait-free* algorithm, every thread will continue to make progress, even when other threads are arbitrarily delayed (or failed). In a *lock-free* algorithm, however, only a single thread is required to make progress. Some research has also focused on weaker requirements (e.g. obstruction-freedom [12]).

2 Problem Statement

We will examine the problem of designing a cache-oblivious, lock-free algorithm for a data structure that supports traversals, insertions, and deletions (i.e., a linked list). This data structure can then be used to develop cache-oblivious, lock-free B-trees, using techniques developed by Bender et al.[4].

Most prior techniques for lock-free linked lists [1, 11, 16, 14] are not readily adaptable to the cache-oblivious model. Bender et al. [3] present a cache-oblivious algorithm for this problem that performs traversals of k elements using $O(\lceil k/B \rceil)$ amortized memory transfers and performs insertions/deletions in $O(\lceil (logN)^2/B \rceil)$ amortized memory transfers (using a data structure similar to that in [13]).

There are two challenges in modifying this algorithm to tolerate concurrency in a lock-free manner. First, insertions must not disrupt ongoing traversal operations: a traversal cannot afford to help complete an insertion, a traversal should not discover the same item more than once, and once an item has been discovered by a traversal, it should be discovered by all later traversals (until it is deleted). Second, the reorganization of the data structure (required by insertions) should be performed efficiently, without requiring too much redundant work.

As in most lock-free algorithms, it is necessary that operations help complete concurrent operations. Fortunately our goal is only to bound amortized memory transfers, so this does not seem immediately infeasible. Care is still needed, however, in ensuring that not too much redundant work is performed. To simplify the problem, insertions can be serialized, using a queue, ensuring that only one insert is modifying the main data structure at any given time. (It is relatively straightforward to implement a cache-oblivious queue with the appropriate lock-free properties, as removing items from the queue does not have to be atomic.) Reorganization can be accomplished in a cooperative, deterministic manner, always moving items in one direction. It remains to fully describe and analyze the resulting algorithm.

References

[1] Agessen, Detlefs, Flood, Garthwaite, Martin, Shavit, and Steele. Dcas-based concurrent deques. In *Proc. 12th ACM Symp. on Parallel Algorithms and Architectures*, July 2000.

[2] Arge, Bender, Demaine, Holland-Minkley, and Munro. Cache-oblivious priority queue and graph algorithm applications. In *STOC*, 2002.

[3] Bender, Cole, Demaine, and Farach-Colton. Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In *ESA*, 2002.

[4] Bender, Demaine, and Farach-Colton. Cache-oblivious b-trees. In *FOCS*, 2000.

[5] Bender, Demaine, and Farach-Colton. Cache-oblivious search trees. In *FOCS*, 2000.

[6] Bender, Demaine, and Farach-Colton. Efficient tree layout in a multilevel memory hierarchy. In *ESA*, 2002.

[7] Bender, Duan, Iacono, and Wu. A locality preserving cache-oblivious dynamic dictionary. In *SODA*, 2002.

[8] Brodal, Fagerberg, and Jacob. Cache oblivious search trees via binary trees of small of small height. In *SODA*, 2002.

[9] Frigo, Leiserson, Prokop, and Ramachandran. Cache-oblivious algorithms. In *FOCS*, 1999.

[10] Greenwald and Cheriton. The synergy between non-blocking synchronization and operating system structure. In *Symposium on Operation System Design and Implementation*, pages 123–136, 1996.

[11] Harris. A pragmatic implementation of non-blocking linked-lists. In *Symposium on Distributed Computing*, oct 2001.

[12] Herlihy, Luchangco, and Moir. Obstruction-free synchronization: Double-ended queues as an example. In *ICDCS*, 2003.

[13] Itai, Konheim, and Rodeh. A sparse table implementation of priority queues. In *Colloquium on Automata, Languages, and Programming*, volume 115, pages 417–431, 1981.

[14] Massalin and Pu. A lock-free multiprocessor os kernel. Technical Report CUCS-005-91, Columbia University, 1991.

[15] Prokop. Cache-oblivious algorithms. Master's thesis, MIT, 1999.

[16] Valois. Lock-free linked lists using compare-and-swap. In *14th Annual ACM Symp. on Principles of Distributed Computing*, pages 214–222, aug 1995.