Cache-Oblivious Sorting
(1999; Frigo, Leiserson, Prokop, Ramachandran)

Gerth Stølting Brodal, Universty of Aarhus, www.daimi.au.dk/~gerth

INDEX TERMS: Sorting, permuting, cache-obliviousness.
SYNONYMS: Funnet sort

1 PROBLEM DEFINITION

Sorting a set of elements is one of the most well studied computational problems. In the cache-oblivious setting the first study of sorting was presented in 1999 in the seminal paper by Frigo *et al.* [8] that introduced the cache-oblivious framework for developing algorithms aimed at machines with (unknown) hierarchical memory.

Model In the cache-oblivious setting the computational model is a machine with two levels of memory: a cache of limited capacity and a secondary memory of infinite capacity. The capacity of the cache is assumed to be M elements and data is moved between the two levels of memory in blocks of B consecutive elements. Computations can only be performed on elements stored in cache, *i.e.* elements from secondary memory need to be moved to the cache before operations can access the elements. Programs are written as acting directly on one unbounded memory, i.e. programs are like standard RAM programs. The necessary block transfers between cache and secondary memory are handled automatically by the model, assuming an optimal offline cache replacement strategy. The core assumption of the cache-oblivious model is that M and B are *unknown to the algorithm* whereas in the related I/O model introduced by Aggarwal and Vitter [1] the algorithms know M and B and the algorithms perform the block transfers explicitly. A throughout discussion of the cache-oblivious model and its relation to multi-level memory hierarchies is given in [8].

Sorting For the sorting problem the input is an array of N elements residing in secondary memory, and the output is required to be an array in secondary memory storing the input elements in sorted order.

2 KEY RESULTS

In the I/O model tight upper and lower bounds were proved for the sorting problem and the problem of permuting an array [1]. In particular it was proved that sorting requires $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers and permuting an array requires $\Theta(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\})$ block transfers. Since lower bounds for the I/O model also hold for the cache-oblivious model, the lower bounds from [1] immediately give a lower bound of $\Omega(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers for cache-oblivious sorting and

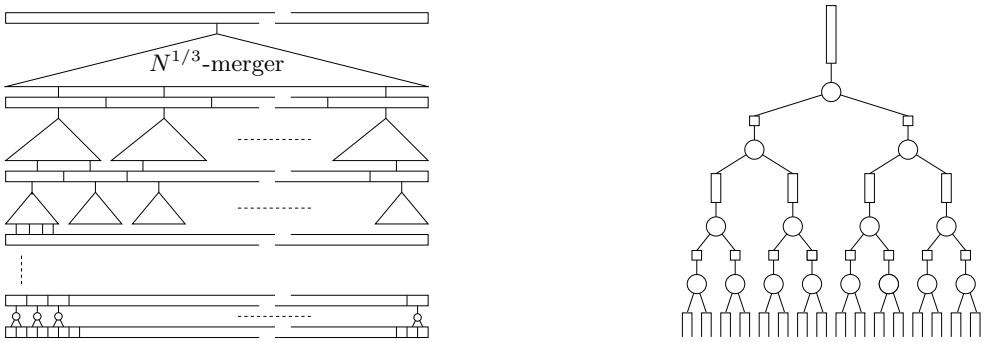

Figure 1: The overall recursion of Funnelsort (left) and a 16-merger (right).

$\Omega(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\})$ block transfers for cache-oblivious permuting. The upper bounds from [1] can not be applied to the cache-oblivious setting since these algorithms make explicit use of B and M.

Binary Mergesort performs $O(N\log_2 N)$ comparisons, but analyzed in the cache-oblivious model it performs $O(\frac{N}{B}\log_2\frac{N}{M})$ block transfers which is a factor $\Theta(\log\frac{M}{B})$ from the lower bound (assuming a recursive implementation of binary Mergesort, in order to get M in the denominator in the $\log N/M$ part of the bound on the block transfers). Another comparison based sorting algorithm is the classical Quicksort sorting algorithm from 1962 by Hoare [9], that performs expected $O(N\log_2 N)$ comparisons and expected $O(\frac{N}{B}\log_2\frac{N}{M})$ block transfers. Both these algorithms achieve their relatively good performance for the number of block transfers from the fact that they are based on repeated scanning of arrays — a property not shared with *e.g.* Heapsort [10] that has a very poor performance of $\Theta(N\log_2\frac{N}{M})$ block transfers. In the I/O model the optimal performance of $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ is achieved by generalizing binary Mergesort to $\Theta(\frac{M}{B})$-way Mergesort [1].

Frigo *et al.* in [8] presented two cache-oblivious sorting algorithms (which can also be used to permute an array of elements). The first algorithm [8, Section 4] is denoted *Funnelsort* and is a reminiscent of classical binary Mergesort, whereas the second algorithm [8, Section 5] is a distribution based sorting algorithm. Both algorithms perform *optimal* $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ block transfers — provided a *tall cache assumption* $M = \Omega(B^2)$ is satisfied.

Funnelsort The basic idea of Funnelsort is to rearrange the sorting process performed by binary Mergesort, such that the processed data is stored "locally". This is achieved by two basic ideas: 1) A top-level recursion that partitions the input into $N^{1/3}$ sequences of size $N^{2/3}$, Funnelsort these sequences recursively, and merge the resulting sorted subsequences using an $N^{1/3}$-*merger*. 2) A k-merger is recursively defined to perform binary merging of k input sequences in a clever schedule with an appropriate recursive layout of data in memory using buffers to hold suspended merging processes (see Figure 1). Subsequently two simplifications were made, without sacrificing the asymptotic number of block transfers performed. In [3] it was proved that the binary merging could be performed lazily, simplifying the scheduling of merging. In [5] it was further observed that the recursive layout of k-mergers is not necessary. It is sufficient that a k-merger is stored in a consecutive array, *i.e.* the buffers can be laid out in arbitrary order which simplifies the construction algorithm for the k-merger.

Implicit cache-oblivious sorting Franceschini in [7] showed how to perform optimal cache-oblivious sorting implicitly using only $O(1)$ space, *i.e.* all data is stored in the input array except for $O(1)$ additional words of information. In particular the output array is just a permutation of the input array.

The rôle of the tall cache assumption The role of the tall cache assumption on cache-oblivious sorting was studied by Brodal and Fagerberg in [4]. If no tall cache assumption is made, they proved the following theorem:

Theorem 1 ([4], Corollory 3)**.** *Let $B_1 = 1$ and $B_2 = M/2$. For any cache-oblivious comparison based sorting algorithm, let t_1 and t_2 be upper bounds on the number of I/Os performed for block sizes B_1 and B_2. If for a real number $d \geq 0$ it is satisfied that $t_2 = d \cdot \frac{N}{B_2} \log_{M/B_2} \frac{N}{B_2}$ then $t_1 > 1/8 \cdot N \log_2 N/M$.*

The theorem shows cache-oblivious comparison based sorting without a tall cache assumption cannot match the performance of algorithms in the I/O model where M and B are known to the algorithm. It also has the natural interpretation that if a cache-oblivious algorithm is required to be I/O-optimal for the case $B = M/2$, then binary Mergesort is best possible—any other algorithm will be the same factor of $\Theta(\log M)$ worse than the optimal block transfer bound for the case $M \gg B$.

For the related problem of permuting an array the following theorem states that for all possible tall cache assumptions $B \leq M^\delta$, no cache-oblivious permuting algorithm exists with a block transfer bound (even only in the average case sense) matching the worst case bound in the I/O model.

Theorem 2 ([4], Theorem 2)**.** *For all $\delta > 0$, there exists no cache-oblivious algorithm for permuting that for all $M \geq 2B$ and $1 \leq B \leq M^\delta$ achieves $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$ I/Os averaged over all possible permutations of size N.*

3 APPLICATIONS

Many problems can be reduced to cache-oblivious sorting. In particular Arge *et al.* [2] developed a cache-oblivious priority queue based on a reduction to sorting. They furthermore showed how a cache-oblivious priority queue can be applied to solve a sequence of graph problems, including list ranking, BFS, DFS, and minimum spanning trees.

Brodal and Fagerberg in [3] showed how to modify the cache-oblivious lazy Funnelsort algorithm to solve several problems within computational geometry, including orthogonal line segment intersection reporting, all nearest neighbors, 3D maxima problem, and batched orthogonal range queries. All these problems can be solved by a computation process very similarly to binary Mergesort with an additional problem dependent twist. This general framework to solve computational geometry problems is denoted *distribution sweeping*.

4 OPEN PROBLEMS

Since the seminal paper by Frigo *et al.* [8] introducing the cache-oblivious framework there has been a lot of work on developing algorithms with a good theoretical performance, but only a limited amount of work has been done one implementing these algorithms. An important issue for future work is to get further experimental results consolidating the cache-oblivious model as a relevant model for dealing efficiently with hierarchical memory.

5 EXPERIMENTAL RESULTS

A detailed experimental study of the cache-oblivious sorting algorithm Funnelsort was performed in [5]. The main result of [5] is a carefully implemented cache-oblivious sorting algorithm can be faster than a tuned implementation of Quicksort already for input sizes well within the limits of RAM. The implementation is also at least as fast as the recent cache-aware implementations included in the test. On disk the difference is even more pronounced regarding Quicksort and the cache-aware algorithms, whereas the algorithm is slower than a careful implementation of multiway Mergesort optimized for external memory such as in TPIE [6].

6 URL to CODE

http://kristoffer.vinther.name/projects/funnelsort/

7 CROSS REFERENCES

1. I/O-model
2. Cache-Oblivious Model
3. External Sorting and Permuting

8 RECOMMENDED READING

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] L. Arge, M. A. Bender, E. D. Demaine, B. Holland-Minkley, and J. I. Munro. Cache-oblivious priority queue and graph algorithm applications. In *Proc. 34th Annual ACM Symposium on Theory of Computing*, pages 268–276. ACM Press, 2002.

[3] G. S. Brodal and R. Fagerberg. Cache oblivious distribution sweeping. In *Proc. 29th International Colloquium on Automata, Languages, and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 426–438. Springer Verlag, Berlin, 2002.

[4] G. S. Brodal and R. Fagerberg. On the limits of cache-obliviousness. In *Proc. 35th Annual ACM Symposium on Theory of Computing*, pages 307–315, 2003.

[5] G. S. Brodal, R. Fagerberg, and K. Vinther. Engineering a cache-oblivious sorting algorithm. *ACM Journal of Experimental Algorithmics, Special Issue of ALENEX 2004*, 12(Article No. 2.2):23, 2007.

[6] Department of Computer Science, Duke University. TPIE: a transparent parallel I/O environment. WWW page, `http://www.cs.duke.edu/TPIE/`, 2002.

[7] G. Franceschini. Proximity mergesort: Optimal in-place sorting in the cache-oblivious model. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 291. SIAM.

[8] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symposium on Foundations of Computer Science*, pages 285–297. IEEE Computer Society Press, 1999.

[9] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–15, April 1962.

[10] J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.