Cache-Oblivious Algorithms and Data Structures

Gerth Stølting Brodal[⋆]

BRICS[⋆⋆], Department of Computer Science, University of Aarhus
IT-parken, Åbogade 34, DK-8200 Århus N, Denmark

Abstract. Frigo, Leiserson, Prokop and Ramachandran in 1999 introduced the ideal-cache model as a formal model of computation for developing algorithms in environments with multiple levels of caching, and coined the terminology of *cache-oblivious algorithms*. Cache-oblivious algorithms are described as standard RAM algorithms with only one memory level, i.e. without any knowledge about memory hierarchies, but are analyzed in the two-level I/O model of Aggarwal and Vitter for an arbitrary memory and block size and an optimal off-line cache replacement strategy. The result are algorithms that automatically apply to multi-level memory hierarchies. This paper gives an overview of the results achieved on cache-oblivious algorithms and data structures since the seminal paper by Frigo *et al.*

1 Introduction

Modern computers are characterized by having a memory system consisting of a hierarchy of several levels of memory, where each level is acting as a cache for the next level [46]. The typical memory levels of current machines are registers, level 1 cache, level 2 cache, level 3 cache, main memory, and disk. While the sizes of the levels increase with the distance from the CPU the access times to the levels also get larger, most dramatically when going from main memory to disk. To circumvent dramatic performance loss data is moved between the memory levels in blocks (cache lines or disk blocks). As a consequence of this organization of the memory, the memory access pattern of an algorithm has a major influence on its practical running time. A basic rule commonly stated in the literature for achieving good running times is to ensure *locality of reference* in the developed algorithms.

1.1 I/O model

Several models have been proposed in recent years to model modern memory hierarchies. The most successful of these models (in terms of number of publications) is the two-level I/O model introduced by Aggarwal and Vitter in 1988 [6]:

[⋆] Supported by the Carlsberg Foundation (contract number ANS-0257/20). Email: gerth@daimi.au.dk.

[⋆⋆] Basic Research in Computer Science, www.brics.dk, funded by the Danish National Research Foundation.

The memory hierarchy is assumed to consist of two levels, a main memory of size M and an infinite secondary memory, where data is transfered between the two levels in blocks of B consecutive elements. Computations are performed on elements in main memory and algorithms have complete control over block transfers, I/Os, between the two levels. The resource studied in the I/O model is the number of I/Os performed by algorithms, e.g. does the scanning of an N element array in secondary memory imply $\Theta(N/B)$ I/Os. Aggarwal and Vitter in their seminal paper [6] proved that in the I/O model, comparison based sorting requires $\Theta(\mathrm{Sort}_{M,B}(N)) = \Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os, which is achieved by $Theta(\frac{M}{B})$-ary multi-way mergesort, and searching requires $\Theta(\log_B N)$ I/Os, which is acheived by B-trees [17].

The success of the I/O model is likely due to its simplicity making the design and analysis of external memory algorithms feasible, while adequately modeling the case where the I/Os between two levels of the memory hierarchy dominates the running time. For an overview of the comprehensive work done related to the I/O model we refer the reader to the surveys by Arge [9] and Vitter [69], and the book [57].

More sophisticated multi level models have been studied in the literature [3–5, 7, 16, 47, 62, 63, 70, 71], but none of these have gained the same level of attention as the I/O model of Aggarwal and Vitter [6], likely due to the complexity of describing algorithms for these models.

A limitation of the I/O model is that the parameters B and M are required to be known to the algorithms. In practice, these parameters might not always be available. Furthermore the available memory for a process may wary over time, e.g. in a multiprocess environment the available memory depends on the memory usage of the other processes being scheduled.

1.2 Ideal-cache model

Frigo, Leiserson, Prokop and Ramachandran in 1999 introduced the *ideal-cache model* and coined the terminology of *cache-oblivious algorithms* [44]. The ideal-cache model can be viewed as a formal framework for analyzing the locality of reference of an algorithm that is oblivious about the presence of the memory hierarchy. The basic idea is to describe algorithms for the standard RAM model with only one memory level, i.e. without any knowledge about memory hierarchies. The algorithms are then analyzed in the two-level I/O model of Aggarwal and Vitter for an arbitrary memory size M and block size B, assuming that I/Os are performed by an optimal off-line cache replacement strategy. Since the analysis of an cache oblivious algorithm should hold for all values of M and B, the analysis also holds for all levels of a multi-level memory hierarchy (see [44] for a detailed discussion of the technical requirements to be satisfied).

For algorithms satisfying that reducing the cache size by a factor two does not increase the number of I/Os by more than a constant factor, Frigo *et al.* [44] proved that the assumption of an optimal off-line cache replacement strategy can be replaced by the on-line least-recently used (LRU) cache replacement strategy, by appealing to Sleator and Tarjan's classic competitiveness result [64] for LRU-paging. Since LRU is adaptive to dynamically changing memory sizes, cache oblivious algorithms are also adaptive to changes in the available memory.

A naive cache-oblivious algorithm is the scanning of an N element array that requires optimal $\Theta(N/B)$ I/Os. The linear time selection algorithm of Blum *et al.* [27] primarily is based on scanning and it can be proved that their selection algorithm is an optimal cache-oblivious algorithm performing $\Theta(N/B)$ I/Os.

Frigo *et al.* in their seminal paper [44] considered cache-oblivious algorithms for several algorithmic problems: The transposition of an $n \times m$ matrix was solved using optimal $\mathcal{O}(mn/B)$ I/Os. The multiplication of an $m \times n$-matrix and an $n \times p$-matrix was solved using $\mathcal{O}((mn+np+mp)/B + mnp/(B\sqrt{M}))$ I/Os. For square matrices this matches a lower bound by Hong and Kung [47] for algorithms computing the matrix product only using additions and multiplications. In [44] it was furthermore proved that Strassen's matrix multiplication algorithm [65] is cache-oblivious and requires $\mathcal{O}(n + n^2/B + n^{\log_2 7}/(B\sqrt{M}))$ I/Os. Optimal comparison based sorting algorithms performing $\mathcal{O}(\mathrm{Sort}(N))$ I/Os were presented, under the so called *tall cache assumption* $M = \Omega(B^2)$. Both merging based (Funnelsort) and distribution based sorting algorithms were presented. Finally an algorithm for fast Fourier transform (FFT) was presented requiring $\mathcal{O}(\mathrm{Sort}(N))$ I/Os. A cache-oblivious algorithm for LU decomposition with pivoting appeared in [66].

The remaining of this paper gives an overview of the results on cache-oblivious algorithms and data structures achieved during the five years since the seminal paper by Frigo *et al.* Recent surveys on cache-oblivious algorithms and data structures can also be found in [13, 38, 50].

2 Sorting and permuting

The first cache-oblivious sorting algorithms were presented by Frigo *et al.* [44]: One based on the merging paradigm, Funnelsort, and one based on the distribution paradigm. Both algorithms require the tall cache assumption $M \geq B^2$. A simplified version of Funnelsort was presented in [28], denoted Lazy Funnelsort, requiring the tall cache assumption $M \geq B^{1+\varepsilon}$. An empirical study of the developed cache-oblivious sorting algorithms is presented in [33].

That I/O optimal cache-oblivious comparison based sorting is not possible without a tall cache assumption is proved in [30]. The paper shows an inherent trade-off for cache-oblivious algorithms between the strength of the tall cache assumption and the overhead for the case $M \gg B$. The result implies that both Funnelsort and recursive binary mergesort are optimal algorithms in the sense that they attain this trade-off, where recursive binary mergesort does not require a tall cache assumption but performs $\mathcal{O}(\frac{N}{B}\log_2\frac{N}{M})$ I/Os.

Permuting N elements in an array can be solved by either moving each element independently to its new position by $\mathcal{O}(1)$ I/Os or by sorting the elements by their new positions. In [6] it is proved that permuting in the I/O model requires $\Theta(\min\{N, \mathrm{Sort}(N)\})$ I/Os. In [30] it is proved that no cache-oblivious algorithm can match this I/O performance, not even in the presence of a tall cache assumption.

Variations of cache-oblivious sorting have been studied. An implicit cache-oblivious sorting algorithm was presented in [41], i.e. an algorithm that works with a single array of size N only storing the N input elements plus $\mathcal{O}(1)$ machine words. Sorting multisets has been studied in [40].

3 List labeling

Itai *et al.* [48] studied the problem of maintaining N elements in sorted order in an array of length $\mathcal{O}(N)$, an important problem in dynamic dictionaries when an efficient range query operation is required to be supported [22]. The problem is commonly denoted the list labeling problem, but in [22] denoted the packed memory management problem. The reorganization primitive in [48] during insertions and deletions of elements is the even redistribution of the elements in a section of the array. Their approach uses amortized $\mathcal{O}(\log^2 N)$ work per update. A matching $\Omega(\log^2 N)$ lower bound for algorithms using even redistribution as the primitive was given in [39]. A worst-case variant was developed by Willard in [72]. Bender *et al.* [22] adapted the algorithms to the cache oblivious setting, supporting insertions and deletions in the array in amortized $\mathcal{O}((\log^2 N)/B)$ I/Os, and guaranteeing that there are only $\mathcal{O}(1)$ empty slots between two consecutive elements in the array. Bender *et al.* [18] refined the last labeling solution to satisfy the property that every update (in addition to every traversal) consists of $\mathcal{O}(1)$ physical scans sequentially through memory. Updates still require amortized $\mathcal{O}((\log^2 N)/B)$ I/Os.

4 Search trees

Prokop in [60] proposed static cache-oblivious search trees with search cost $\mathcal{O}(\log_B N)$ I/Os, matching the search cost of standard (cache-aware) B-trees [17]. The search trees of Prokop are related to a data structure of van Emde Boas [67, 68], since the recursive layout of a search tree generated by Prokop's scheme resembles the layout of the search trees of van Emde Boas. The constant in the $\mathcal{O}(\log_B N)$ search cost was studied in [21], where it is proved that no cache-oblivious algorithm can achieve a performance better than $\log_2 e \cdot \log_B N$ I/Os, i.e. a factor ≈ 1.44 slower than a cache-aware algorithm. Cache oblivious search trees avoiding the usage of pointers were presented in [31, 53, 59].

Dynamic B-trees were first presented by Bender *et al.* [22] achieving searches in $\mathcal{O}(\log_B N)$ I/Os and updates requiring amortized $\mathcal{O}(\log_B N)$ I/Os. Simplified constructions were presented in [23] and [31], where [31] is based on combining the recursive static layout of Prokop [60] and the dynamic search trees of low height by Andersson and Lai [8], and [23] is based on combining the static layout of Prokop with a data structure for the list labeling problem.

A cache-oblivious dictionary based on exponential search trees was presented in [19]. The paper shows how to make the exponential search trees partially persistent, i.e. support queries in previous versions of the search tree, and how to support efficient cache-oblivious finger searches, i.e. searches in the vicinity of a given element. The layout of arbitrary static trees was considered in [20]. Finally, optimal cache-oblivious implicit dictionaries were developed in [42] and [43].

5 Priority queues

Arge *et al.* [11] presented the first cache-oblivious priority, supporting inserts and delete-min operations in $\mathcal{O}(\frac{1}{B}\log_{M/B}\frac{N}{B})$ I/Os. This matches the performance achieved in the I/O model by e.g. the buffer trees of Arge [10]. The construction in [11] is a general reduction to sorting. An alternative cache-oblivious priority achieving the same I/O complexity as [11] was presented in [29]. This solution is a more direct solution based on k-mergers introduced in the Funnelsort algorithm [44, 28].

6 Graph algorithms

The existence of a cache-oblivious priority queue enabled a sequence of cache-oblivious graph algorithms. In [11] the following deterministic cache-oblivious bounds are obtained: List ranking, computing the Euler tour of a tree, breadth first search (BFS) of a tree, and depth first search (DFS) of a tree, all requiring $\mathcal{O}(\text{Sort}(E))$ I/Os, matching the known bounds for the I/O model achieved in [36]. For directed BFS and DFS on general graphs a cache-oblivious algorithm was presented performing $\mathcal{O}((V+E/B)\log V+\text{Sort}(E))$ I/Os, matching the known best bounds for the I/O model [34]. For undirected DFS, an algorithm performing $\mathcal{O}(V+\text{Sort}(E))$ I/Os was achieved, matching the bound for the I/O model in [58]. Finally an $\mathcal{O}(\text{Sort}(E)\log\log V)$ I/O minimum spanning tree algorithm was presented, nearly matching the $\mathcal{O}(\text{Sort}(E)\log\log\frac{VB}{E})$ I/O bound in [12] for the I/O model.

Abello *et al.* [1] presented for the I/O model a functional approach to solve a sequence of graph problems based on recursion and repeated use of sorting and scanning. Their randomized minimum spanning tree algorithm is actually also cache-oblivious and performs expected $\mathcal{O}(\text{Sort}(E))$ I/Os.

In [56] it was shown how to solve undirected BFS in $\mathcal{O}(\text{ST}(E)+\text{Sort}(E)+\sqrt{VE/B})$ I/Os for the I/O model, where $\text{ST}(E)$ denotes the I/O bound for computing a spanning tree of the graph. In [32] two cache-oblivious versions of the algorithm in [56] were developed requiring $\mathcal{O}(\text{ST}(E)+\text{Sort}(E)+\frac{E}{B}\log V+\sqrt{VE/B})$ and $\mathcal{O}(\text{ST}(E)+\text{Sort}(E)+\frac{E}{B}\cdot\frac{1}{\varepsilon}\cdot\log\log V+\sqrt{VE/B}\cdot\sqrt{VB/E}^{\varepsilon})$ I/Os respectively.

Undirected single source shortest path (SSSP) can be solved cache-obliviously in $\mathcal{O}(V+E/B\log(E/B))$ I/Os [32, 37], matching the known bounds for the I/O model [51].

7 Computational geometry

Goodrich *et al.* [45] introduced the distribution sweeping approach to solve a sequence of problems within computational geometry in the I/O model. A cache-oblivious version of the distribution sweeping approach is developed in [28], achieving the following results, where N is the input size, T the output size: The 3D maxima problem on a set of points, computing the measure of a set of axis-parallel rectangles, the all nearest neighbors problem, and computing the visibility of a set of non-intersecting line segments from a point can be solved using optimal $\mathcal{O}(\text{Sort}(N))$ I/Os. The orthogonal line segment intersection reporting problem, batched orthogonal range queries, and reporting pairwise intersections of axis-parallel rectangles can be solved using optimal $\mathcal{O}(\text{Sort}(N)+\frac{T}{B})$ I/Os,

A cache-oblivious data structure for the planar point location problem was presented in [19]. In requires linear space, taking optimal $\mathcal{O}(\log_B N)$ I/Os for point location queries, where N is the number of line segments specifying the partition of the plane. The pre-processing requires $\mathcal{O}((N/B)\log_{M/B} N)$ I/Os.

Cache-oblivious algorithms for orthogonal range searching were presented in [2], both a kd-tree and range-tree solution were presented. A cache-oblivious kd-tree is simply a normal kd-tree [24] laid out in memory using the van Emde Boas layout. This structure uses linear space and answers queries in $\mathcal{O}(\sqrt{N/B}+\frac{K}{B})$ I/Os; this is optimal among linear space structures [49]. Insertions are facilitated using the so-called logarithmic method of Bentley [25], and require $\mathcal{O}(\frac{\log N}{B}\log_{M/B} N)$ I/Os. The cache-oblivious range-tree presented in [2] supports range queries in $\mathcal{O}(\log_B N+\frac{K}{B})$ I/Os and requires space $\mathcal{O}(N\log^2 N)$.

8 Lower bounds

A general reduction technique for proving lower bounds for comparison based algorithms for the I/O model was presented in [15], allowing the reduction to standard comparison trees. Lower bounds achieved for the I/O model immediately apply to cache-oblivious algorithms also.

Bilardi and Peserico [26] have investigated the portability of algorithms across memory hierarchies in the HRAM-model, where they provide a CDAG computation and two machines such that any scheduling of the computation is a factor polynomial from optimal on at least one of the machines. For cache-oblivious algorithms lower bounds have been given for searching [21], and sorting and permuting [30].

9 Empirical work

The impact of different memory layouts for data structures has been studied before in different contexts. In connection with matrices, significant speedups can be achieved by using layouts optimized for the memory hierarchy—see e.g. the paper by Chatterjee *et al.* [35] and the references it contains.

Ladner *et al.* considered the effect of caches in connection with heaps [54], sorting [55], and sequential and random traversals [52]. Using registers to improve the running time of sorting was considered in [14]. Minimizing translation look-aside buffer (TLB) misses, and the case of low cache associativity was studied in [73]. Rahman *et al.* [61] made an empirical study of the performance of various search tree implementations, with focus on showing the significance of minimizing TLB misses. Brodal *et al.* [31] studied different memory layouts for near perfect-balanced search trees. Ladner *et al.* [53] gave a comparison of cache aware and cache-oblivious static search trees using program instrumentation. Empirical investigations of the practical efficiency of cache-oblivious algorithms for sorting was done in [33],

The overall conclusion of these investigations is that cache-oblivious methods often outperform RAM algorithms, but not always as much as algorithms tuned to the specific memory hierarchy and problem size. On the other hand, cache-oblivious algorithms perform well on all levels of the memory hierarchy, and seem to be more robust to changing parameter sizes than cache-aware algorithms.

10 Summary

Since the seminal paper by Frigo *et al.* [44] in 1999 an amazing sequence of papers has been published on various cache-oblivious problems and data structures, establishing cache-oblivious algorithms as an important subfield of external memory algorithms. Empirical work has documented the soundness of the cache-oblivious approach. The level of success (in terms of number of publications) as for the I/O model of Aggarwal and Vitter has not been achieved yet for the cache-oblivious model, likely due to the complexity of the algorithm descriptions: The ideal-cache model forces the logical structure of an cache-oblivious algorithm in most cases to be more complex than the structure of a corresponding algorithm for the I/O model.

References

1. J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32(3):437–458, 2002.
2. P. Agarwal, L. Arge, A. Danner, and B. Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proc. 19th ACM Symposium on Computational Geometry*, pages 237 – 245. ACM Press, 2003.
3. A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir. A model for hierarchical memory. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 305–314. AMC Press, 1987.
4. A. Aggarwal and A. Chandra. Virtual memory algorithms. In *Proc. 20th Annual ACM symposium on Theory of computing*, pages 173–185. ACM Press, 1988.
5. A. Aggarwal, A. K. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 204–216. IEEE Computer Society Press, 1987.

6. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, Sept. 1988.
7. B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2–3):72–109, 1994.
8. A. Andersson and T. W. Lai. Fast updating of well-balanced trees. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 111–121. Springer, 1990.
9. L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, 2002.
10. L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.
11. 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.
12. L. Arge, G. Brodal, and L. Toma. On external-memory MST, SSSP and multi-way planar graph separation. In *Proc. 8th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 433–447. Springer, 2000.
13. L. Arge, G. S. Brodal, and R. Fagerberg. Cache-oblivious data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, page 27. CRC Press, 2004.
14. L. Arge, J. Chase, J. Vitter, and R. Wickremesinghe. Efficient sorting using registers and caches. *ACM Journal of Experimental Algorithmics*, 7(9), 2002.
15. L. Arge, M. Knudsen, and K. Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In *Proc. 3rd Workshop on Algorithms and Data Structures*, volume 709 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 1993.
16. R. D. Barve and J. S. Vitter. A theoretical framework for memory-adaptive algorithms. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 273–284. IEEE Computer Society Press, 1999.
17. R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.
18. M. Bender, R. Cole, E. Demaine, and M. Farach-Colton. Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In *Proc. 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 139–151. Springer, 2002.
19. M. Bender, R. Cole, and R. Raman. Exponential structures for cache-oblivious algorithms. In *Proc. 29th International Colloquium on Automata, Languages, and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2002.
20. M. Bender, E. Demaine, and M. Farach-Colton. Efficient tree layout in a multilevel memory hierarchy. In *Proc. 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 165–173. Springer, 2002. Full version at http://www.cs.sunysb.edu/~bender/pub/treelayout-full.ps.
21. M. A. Bender, G. S. Brodal, R. Fagerberg, D. Ge, S. He, H. Hu, J. Iacono, and A. López-Ortiz. The cost of cache-oblivious searching. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282. IEEE Computer Society Press, 2003.

22. M. A. Bender, E. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 399–409. IEEE Computer Society Press, 2000.
23. M. A. Bender, Z. Duan, J. Iacono, and J. Wu. A locality-preserving cache-oblivious dynamic dictionary. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 29–38. ACM-SIAM, 2002.
24. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communication of the ACM*, 18:509–517, 1975.
25. J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.
26. G. Bilardi and E. Peserico. A characterization of temporal locality and its portability across memory hierarchies. In *Proc. 28th Annual International Colloquium on Automata, Languages and Programming*, volume 2076, pages 128–139. Springer, 2001.
27. M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.
28. 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, 2002.
29. G. S. Brodal and R. Fagerberg. Funnel heap - a cache oblivious priority queue. In *Proc. 13th Annual International Symposium on Algorithms and Computation*, volume 2518 of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2002.
30. G. S. Brodal and R. Fagerberg. On the limits of cache-obliviousness. In *Proc. 35th ACM Symposium on Theory of Computing*, pages 307–315. ACM Press, 2003.
31. G. S. Brodal, R. Fagerberg, and R. Jacob. Cache oblivious search trees via binary trees of small height. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 39–48. ACM-SIAM, 2002.
32. G. S. Brodal, R. Fagerberg, U. Meyer, and N. Zeh. Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In *Proc. 9th Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science. Springer, 2004.
33. G. S. Brodal, R. Fagerberg, and K. Vinther. Engineering a cache-oblivious sorting algorithm. In *Proc. 6th Workshop on Algorithm Engineering and Experiments*, page 14, 2004.
34. A. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. Westbrook. On external memory graph traversal. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 859–860. ACM Press, 2000.
35. S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *Proc. 1999 Conference on Supercomputing*, ACM SIGARCH, pages 444–453. ACM Press, 1999.
36. Y. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149. ACM-SIAM, 1995.
37. R. A. Chowdhury and V. Ramachandran. Cache-oblivious shortest paths in graphs using buffer heap. In *Proc. 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures.* ACM Press, 2004.
38. E. D. Demaine. Cache-oblivious data structures and algorithms. In *Proc. EFF summer school on massive data sets*, Lecture Notes in Computer Science. Springer, 2004, to appear.

39. P. F. Dietz and J. Zhang. Lower bounds for monotonic list labeling. In J. R. Gilbert and R. G. Karlsson, editors, *Proc. 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 173–180. Springer, 1990.
40. A. Farzan and J. I. Munro. Cache-oblivious sorting and searching in multisets. Manuscript, 2004.
41. G. Franceschini. Proximity mergesort: optimal in-place sorting in the cache-oblivious model. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–299. ACM-SIAM, 2004.
42. G. Franceschini and R. Grossi. Optimal cache-oblivious implicit dictionaries. In *Proc. 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 316–331. Springer, 2003.
43. G. Franceschini and R. Grossi. Optimal worst-case operations for implicit cache-oblivious search trees. In *Proc. 8th International Workshop on Algorithms and Data Structures*, volume 2748 of *Lecture Notes in Computer Science*, pages 114–126. Springer, 2003.
44. M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual IEEE Symposium on Foundations of Computer Science*, pages 285–297. IEEE Computer Society Press, 1999.
45. M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proc. 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 714–723. IEEE Computer Society Press, 1993.
46. J. L. Hennessy and D. A. Patterson, editors. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 3 edition, 2002.
47. J.-W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proc. 13th Annual ACM Symposium on Theory of Computation*, pages 326–333. AMC Press, 1981.
48. A. Itai, A. G. Konheim, and M. Rodeh. A sparse table implementation of priority queues. In *Automata, Languages and Programming, 8th Colloquium*, volume 115 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1981.
49. K. V. R. Kanth and A. K. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proc. 7th International Conference on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 1999.
50. P. Kumar. Cache oblivious algorithms. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*, pages 193–212. Springer, 2003.
51. V. Kumar and E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proc. 8th SPDP*, pages 169–177. IEEE Computer Society Press, 1996.
52. R. E. Ladner, J. D. Fix, and A. LaMarca. Cache performance analysis of traversals and random accesses. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 613–622. ACM-SIAM, 1999.
53. R. E. Ladner, R. Fortna, and B.-H. Nguyen. A comparison of cache aware and cache oblivious static search trees using program instrumentation. In *Experimental Algorithmics*, volume 2547 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2002.
54. A. LaMarca and R. E. Ladner. The influence of caches on the performance of heaps. *ACM Journal of Experimental Algorithms*, 1:4, 1996.
55. A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. *Journal of Algorithms*, 31:66–104, 1999.

56. K. Mehlhorn and U. Meyer. External-memory breadth-first search with sublinear I/O. In *Proc. 10th ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 723–735. Springer, 2002.
57. U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*. Springer, 2003.
58. K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–694. ACM-SIAM, 1999.
59. D. Ohashi. Cache oblivious data structures. Master's thesis, Department of Computer Science, University of Waterloo, Waterloo, Canada, 2000.
60. H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, June 1999.
61. N. Rahman, R. Cole, and R. Raman. Optimised predecessor data structures for internal memory. In *Proc. 5th International Workshop on Algorithm Engineering*, volume 2141, pages 67–78. Springer, 2001.
62. J. E. Savage. Extending the Hong-Kung model to memory hierachies. In *Proc. 1st Annual International Conference on Computing and Combinatorics*, volume 959 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 1995.
63. S. Sen and S. Chatterjee. Towards a theory of cache-efficient algorithms. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 829–838. ACM-SIAM, 2000.
64. D. D. Sleator and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28:202–208, 1985.
65. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
66. S. Toledo. Locality of reference in *LU* decomposition with partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1065–1081, 1997.
67. P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.
68. P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.
69. J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, June 2001.
70. J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.
71. J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory II: Hierarchical multilevel memories. *Algorithmica*, 12(2–3):148–169, 1994.
72. D. E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Information and Computation*, 97(2):150–204, 1992.
73. L. Xiao, X. Zhang, and S. A. Kubricht. Improving memory performance of sorting algorithms. *ACM Journal of Experimental Algorithmics*, 5(3), 2000.