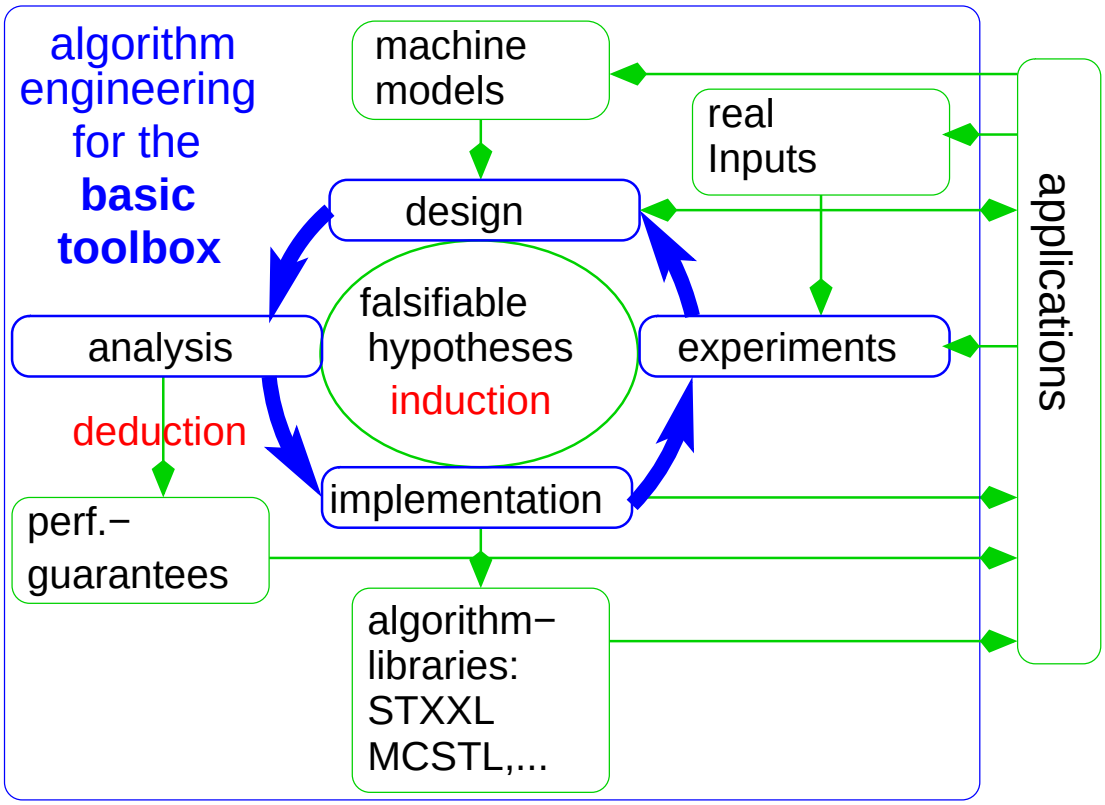
Algorithm Engineering for the Basic Toolbox

Peter Sanders
Fakultät für Informatik
Universität Karlsruhe
76128 Karlsruhe

June 4, 2007

Excerpt from proposal for DFG Priority Programme 1307 — Algorithm Engineering



Contents

1 New Proposal for a Research Grant

1.1 Topic

Engineering efficient algorithms for the basic algorithmic toolbox with emphasis on algorithm libraries, memory hierarchies and parallelism.

1.2 Field and Area (Fachgebiet und Arbeitsrichtung)

computer science, algorithmics

1.3 Summary (Zusammenfassung)

This project addresses algorithm engineering for basic algorithms and data structures that are the most important building blocks for many computer applications — sorting, searching, graph traversal,.... Although this topic is as old as computers science itself, many interesting new results have appeared in recent years and many gaps between theory and practice remain. In particular, many interesting approaches have not been thoroughly tried experimentally. Ever more complex hardware with memory hierarchies and several types of parallel processing requires refined models, new algorithms, and efficient implementations. We plan to incorporate the most successful implementations into reusable software libraries such as the the C++ STL.

2 Previous Work

Since the basic toolbox is a wide area, it is impossible to give a complete review of previous work. The presented selection aims at demonstrating that there are many interesting open problems. We also give some background for subareas we plan to work on.

2.1 State of Research (Stand der Forschung)

Sometimes, previous work bearing direct relations to our own past or planned research is mentioned in the later sections to simplify reading and to reduce redundancy.

2.1.1 Models

The RAM model remains a simple and useful model for sequential programs with good memory locality. Disk accesses are successfully modelled by transfers of blocks of size B between secondary memory and a fast memory of size M [AV88, VS94a]. This *I/O-model* is also useful between main memory and one cache level. By *hiding* the parameters B and M from the algorithm, we arrive at the cache-oblivious model [FLPR99]. cache-oblivious algorithms work equally well on all levels of a memory hierarchy. However, there are only few implementations of cache oblivoius algorithms that yield good performance in practice (e.g., [BFV04]). Parallel models are more problematic. The PRAM model is largely regarded as unrealistic and there is no generally accepted model for asynchronous shared memory machines. BSP [Val94] is a realistic model for distributed memory machines but bases all communication on an expensive and inflexible globally synchronized data exchange. No model is in sight that adequately models multiple levels of caches and local memories, that are shared between some but not all processors. There are several attempts but most of them are too abstract or too complicated. For example, [VS94b] is too complicated although it ignores communication costs.

More detailed and more specialized models are needed for *runtime prediction*. There has been very little work outside the the area of numeric computing (e.g. [FM97]).

2.1.2 Sequence Representation

State of the art for RAM algorithms are (resizable) arrays and linked list. A folklore result is that for cache efficiency, a hybrid between these approaches if often better. But only very few implementations actually implement this (e.g. [FPR06]). In Section 2.1.10 we outline challenges in the context of concurrent access to sequences.

2.1.3 Hashing

Hashing with chaining and linear probing are still the algorithms most widely used in practice. It remains an interesting question to what extent perfect hashing [DKM$^+$94] or other ways to achieve worst case constant access time will turn out to be useful in practice. For example, cuckoo hashing and its relatives [PR01, DW05] look elegant, simple and efficient on the first glance and experiments look good. However, cuckoo insertion is only guaranteed to terminate for hash functions with certain performance guarantees that are likely to be too expensive in practice. More generally, there seems to be a relateively little work yet on hash functions that are both practical and give good performance guarantees [DadH90, Tho00].

2.1.4 Sorting and Selection

Sorting is perhaps the single most intensively studied algorithmic problem because it is at the same time simple, nontrivial, and immensely useful. The sorting literature shows the typical gap between theory and practice. For example, there is an "optimal" deterministic algorithm for parallel disk sorting [NV95] but its constant factors are so big that "suboptimal" implementations in data base papers outclass it for all practical situations. In some advanced models of computation, there is not even an aymptotically optimal algorithm published.

For internal memory sequential sorting, there has been considerable recent work on cache-efficient sorting, e.g., [RKU00, WACV02].

Theoretical algorithms for integer sorting have made impressive progress [YH02] but none of them is practical and even radix sort for small keys needs very careful (cache-efficient) implementation to beat quicksort (e.g. [GNLP01, Rah03]).

2.1.5 Priority Queues

There are a number of very interesting algorithm engineering results on priority queues: one of the first systematic algorithm engineering papers [Jon86]; a DIMACS implementation challenge; studies of priority queues together with MST [MS94] and shortest path [CGR96, Gol01] calculations. Some theoretical results like pairing heaps [FSST86] are inspired by the idea of simplifying theoretically optimal algorithms like Fibonacci heaps [FT84]. However, this has resulted in a zoo of variants (e.g. [Hœy95] without conclusive experimental comparisons. The only experimental paper we are aware of makes simulations with variants of pairing heaps [SV87] using synthetic inputs but gives little insight on performance of real implementations with real inputs.

There is a gap between theoretical [Tho04] and practical integer priority queues [Bro88, ELL94, AT96] analogous to the one in integer sorting.

There are several interesting results on parallel priority queues (e.g. [Ran94, DPS96, BTZ98]) where it is not clear whether they are practical.

2.1.6 Sorted Sequences

Efficient operations on sorted sequences can be implemented using search tree (e.g. [AVL62]) and skip-list [Pug90] data structures. There are many variants. Experimental studies with sequential, internal memory data structures [MN99] indicate that performance differences are more dependent on implementation details than on algorithmic differences.

There is also a lot of work on search trees in memory hierarchies. There is a lot of implementation work for B-Trees [BM72] in data bases [Com79, GL01, Pag03]. There are theoretical results for string B-Trees [FG99] that contain very interesting ideas but do not look practical yet. As for sorting and priority queues, there is comparably little work on algorithm engineering for sorted sequences with integer keys [vEB77, MN90].

2.1.7 Graphs

While textbooks still only teach the difference between adjacency matrices and adjacency lists, it is now well known that *adjacency arrays* are the most efficient practical representation of graphs for most algorithms [NZ02]. There is also some work on clustering graphs such that graph traversal is accelerated [MZ03]. However, these techniques may not be fully developed yet.

Relatively little work has been done on systematic engineering of the most basic graph traversal algorithms breadth-first-search and depth-first-search. The reason may be that this looks trivial. However, even here nontrivial algorithmic choices are to be made.

There is so much work on algorithm engineering for shortest paths that we abstain from looking at it. We also omit more complex graph algorithms like maximum flow.

2.1.8 Strings

Algorithm engineering for string processing is highly developed but still suffers from gaps between theory and practice. Section 2.2.9 gives an example for the problem of suffix sorting.

2.1.9 Algorithm Libraries

Numerical algorithm libraries are routinely used since the early 1970s. Progress was much slower for nonnumerical algorithms perhaps because without generic typing, algorithm libraries remain awkward. For example, the C standard library always had a quick sort, but it is slow and rarely used because comparison relies on a pointer to a comparison function.

Object oriented programming languages like Smalltalk, C++ or Java now have algorithm libraries as part of their standard. In particular, collection classes for sets and sequences are used extensively.

More advanced algorithms and data structures are available in LEDA [MN99], JDSL [MN99], or Boost [Boo]. These libraries are widely used, yet perhaps not

as widely as one should think. One problem is performance. For example, programs with a handwritten graph data structure are often several times faster than comparable programs using a library.

2.1.10 Parallel Processing

We will discuss parallel algorithms for fundamental problems in the sections for these problems. However, the basic toolbox for writing parallel programs contains additional ingredients that have their own algorithmic challenges:

Communication. Processors of distributed memory machines interact using a set of frequently occurring communication patterns that are therefore part of message passing libraries such as MPI [GLS95]. In particular, the *collective communication* operations broadcast, reduction, prefix sum, gossipping, all-to-all, etc. are algorithmically interesting.

Shared Data Structures. Threads in a shared memory machine interact by concurrent access to memory. Synchronizing these accesses is a nontrivial requirement for obtaining correct programs. Currently, synchronization is mostly done via low-level primitives such as locks and semaphores that have limited scalability. Efficient synchronization uses even lower level atomic operations provided by the hardware, such as compare-and-swap and fetch-and-increment. Data structures for memory management, queues, hash tables, search trees, ... implemented using these atomic operations promise to become scalable, high level tools for writing shared memory programs. Interesting algorithms for maintaining stacks and FIFOs on shared memory machines have been developed and evaluated experimentally (e. g. [ALS94, HLM03, DHLM04, DHLM04, LH04, LAKL04, TZ03]). However, this is only slowly adapted into widely available algorithm libraries. For example, even the "Intel Threading Building Blocks" library[1] which claims to give efficient implementations of parallel programming primitives, currently has a quite unsatisfactory FIFO.

Scheduling and Load Balancing: Keeping all processors busy with relevant work without incurring too many expensive interactions is a crucial problem of parallelization. A small set of load balancing algorithms can be used for a large spectrum of applications. This problem has been intensively studied both in theory and practice. However, there are several open problems: scheduling for low energy consumption, coping with ever more complex parallel memory hierarchies, and reusable libraries.

[1] `http://www.intel.com/cd/software/products/asmo-na/eng/294797.htm`

2.2 Own Previous Work (Eigene Vorarbeiten)

2.2.1 Methodology

We have done work on presenting experimental data [San02a], how (or whether) to infer asymptotic behavior from experiments [SF00, MSF+02], and on parallel algorithm engineering [BMS02].

2.2.2 Models

In [San94b] we have given the strongest known result how single instruction multiple data (SIMD) parallel machines can emulate multiple instruction multiple data (MIMD) machines. Interestingly, the SIMD model has recently been "rediscovered" for the Clearspeed coprocessors which have 96 SIMD processors [Tec06]. Another older result that is increasingly becoming relevant is a paper that analyzes how energy consumption bounds the scalability of parallel machines [SVW97]. A particular focus has been on realistic models for storage arrays [SEK00, SEK03, San04b, San02b, CSW07] and external memory [San03a]. The main result here is that using randomization, redundancy, and sophisticated scheduling algorithms, one can hide the complexity of such systems by emulating a much more powerful model that allows concurrent access to arbitrary data blocks. The influence of the limited associativity of hardware caches is considered in [San99a, MS03a]. Surveys on the I/O model are given in [MSS03, San04a].

2.2.3 Sequence Representation

The hybrid representation mentioned in Section 2.1.2 is used in [KSB05] to design a space efficient linear time suffix array construction algorithm. In [DSSS04], this kind of data structure (in internal memory!) is important for efficient external memory minimum spanning trees. We also have an unpublished space efficient implementation of the priority queue data structure [San00b].

2.2.4 Hashing

Several new and practical hash functions with provable performance guarantees are given in [MS07]. A practical and space efficient data structure for dynamic hashing with worst case constant access time is given in [FPSS03, FPSS05]. Closely related space efficient static hash tables are used in [BFM+07] for fast shortest path queries and in [ST07] for representing inverted indices. This is a good example for our claim that there are enough crosscutting issues in algorithm engineering to justify a quite wide project area.

2.2.5 Sorting, Selection, and Permuting

Sorting and related problems has been one of our main areas of research. The first parallel implementation of quicksort that works efficiently on massively parallel computers is given in [SH97]. On grid connected machines it might be the best practical algorithm. We also have one of the fastest practical external memory sorters [DS03, DKS05] that in contrast to other practical implementations also has good theoretical performance guarantees [HSV01, HSV05].

Even for internal memory, sequential sorting we managed to obtain some interesting results. For example, even highly tuned cache-efficient sorting algorithms such as [RKU00, WACV02, BFV04] yield only about 20 % speedup compared to standard quicksort [Hoa61, Mus97], although they incur several times less cache faults. The reason is that quicksort is "sufficiently" cache-efficient so that another bottleneck can manifest itself—branch mispredictions. Indeed, we have shown [KS06a] that the seemingly paradoxical measure to choose pivots that are far from the median can slightly accelerate quicksort because a significantly reduced number of branch mispredictions more than outweighs the increased number of executed instructions and memory accesses. On the first glance, there seems to be no way out of the dilemma between too many executed instructions and too many branch mispredictions for comparison based sorting—the information theoretic lower bound of $\log n!$ comparisons can only be obtained if the outcome of the comparisons is completely unpredictable. However, in [SW04] we could show that comparisons can be completely decoupled from branch instructions altogether using *predicated instructions*. Since the same measure reduces *data dependencies*, we can also improve instruction parallelism. Finally, the method is very cache-efficient. Only combining all these architectural effects, it was possible to obtain a factor up to two speedup compared to an efficient implementation of quicksort.

The recursion of quicksort is also a practical way to solve the more general problem of *multiple selection* where we only ask for a subset of the input elements with specified ranks [Cha71]. Interestingly, despite of intensive work on multiple selection, the exact analysis of this algorithm was open for 35 years. We have shown that it can be made optimal up to a linear additive term in a strong information theoretical sense that takes the actual values of the ranks into account [KMMS05]. There was also no other algorithm with similar performance known until we gave a deterministic (albeit not so practical) algorithm with similar asymptotic performance [KMMS05].

2.2.6 Priority Queues

We have engineered a parallel priority queue that allows fast concurrent access by many processors [San95b, San98a]. This is important for parallel processing because it allows a bottleneck free implementation of a pool of prioritized jobs. The implementation [San98a] indicates that already for more than about 40 processors, the algorithm outperforms a centralized implementation. The algorithm also has equal or better theoretical performance guarantees than a considerable body of previous theoretical research.

Despite all the work described in Section 2.1.5, from the 1960s until 1999 there was no serious competitor for binary heaps [Wil64] for sequential, comparison-based priority queues in internal memory, if only insertion and delete-min are needed. Small improvements due to better cache efficiency are possible by increasing the degree of the heap [LL96]. But I/O-efficient priority queues like [Arg95, FJKT97, BK98, BCMF99] that promised higher asymptotic improvements have constant factors that make them impractical for the hierarchy between cache and main memory. However, in [San99c, San00b] we showed how to remove a factor of ≥ 3 in I/O complexity from the approaches [FJKT97, BK98, BCMF99], and thus going close to the lower bound. An efficient internal memory implementation is 2–3 times faster than binary heaps for large inputs. This implementation has now also been adapted to the external setting where it is the work horse for several world leading implementations of external memory graph algorithms [DSSS04, DKS05, Dem06].

2.2.7 Sorted Sequences

In 1977, van Emde Boas invented an intriguing idea for search trees with integer keys in the range $0..2^k - 1$ that allow access in time $\mathcal{O}(\log k)$. But only in 1990 there were the first, space efficient implementations [MN90, Wen92]. They turned out to be about a factor two *slower* than comparison-based search trees [Wen92]. In 2004, we reversed this performance ratio for the important case of 32 bit keys using a highly tuned implementation [DKMS04].

2.2.8 Graphs

We have bridged two gaps between theory and practice for computing minimum spanning trees: [DSSS04] describes the first practical external memory algorithm for minimum spanning trees. The theoretical performance guarantees are asymptotically worse than the best theoretical results [ABT04, ABW02], but a factor ≥ 4 *better* than all previously known algorithms, if one takes into account that external

memory is at best a few hundred times cheaper than internal memory. Our implementation is also only a factor 2–5 slower than a fast internal algorithm that is given enough fast memory. It can process graphs with billions of nodes in a few hours.

The theoretically best internal memory MST algorithms use the cycle property and sophisticated data structures for least common ancestor queries to eliminate heavy edges in constant time per edge [Kin97, KKT95]. However, the algorithms are so complicated and the involved constant factors are so large that a practical implementation looks prohibitive. In [KST03], we give the first practical MST algorithm that profits from the cycle property at least for rather dense graph. At the cost of slower preprocessing, it simplifies the data structures and speeds up the cost per edge by a crucial constant factor.

We have done intensive work on the shortest path problem. Our parallel shortest path algorithms [CMMS98, MS98, MS00, MS03b] are now used in several implementations [EBGL06, MBBC06]. A current focus are shortest path queries in large road networks when (fast and space efficient) preprocessing is allowed. In the last few years, there has been a veritable race for the fastest method. Since 2005 we are leading this race with short interruptions [SS05, SS06, BFM+07]. For example, recently we have won the DIMACS implementation challenge on shortest path queries [2]. In the same context, we have also obtained results on goal directed speedup techniques [MSM06, DSSW06] and the first algorithm for very fast computation of distance tables for vehicle routing problems [KSS+07].

Further algorithm engineering work on fundamental graph algorithms includes the first implementations of external memory breadth first search [ADM07], a simple algorithm for approximate maximum weighted matching [PS04], and the first implementation of the theoretically best algorithm for maximum flow [HST98].

2.2.9 Strings

In 2003 we developed the first linear time algorithm for constructing suffix arrays [KS03, KSB05]. Since this algorithm is very simple and because suffix arrays have many applications in data compression, full text indexing, and bioinformatics, there have been several implementations. So far, these implementations cannot beat highly engineered asymptotically suboptimal implementations such as [MF02] for real world instances. However, the algorithm turned out to be the basis for the best practical algorithms for parallel [KS06b, KS07] and external memory [DKMS05, DKMS07] suffix array construction.

[2] `http://dimacs.rutgers.edu/Workshops/Challenge9/`

2.2.10 Algorithm Libraries

We have experience with algorithm libraries for algebraic data types [San93], parallel search [San97, San98b], external memory (STXXL) [DKS05, Dem06], and multi-cores (MCSTL) [SS07]. The latter two libraries augment and partially reimplement the C++ standard template library STL. Thus, some programs can be externalized or parallelized by simple recompilation. At the very least, we get an interface that is already known to millions of programmers. Furthermore, we can expect that compiler writers invest a lot of work in efficiently compiling programs written in "STL style". The STXXL is already used at ≥ 17 sites.

2.2.11 Parallel Processing Primitives

We have worked on collective communication operations for large messages [SS00, SS03, ST06], for hierarchical machines [ST02] and for irregular message lengths [SSO00, SSO01]. Some of these results have been incorporated into the NEC MPI implementation. We have also developed a widely used benchmark for communication in MPI [RSPM98]. The MCSTL [SS07] uses atomic shared memory operations supported by the hardware for providing its own implementations of a shared queue data structure. MCSTL also load-balances many of its algorithms and contains a dynamic load balancer for parallelizable loops. Further work on load balancing includes [San94a, San95a, San96, San99b, San00a, SEK00, San02c, San03b, KSV03, CSW07].

Appendices

- References cited in the text. Papers with coauthors from our group are marked with a bullet.
- CV with list of publications for Peter Sanders
- A selection of three papers by Peter Sanders [SW04, DKS05, BFM$^+$07] (A short version of the [BFM$^+$07], [BFSS07], will appear in Science.)

References

[ABT04] L. Arge, G. S. Brodal, and L. Toma. On external-memory MST, SSSP and multi-way planar graph separation. *J. Algorithms*, 53(2):186–206, 2004.

[ABW02] J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32:437–458, 2002.

[ADADC+97] A. C. Arpaci-Dusseau, R. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. A. Patterson. High-performance sorting on networks of workstations. In Joan Peckham, editor, *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 243–254, Tucson, Arizona, 13–15 June 1997.

[ADM07] D. Ajwani, R. Dementiev, and U. Meyer. A computational study of external-memory bfs algorithms. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 601–610, 2007.

[AFGV97] L. Arge, P. Ferragina, R. Grossi, and J. S. Vitter. On sorting strings in external memory. In *29th ACM Symposium on Theory of Computing*, pages 540–548, El Paso, May 1997. ACM Press.

[ALS94] H. Attiya, N. Lynch, and N. Shavit. Are wait-free algorithms fast? *J. ACM*, 41(4):725–763, 1994.

[AP94] A. Aggarwal and C. G. Plaxton. Optimal parallel sorting in multi-level storage. In *5th ACM-SIAM Symposium on Discrete Algorithms*, pages 659–667, 1994.

[Arg95] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. In *4th Workshop on Algorithms and Data Structures*, number 955 in LNCS, pages 334–345. Springer, 1995.

[AT96] A. Andersson and M. Thorup. A pragmatic implementation of monotone priority queues. In *DIMACS'96 implementation challenge*, 1996.

[AV88] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[AVL62] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

[BCMF99] K. Brengel, A. Crauser, U. Meyer, and P. Ferragina. An experimental study of priority queues in external memory. In *3rd International Workshop on Algorithmic Engineering (WAE)*, pages 345–359, 1999. full paper in ACM Journal of Experimental Algorithmics.

[BDF00] M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In Danielle C. Young, editor, *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 399–409, Los Alamitos, California, November 12–14 2000. IEEE Computer Society.

[BFJ02] G. S. Brodal, R. Fagerberg, and R. Jacob. Cache oblivious search trees via binary trees of small height. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA-02)*, pages 39–48, New York, January 6–8 2002. ACM Press.

[BFM+07] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In • transit to constant time shortest-path queries in road networks. In *9th Workshop on Algorithm Engineering and Experiments*, 2007.

[BFSS07] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road • networks with transit nodes. *Science*, 2007. accepted.

[BFV04] G. S. Brodal, R. Fagerberg, and K. Vinther. Engineering a cache-oblivious sorting algorithm. In *6th Workshop on Algorithm Engineering and Experiments*, 2004.

[BK98] G. Stølting Brodal and J. Katajainen. Worst-case efficient external-memory priority queues. In *6th Scandinavian Workshop on Algorithm Theory*, number 1432 in LNCS, pages 107–118. Springer Verlag, Berlin, 1998.

[BM72] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173 – 189, 1972.

[BMS02] D. Bader, B. Moret, and P. Sanders. Algorithm engineering for par- • allel computation. In *Experimental Algorithmics — From Algorithm*

Design to Robust and Efficient Software, volume 2547 of *LNCS*, pages 1–23. Springer, 2002.

[Boo] Boost.org. boost C++ Libraries. `www.boost.org`.

[Bro88] R. Brown. Calendar queues: A fast $O(1)$ priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, 1988.

[BTZ97] G. S. Brodal, J. L. Träff, and C. D. Zaroliagis. A parallel priority data structure with applications. In *11th International Parallel Processing Symposium*, 1997. to appear.

[BTZ98] G. S. Brodal, J. L. Träff, and C. D. Zaroliagis. A parallel priority queue with constant time operations. *Journal of Parallel and Distributed Computing*, 49(1):4–21, 25 February 1998.

[CGR96] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest path algorithms: Theory and experimental evaluation. *Math. Programming*, 73:129–174, 1996.

[Cha71] J. Chambers. Partial sorting (algorithm 410). *Communications of the ACM*, 14:357–358, 1971.

[Cle84] J. G. Cleary. Compact hash tables using bidirectional linear probing. *IEEE Transactions on Computers*, C-33(9):828–834, 1984.

[CM96] J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*, 15(6):521–549, 1996.

[CMMS98] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra's shortest path algorithm. In *23rd Symposium on Mathematical Foundations of Computer Science*, number 1450 in LNCS, pages 722–731, Brno, Czech Republic, 1998. Springer.

[Com79] D. Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[CSW07] J. Cain, P. Sanders, and N. Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *18th ACM-SIAM Symposium on Discrete Algorithms*, 2007. to appear.

[DadH90] M. Dietzfelbinger and F. Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Automata, Languages and Programming (ICALP '90)*, pages 6–19, Berlin - Heidelberg - New York, July 1990. Springer.

[Dem06] R. Dementiev. *Algorithm Engineering for Large Data Sets*. PhD thesis, Saarland University, 2006. •

[DHLM04] S. Doherty, M. Herlihy, V. Luchangco, and M. Moir. Bringing practical lock-free synchronization to 64-bit applications. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 31–39, New York, NY, USA, 2004. ACM Press.

[DKM$^+$94] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, August 1994.

[DKMS04] R. Dementiev, L. Kettner, J. Mehnert, and P. Sanders. Engineering a sorted list data structure for 32 bit keys. In *Workshop on Algorithm Engineering & Experiments*, New Orleans, 2004. •

[DKMS05] R. Dementiev, J. Kärkkäinen, J. Mehnert, and P. Sanders. Better external memory suffix array construction. In *Workshop on Algorithm Engineering & Experiments*, pages 86–97, Vancouver, 2005. •

[DKMS07] R. Dementiev, J. Kärkkäinen, J. Mehnert, and P. Sanders. Better external memory suffix array construction. *ACM Journal of Experimental Algorithmics*, 2007. to appear in special issue on Alenex 2005. •

[DKS05] R. Dementiev, L. Kettner, and P. Sanders. STXXL: Standard Template Library for XXL data sets. In *13th European Symposium on Algorithms*, volume 3669 of *LNCS*, pages 640–651. Springer, 2005. •

[DMPP02] F. Dehne, S. Mardegan, A. Pietracaprina, and G. Prencipe. Distribution sweeping on clustered machines with hierarchical memories. In *IPDPS*, 2002.

[DPS96] S. K. Das, M. C. Pinotti, and F. Sarkar. Optimal and load balanced mapping of parallel priority queues in hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 7(6):555–564, 1996.

[DS03] R. Dementiev and P. Sanders. Asynchronous parallel disk sorting. In *15th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 138–148, San Diego, 2003.

[DSSS04] R. Dementiev, P. Sanders, D. Schultes, and J. Sibeyn. Engineering an external memory minimum spanning tree algorithm. In *IFIP TCS*, Toulouse, 2004.

[DSSW06] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Highway hierarchies star. In *9th DIMACS Implementation Challenge - Shortest Paths*, 2006. `http://www.dis.uniroma1.it/~challenge9/papers.shtml`.

[DW05] M. Dietzfelbinger and C. Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 166–178. Springer, 2005.

[EBGL06] N. Edmonds, A. Breuer, D. Gregor, and A. Lumsdaine. Single-source shortest paths with the parallel boost graph library. In *9th DIMACS Implementation Challenge - Shortest Paths*, 2006. `http://www.dis.uniroma1.it/~challenge9/papers.shtml`.

[ELL94] K. B. Erickson, R. E. Ladner, and A. LaMarca. Optimizing static calendar queues. In Shafi Goldwasser, editor, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 732–743, Los Alamitos, CA, USA, November 1994. IEEE Computer Society Press.

[FG99] P. Ferragina and R. Grossi. The string b-tree: a new data structure for string search in external memory and its applications. *J. ACM*, 46(2):236–280, 1999.

[FJKT97] R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. External heaps combined with effective buffering. In *4th Australasian Theory Symposium*, volume 19-2 of *Australian Computer Science Communications*, pages 72–78. Springer, 1997.

[FLPR99] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Symposium on Foundations of Computer Science*, pages 285–298, 1999.

[FM97] U. Finkler and K. Mehlhorn. Runtime prediction of real programs on real machines. In *SODA'97*, pages 380–389, 1997.

[FPR06] L. Frias, J. Petit, and S. Roura. Lists Revisited: Cache Conscious STL Lists. In *5th International Workshop on Experimental Algorithms (WEA)*, volume 4007, pages 121–133. Lecture Notes in Computer Science, May 2006.

[FPSS03] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space efficient hash tables with worst case constant access time. In *20th International Symposium on Theoretical Aspects of Computer Science*, number 2607 in LNCS, pages 271–282, Berlin, 2003. Springer.

[FPSS05] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, 2005.

[FSST86] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. The pairing heap: A new form of self adjusting heap. *Algorithmica*, 1(1):111–129, 1986.

[FT84] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *25th Annual Symposium on Foundations of Computer Science*, pages 338–346, Los Angeles, Ca., USA, October 1984. IEEE Computer Society Press.

[GL01] G. Graefe and Per-Ake Larson. B-tree indexes and cpu caches. In *ICDE*, pages 349–358. IEEE, 2001.

[GLS95] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. MIT Press, 1995.

[GNLP01] D. J. Gonzalez, J. J. Navarro, and J. Larriba-Pey. Fast parallel in-memory 64-bit sorting. In *ACM Supercomputing*, pages 114–122, 2001.

[Gol01] A. V. Goldberg. Shortest path algorithms: Engineering aspects. In *12th International Symposium on Algorithms and Computation*, number 2223 in LNCS, pages 502–513. Springer, 2001.

[HLM03] M. Herlihy, V. Luchangco, and M. Moir. Obstruction-free synchronization: Double-ended queues as an example. In *23th International Conference on Distributed Computing Systems*, pages 522–529, Providence, RI, May 2003. IEEE Computer Society.

[Hoa61] C. A. R. Hoare. Quicksort. *Communication of the ACM*, 4(7):321, 1961.

[Hœy95] P. Hœyer. A general technique for implementation of efficient priority queues. In *3rd Israeli Symposium on Theory of Computing and Systems*, pages 57–66, 1995.

[HST98] T. Hagerup, P. Sanders, and J. L. Träff. An implementation of the binary blocking flow algorithm. In K. Mehlhorn, editor, *2nd Workshop on Algorithm Engineering*, number MPI-I-98-1-019, ISSN: 0946-011X in Research Reports MPII, pages 143–154, 1998.

[HSV01] D. A. Hutchinson, P. Sanders, and J. S. Vitter. Duality between prefetching and queued writing with parallel disks. In *9th European Symposium on Algorithms (ESA)*, number 2161 in LNCS, pages 62–73. Springer, 2001.

[HSV05] D. A. Hutchinson, P. Sanders, and J. S. Vitter. Duality between prefetching and queued writing with parallel disks. *SIAM Journal on Computing*, 34(6):1443–1463, 2005.

[IKR81] A. Itai, A. G. Konheim, and M. Rodeh. A sparse table implementation of priority queues. In S. Even and O. Kariv, editors, *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, volume 115 of *LNCS*, pages 417–431, Acre, Israel, July 1981. Springer.

[Jon86] D. Jones. An empirical comparison of priority-queue and event set implementations. *Communications of the ACM*, 29(4):300–311, 1986.

[Kin97] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997.

[KKT95] David Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. *J. Assoc. Comput. Mach.*, 42:321–329, 1995.

[KMMS05] K. Kaligosi, K. Mehlhorn, J. I. Munro, and P. Sanders. Towards optimal multiple selection. In *32th International Colloquium on Automata, Languages and Programming*, number 3580 in LNCS, pages 103–114, Lissabon, 2005. Springer.

[Knu98] D. E. Knuth. *The Art of Computer Programming—Sorting and Searching*, volume 3. Addison Wesley, 2nd edition, 1998.

[KS03] J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In *30th International Colloquium on Automata, Languages and Programming*, number 2719 in LNCS, pages 943–955, 2003.

[KS06a] K. Kaligosi and P. Sanders. How branch mispredictions affect quicksort. In *14th European Symposium on Algorithms (ESA)*, volume 4168 of *LNCS*, pages 780–791, 2006.

[KS06b] F. Kulla and P. Sanders. Scalable parallel suffix array construction. In *Euro PVM/MPI*, volume 4192, of *LNCS*, pages 22–29, Bonn, 2006. distinguished paper.

[KS07] F. Kulla and P. Sanders. Scalable parallel suffix array construction. *Parallel Computing*, 2007. invited for special issue on Euro PVM/MPI 2006.

[KSB05] J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 2005. to appear (last published issue Sept. 2005 !).

[KSS+07] S. Knopp, P. Sanders, D. Schultes, F. Schulz, and D. Wagner. Computing many-to-many shortest paths using highway hierarchies. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007.

[KST03] I. Katriel, P. Sanders, and J. L. Träff. A practical minimum spanning tree algorithm using the cycle property. In *11th European Symposium on Algorithms (ESA)*, number 2832 in LNCS, pages 679–690. Springer, 2003.

[KSV03] P. Krysta, P. Sanders, and B. Vöcking. Scheduling and traffic allocation for tasks with bounded splittability. In *28th International Symposium on Mathematical Foundations of Computer Science*, number 2747 in LNCS, pages 500–510. Springer, 2003.

[LAKL04] S. Lie, K. Asanovic, B. Kuszmaul, and C. E. Leiserson. Hardware transactional memory. Technical report, MIT CS, Jan 2004. `https://dspace.mit.edu/handle/1721.1/3860`.

[LH04] K. Olukotun et al. L. Hammond. Transactional memory coherence and consistency. *ACM SIGARCH Computer Architecture News*, 2004.

[LL96] A. LaMarca and R. E. Ladner. The influence of caches on the performance of heaps. *ACM Journal of Experimental Algorithmics*, 1(4), 1996.

[MBBC06] K. Madduri, D. Bader, J. W. Berry, and J. R. Crobak. Parallel shortest path algorithms for solving large-scale instances. In *9th DIMACS Implementation Challenge - Shortest Paths*, 2006. `http://www.dis.uniroma1.it/~challenge9/papers.shtml`.

[MF02] G. Manzini and P. Ferragina. Engineering a lightweight suffix array construction algorithm. In *10th European Symposium on Algorithms*, volume 2461 of *LNCS*, pages 698–710. Springer, 2002.

[MN90] K. Mehlhorn and S. Näher. Bounded ordered dictionaries in $O(\log\log N)$ time and $O(n)$ space. *Information Processing Letters*, 35(4):183–189, 1990.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[MS94] B.M.E. Moret and H.D. Shapiro. An empirical assessment of algorithms for constructing a minimum spanning tree. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 15:99–117, 1994.

[MS98] U. Meyer and P. Sanders. Δ-stepping: A parallel shortest path algorithm. In *6th European Symposium on Algorithms (ESA)*, number 1461 in LNCS, pages 393–404. Springer, 1998.

[MS00] U. Meyer and P. Sanders. Parallel shortest path for arbitrary graphs. In *6th Euro-Par*, number 1900 in LNCS, pages 461–470, 2000. •

[MS03a] K. Mehlhorn and P. Sanders. Scanning multiple sequences via cache memory. *Algorithmica*, 35(1):75–93, 2003. •

[MS03b] U. Meyer and P. Sanders. Δ-stepping: A parallelizable shortest path algorithm. *Journal of Algorithms*, 2003. in press. •

[MS07] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures — The Basic Toolbox*. Springer, 2007. in preparation. •

[MSF+02] C. McGeoch, P. Sanders, R. Fleischer, P. R. Cohen, and D. Precup. Using finite experiments to study asymptotic performance. In *Experimental Algorithmics — From Algorithm Design to Robust and Efficient Software*, volume 2547 of *LNCS*, pages 1–23. Springer, 2002. •

[MSM06] J. Maue, P. Sanders, and D. Matijevic. Goal directed shortest path queries using
Precomputed Cluster Distances. In *5th Workshop on Experimental Algorithms (WEA)*, volume 4007 of *LNCS*, pages 316–328. Springer, 2006. •

[MSS03] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*. Springer, 2003. •

[Mus97] D. R. Musser. Introspective sorting and selection algorithms. *Softw. Pract. Exper.*, 27(8):983–993, 1997.

[MZ03] A. Maheshwari and N. Zeh. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*, chapter A Survey of Techniques for Designing I/O-Efficient Algorithms, pages 36–61. Springer, 2003.

[NV93] M. H. Nodine and J. S. Vitter. Deterministic distribution sort in shared and distributed memory multiprocessors. In *5th ACM Symposium on Parallel Algorithms and Architectures*, pages 120–129, Velen, Germany, 1993.

[NV95] M. H. Nodine and J. S. Vitter. Greed sort: An optimal sorting algorithm for multiple disks. *Journal of the ACM*, 42(4):919–933, 1995.

[NZ02] Stefan Näher and Oliver Zlotowski. Design and implementation of efficient data types for static graphs. *Lecture Notes in Computer Science*, 2461:748–??, 2002.

[PAB+05] Pham, D. Asano, S. Bolliger, M. Day, M.N. Hofstee, H.P. Johns, C. Kahle, J. Kameyama, A. Keaty, J. Masubuchi, Y. Riley, M. Shippy, D. Stasiak, D. Suzuoki, M. Wang, M. Warnock, J. Weitzel, S. Wendel, D. Yamazaki, T. Yazawa, and K. Pham et al. The design and implementation of a first-generation CELL processor. In *Solid-State Circuits Conference*, volume 1, pages 184–192, 2005.

[Pag03] R. Pagh. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*, chapter Basic External Memory Data Structures, pages 14–35. Springer, 2003.

[Pec02] Marcin Peczarski. Sorting 13 elements requires 34 comparisons. In Rolf H. Möhring and Rajeev Raman, editors, *10th European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 785–794. Springer, 2002.

[PPR05] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal bloom filter replacement. In *SODA*, pages 823–829. SIAM, 2005.

[PR01] Pagh and Rodler. Cuckoo hashing. In *ESA: Annual European Symposium on Algorithms*, volume 2161 of *LNCS*. Springer, 2001.

[PS04] S. Pettie and P. Sanders. A simpler linear time $2/3 - \epsilon$ approximation • for maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.

[Pug90] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[Rah03] N. Rahman. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*, chapter Algorithms for Hardware Caches and TLB, pages 171–192. Springer, 2003.

[Ran94] A. Ranade. Optimal speedup for backtrack search on a butterfly network. *Mathematical Systems Theory*, 27(1):85–101, 1994.

[RCD+94] A. Ranade, S. Cheng, E. Deprit, J. Jones, and S. Shih. Parallelism and locality in priority queues. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 97–103, October 1994.

[Ric94] Ivan L. M. Ricarte. *Performance and Scalability of Parallel Database Systems*. PhD thesis, University of Maryland College Park, College Park, MD, USA, January 1994.

[RKU00] A. Ranade, S. Kothari, and R. Udupa. Register efficient mergesorting. In *High Performance Computing — HiPC*, volume 1970 of *LNCS*, pages 96–103. Springer, 2000.

[RSPM98] R. Reussner, P. Sanders, L. Prechelt, and M. Müller. SKaMPI: A detailed, accurate MPI benchmark. In *EuroPVM/MPI*, number 1497 in LNCS, pages 52–59, 1998.

[San93] P. Sanders. A case study in object oriented programming: Algebraic data structures in Eiffel. In R. Ege, M. Singh, and B. Meyer, editors, *Technology of Object-Oriented Languages and Systems*, number 11 in TOOLS, pages 379–388, Santa Barbara, 1993. Prentice Hall.

[San94a] P. Sanders. A detailed analysis of random polling dynamic load balancing. In *International Symposium on Parallel Architectures, Algorithms and Networks*, pages 382–389, Kanazawa, Japan, 1994. IEEE, Los Alamitos, CA.

[San94b] P. Sanders. Emulating MIMD behavior on SIMD machines. In *International Conference Massively Parallel Processing Applications and Development*, pages 313–321, Delft, 1994. Elsevier.

[San95a] P. Sanders. Better algorithms for parallel backtracking. In *Workshop on Algorithms for Irregularly Structured Problems*, number 980 in LNCS, pages 333–347, Lyon, France, 4–6 September, 1995. Springer-Verlag Berlin.

[San95b] P. Sanders. Fast priority queues for parallel branch-and-bound. In *Workshop on Algorithms for Irregularly Structured Problems*, number 980 in LNCS, pages 379–393, Lyon, 1995. Springer.

[San96] P. Sanders. *Lastverteilungsalgorithmen für parallele Tiefensuche*. Dissertation, Universität Karlsruhe, 1996.

[San97] P. Sanders. *Load Balancing Algorithms for Parallel Depth First Search (In German: Lastverteilungsalgorithmen für parallele Tiefensuche)*. Number 463 in Fortschrittsberichte, Reihe 10. VDI Verlag, Berlin, 1997.

[San98a] P. Sanders. Randomized priority queues for fast parallel access. *Journal Parallel and Distributed Computing, Special Issue on Parallel and Distributed Data Structures*, 49:86–97, 1998.

[San98b] P. Sanders. Tree shaped computations as a model for parallel applications. In *ALV'98 Workshop on Application Based Load Balancing*, number TUM-INFO-02-I9806-100/1.-FI in TR TU Munich, Germany, Munich, Germany, 25–26 March, 1998. SFB 342. `http://www.mpi-sb.mpg.de/~sanders/papers/alv.ps.gz`.

[San99a] P. Sanders. Accessing multiple sequences through set associative caches. In *ICALP*, number 1644 in LNCS, pages 655–664, 1999.

[San99b] P. Sanders. Asynchronous random polling dynamic load balancing. In *ISAAC: 10th International Symposium on Algorithms and Computation*, number 1741 in LNCS, pages 37–48, Chennai, India, 16–18 December, Dec. 1999. Springer-Verlag Berlin.

[San99c] P. Sanders. Fast priority queues for cached memory. In *ALENEX '99, Workshop on Algorithm Engineering and Experimentation*, number 1619 in LNCS, pages 312–327. Springer, 1999.

[San00a] P. Sanders. Asynchronous scheduling of redundant disk arrays. In *12th ACM Symposium on Parallel Algorithms and Architectures*, pages 89–98, 2000.

[San00b] P. Sanders. Fast priority queues for cached memory. *ACM Journal of Experimental Algorithmics*, 5, 2000.

[San02a] P. Sanders. Presenting data from experiments in algorithmics. In *Experimental Algorithmics — From Algorithm Design to Robust and Efficient Software*, volume 2547 of *LNCS*, pages 181–196. Springer, 2002.

[San02b] P. Sanders. Reconciling simplicity and realism in parallel disk models. *Parallel Computing*, 28(5):705–723, 2002. Special Issue on Parallel Data Intensive Algorithms and Applications.

[San02c] Peter Sanders. Randomized receiver initiated load balancing algorithms for tree shaped computations. *The Computer Journal*, 45(5):561–573, 2002.

[San03a] P. Sanders. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*, chapter 1: Memory Hierarchies — Models and Lower Bounds, pages 1–13. Springer, 2003.

[San03b] P. Sanders. Asynchronous scheduling of redundant disk arrays. *IEEE Transactions on Computers*, 52(9):1170–1184, 2003.

[San04a] P. Sanders. Algorithms for memory hierarchies. *EATCS Bulletin*, 83:67–85, June 2004.

[San04b] P. Sanders. Algorithms for scalable storage servers. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *LNCS*, pages 82–101. Springer, 2004.

[SEK00] P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium on Discrete Algorithms*, pages 849–858, 2000.

[SEK03] P. Sanders, S. Egner, and Jan Korst. Fast concurrent access to parallel disks. *Algorithmica*, 35(1):21–55, 2003.

[SF00] P. Sanders and R. Fleischer. Asymptotic complexity from experiments? a case study for randomized algorithms. In *Workshop on Algorithm Engineering*, number 1982 in LNCS, pages 135–146, 2000.

[SH97] P. Sanders and T. Hansch. On the efficient implementation of massively parallel quicksort. In G. Bilardi, A. Ferreira, R. Lüling, and J. Rolim, editors, *4th International Symposium on Solving Irregularly Structured Problems in Parallel*, number 1253 in LNCS, pages 13–24. Springer, 1997.

[Sha81] M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers and Mathematics with Applications*, 7(1):67–72, 1981.

[SS00] P. Sanders and J. Sibeyn. A bandwidth latency tradeoff for broadcast and reduction. In *6th Euro-Par*, number 1900 in LNCS, pages 918–926, 2000.

[SS03] P. Sanders and J. Sibeyn. A bandwidth latency tradeoff for broadcast and reduction. *Information Processing Letters*, 86(1):33–38, 2003.

[SS05] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *13th European Symposium on Algorithms (ESA)*, volume 3669 of *LNCS*, pages 568–597. Springer, 2005.

[SS06] P. Sanders and D. Schultes. Engineering highway hierarchies. In *14th European Symposium on Algorithms (ESA)*, volume 4168 of *LNCS*, pages 804–816, 2006.

[SS07] P. Sanders and J. Singler. MCSTl: The multi-core standard template library. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2007. accepted (short paper).

[SSO00] P. Sanders and R. Solis-Oba. How helpers hasten h-relations. In *8th European Symposium on Algorithms (ESA)*, number 1879 in LNCS, pages 392–402. Springer ©, 2000.

[SSO01] P. Sanders and R. Solis-Oba. How helpers hasten h-relations. *Journal of Algorithms*, 41:86–98, 2001.

[ST02] Peter Sanders and Jesper Larsson Träff. The factor algorithm for regular all-to-all communication on clusters of SMP nodes. In *8th Europar*, number 2400 in LNCS, pages 799–803. Springer, 2002.

[ST06] P. Sanders and J. Larsson Träff. Parallel prefix (scan) algorithms for MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 13th European PVM/MPI Users' Group Meeting*, volume 4192 of *LNCS*, pages 49–57. Springer, 2006.

[ST07] P. Sanders and F. Transier. Fast intersection in randomized inverted indices. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007.

[SV87] J. T. Stasko and J. S. Vitter. Pairing heaps: experiments and analysis. *Comm.A.C.M.*, 30(3):234–249, March 1987.

[SVW97] Peter Sanders, Roland Vollmar, and Thomas Worsch. Feasible models of computation: Three-dimensionality and energy consumption. Technical report 2/97, Universität Karlsruhe, Fakultät für Informatik, 1997.

[SW04] P. Sanders and S. Winkel. Super scalar sample sort. In *12th European Symposium on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 784–796. Springer, 2004.

[Tar72] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[Tec06] ClearSpeed Technology. CSX processor architecture. Technical report, 2006. `http://www.clearspeed.com/docs/resources/ClearSpeed_Architecture_Whitep%aper_0611.pdf`.

[Tho00] Mikkel Thorup. Even strongly universal hashing is pretty fast. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–497, N.Y., January 9–11 2000. ACM Press.

[Tho04] M. Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *J. Comput. Syst. Sci.*, 69(3):330–353, 2004.

[TZ03] Philippas Tsigas and Yi Zhang. A simple, fast parallel implementation of quicksort and its performance evaluation on SUN enterprise 10000. In *PDP*, pages 372–381. IEEE Computer Society, 2003.

[Val94] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8), 1994.

[vEB77] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. *Information Processing Letters*, 6(3):80–82, 1977.

[VS94a] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, I: Two level memories. *Algorithmica*, 12(2/3):110–147, 1994.

[VS94b] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, II: Hierarchical multilevel memories. *Algorithmica*, 12(2/3):148–169, 1994.

[WACV02] R. Wickremesinghe, L. Arge, J. S. Chase, and J. S. Vitter. Efficient sorting using registers and caches. *ACM Journal of Experimental Algorithmics*, 7(9), 2002.

[Wen92] Michael Wenzel. Wörterbücher für ein beschränktes Universum (dictionaries for a bounded universe). Master's thesis, Saarland University, Germany, 1992.

[Wil64] J. W. J. Williams. Algorithm 232 (HEAPSORT). *Communications of the ACM*, 7:347–348, 1964.

[YH02] M. Thorup Y. Han. Integer sorting in $\mathcal{O}\left(n\sqrt{\log\log n}\right)$ expected time and linear space. In *42nd Symposium on Foundations of Computer Science*, pages 135–144, 2002.