

Project Performance Engineering: Final Report

Institution: Department of Computing, University of Copenhagen (DIKU)
Project duration: 1.1.1999–31.1.2002 (extended with one month)
Principal investigator: Jyrki Katajainen, Assoc. Prof.
Other investigators: Jesper Bojesen, M. Sc., Medical Insight
Christopher Derek Curry, B. Sc.
Anders Sewerin Johansen, Ph. D. student, IT-C
Jeppe Nejsum Madsen, B. Sc., Array Technology A/S
Bjarke Buur Mortensen, B. Sc.
Sofus Mortensen, B. Sc., lambdasoft
Morten Nicolaj Pedersen, M. Sc., Cap Gemini
Maz Spork, M. Sc., sporkLAWÆTZ Advisors
Hans-Henrik Stærfeldt, M. Sc., CBS, DTU
Lars Yde, M. Sc.

Executive summary

In performance engineering the goal is to improve the running time and space efficiency of programs. In the algorithmics community the subject is called *algorithm engineering*, but we prefer the term *performance engineering* since the study objects are programs, not algorithms. At the end of 1998, the undersigned and his graduate students received a grant from the Danish Natural Science Council under contract 9801749 (project Performance Engineering¹), which was used to found the performance engineering laboratory² (PE-lab) at the Department of Computing, University of Copenhagen.

The activities of the PE-lab have been diverse ranging from performance engineering, and algorithmics, to software development in general. The organization of the laboratory is project-oriented. Of the projects initiated the following two should be mentioned.

CPH STL: Our goal is to develop an enhanced version of the Standard Template Library (STL), which is part of the ISO standard for the C++ programming language³. The project was started in autumn 2000, and the first phase of the project, in which the requirement analysis was carried out and prototypes for various components were built, was

¹ The original application is available at [http://www.diku.dk/hjemmesider/ansatte/jyrki/Course/Performance-Engineering-1998/SNFapplication\(1998\).ps](http://www.diku.dk/hjemmesider/ansatte/jyrki/Course/Performance-Engineering-1998/SNFapplication(1998).ps).

² <http://www.diku.dk/research-groups/performance-engineering/>

³ International Organization for Standardization (ISO), *ISO/IEC 14882: Standard for the C++ Programming Language* (1998).

completed in autumn 2001. More information about the CPH STL is available at the Internet⁴.

Peerview: Peerview is a prototype tool supporting intellectual collaborative work through artifact browsing and group review. The project was finished in summer 2001, and the Java source code was made publicly available. For more information, see PeerView's homepage⁵.

The main purpose to raise funds was to buy our students out from their workplaces, get them involved in the research activities of the department, and this way get them to finish their studies. In this respect the project was successful. In total, nine M. Sc. theses and one Ph. D. thesis were completed under the supervision of the principal investigator over the past three years. Several other written projects were accomplished. In most of the student projects the main concern was performance.

Even though the original idea was to concentrate on the implementation issues of computer algorithms, the students seem to direct our efforts towards software development. Our plan is to extend the PE-lab to a software-development laboratory.

Grant

The total funding provided by the Danish Natural Science Council for the project was 288 000 DKK, evenly distributed over the years 1999, 2000, and 2001. The grant was used in accordance with the application. The main budget item was the salaries for research assistants. The following persons were employed at the PE-lab:

Morten Nicolaj Pedersen, 16.3.1999–15.6.1999, algorithmist

Lars Yde, 1.5.2000–31.10.2000, software engineer, and 1.2.2001–30.4.2001, teaching assistant

Jesper Bojesen, 1.8.2000–31.1.2001, performance engineer (own financing)

Christopher Derek Curry, 1.11.2001–31.1.2002, research assistant

Among the other budget items were costs associated with participation in conferences and meetings, the reimbursement of costs borne by our guests, the purchase of relevant books and journals, the acquisition of the `cphstl.dk` computer, and the organization of three workshops.

The PE-lab was visited by [Tomi Pasanen](#) (Post. Doc.) in 2001 (total two months). For all short-term visits, one gets a good picture by looking at the announcements of the talks and presentations hosted by the PE-lab⁶.

The PE-lab organized three workshops, internal for the CPH STL project group⁷.

⁴ <http://www.cphstl.dk/>

⁵ <http://www.diku.dk/research-groups/performance-engineering/PeerView/>

⁶ <http://www.diku.dk/research-groups/performance-engineering/Talks/2000.html>
<http://www.diku.dk/research-groups/performance-engineering/Talks/2001.html>

⁷ 1st workshop: <http://www.diku.dk/undervisning/2000e/e00.505/#workshop>

2nd workshop: <http://www.diku.dk/undervisning/2001f/f01.650/#workshop>

3rd workshop: <http://www.diku.dk/undervisning/2001e/e01.718/3rd-STL-workshop.html>

Results obtained

In this section we give an overview of the results obtained. In the forthcoming sections, individual investigators give their personal view of the research carried out. The ordering of the sections is alphabetical. At the end there is a list of the publications produced. The publications are divided into three categories: P) scientific papers, R) technical reports, and T) theses supervised by the principal investigator.

In our original research plan¹ we decided to attack the performance-tuning issues from five different angles. Below we briefly discuss the results in each of the projections considered.

Reducing instruction count: In the beginning we enthusiastically studied all the code-tuning tricks introduced, for instance, in the exercises of Knuth's books⁸. However, very soon we realized that the efficiency of a program cannot be improved much using traditional code-tuning techniques. This is apparent, e.g., in the paper [P3] even though the result was not explicitly mentioned.

Since in contemporary computers the cost of different instructions varies so much, the simple instruction count is not a reliable performance measure. In [T1,P2,P3] the pure-C model of Katajainen and Träff⁹ was extended to include the costs associated with cache misses, and in [T8] the costs associated with branch mispredictions. Also, in [T8] an attempt was made to handle instruction-level parallelism, but at the moment we cannot handle this satisfactorily.

Utilizing word parallelism: In theory community there has been considerable interest in algorithms utilizing bitwise operations efficiently. For instance, it has been shown that n integers can be sorted in $O(n \log_2 \log_2 n)$ time on a word RAM¹⁰. The method has even been claimed to be practical¹¹. The experiments reported in [T3] show the opposite: the modern sorting algorithms rely heavily on random-access capabilities resulting in a poor cache behaviour. The best implementation of a modern method was a factor of 7–8 times slower than a quicksort-based implementation available in the C++ standard library.

Improving cache behaviour: Many of our early experiments pointed out the importance of the locality of memory references. For instance, in our mergesort study⁹ much of the performance boost was due to the cache behaviour even though we did not realize this at the time we wrote the paper (1996). This became apparent in later studies [T1,T8].

⁸ Donald E. Knuth, *The Art of Computer Programming*, Volumes 1–3, Addison Wesley Longman (1968–1998)

⁹ Jyrki Katajainen and Jesper Larsson Träff, [A meticulous analysis of mergesort programs](#), *Proceedings of the 3rd Italian Conference on Algorithms and Complexity, Lecture Notes in Computer Science* **1203**, Springer-Verlag (1997), 217–228

¹⁰ Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman, [Sorting in linear time?](#), *Journal of Computer System Sciences* **57** (1998), 74–93

¹¹ Stefan Nilsson, [The fastest sorting algorithm?](#), *Dr. Dobb's Journal* **311** (2000), 38–45

In [P1,P2,P3] we analysed the cache behaviour of some well-known algorithms. These papers show that cache-conscious algorithms can be very different in their design compared to algorithms designed for the word RAM — although methods based on the divide-and-conquer paradigm seem to work exceptionally well. In the underlying cost model we assumed that caches are fully associative and incorporate the least-recently-used cache-block replacement policy. Afterwards it is interesting to compare our approach to the ideal-cache model used in an independent work by Prokop and his collaborators¹² done at about the same time as ours.

Improving paging performance: Recently, there has been lot of progress in the area of external-memory algorithms (see, e.g., Vitter’s survey¹³). Our efforts have concentrated on efficient priority queues [P4,P6]. The methods have optimal paging behaviour, but a recent experimental study shows¹⁴ that the bottleneck in many external-memory priority queues is their weak CPU behaviour, so this deserves a further study.

Utilizing computer parallelism: An implementation of a distributed shared-memory system was described in [T4]. The algorithmic issues in distributed computing are important, but at the moment the area is not known well enough by our group, so more resources must be invested before this line of research can be continued.

Jesper Bojesen

In my work on performance engineering the main focus was on understanding the impact that the — possibly implicit — cost model used by the programmer has on the resulting algorithms. This resulted in two case studies:

- The problem of interchanging two array segments in place was studied. Three methods, available at the Silicon Graphics Inc. implementation of the STL (SGI STL), were analysed in a hierarchical memory environment, and two new programs were developed. One program is a simple variation of one of the SGI STL programs, while the other is a complicated adaptation of another of the SGI STL programs. Interestingly, even though all programs were predicted to be about equally fast when measured in traditional means, the slowest program was more than 30 times slower than the fastest on real hardware. Moreover, the supposedly fastest of the programs was actually the slowest. This work was presented at the *4th Workshop on Algorithm Engineering* in 2000 [P1].

¹² Harald Prokop, [Cache-oblivious algorithms](#), M.Sc. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1999), 70 pp.

¹³ Jeffrey Scott Vitter, [External memory algorithms and data structures: Dealing with massive data](#), *ACM Computing Surveys* **33** (2001), 209–271

¹⁴ Klaus Brengel, Andreas Crauser, Paolo Ferragina, and Ulrich Meyer, An experimental study of priority queues in external memory, *The ACM Journal of Experimental Algorithms* **5** (2000), Article 17

- The problem of constructing an implicit binary heap was considered in detail. The starting point for this study was the original programs by Williams and Floyd, but also a recent method by Fadel et al. [P6] was considered. Each of the programs was modified to take the memory hierarchy into account. However, in this study, the cost model employed was specified in much more detail, and followed rigorously. The cost model, weighted pure C, is an variation of the word-RAM model, extended with a penalty when a cache miss is incurred. This work was presented at the *3rd Workshop on Algorithm Engineering* in 1999 [P2], after which it was invited to be included in the *ACM Journal of Experimental Algorithmics* in a special issue devoted to the best papers presented at the workshop [P3].

Jyrki Katajainen

Over the past three years my research has, in one way or another, been related to the impact of the memory hierarchy on the performance of programs. In applications that process massive amounts of data, the main concern is to minimize the input/output communication between fast internal memory and slow external memory such as disks. My work on external-memory algorithms has mainly been theoretical. When the computational problem under consideration is small enough to be handled inside internal memory, the underlying problem is similar but in a smaller scale. Here the main concern is to improve the locality of memory references in order to utilize the cache memories as efficiently as possible. My work on the efficiency of programs in a hierarchical memory system has mainly been practical.

A new space- and time-efficient data structure for double-ended queues was developed for the CPH STL [P9,R9]. Also, space-economical algorithms for the selection [P11,R15] and convex-hull [P5] problems were devised. A modification of the selection algorithm was implemented as part of the CPH STL [R21], whereas in the convex-hull algorithms the routines of the library are extensively used.

Jeppe Nejsum Madsen

The main contribution of the report [R18] is a detailed description and complexity analysis of a heuristic to compress relational tables. This heuristic was previously only informally described, with no complexity results. Furthermore we have implemented the compress algorithm, and a join algorithm that works on compressed relations. These implementations have been benchmarked against existing implementations. The results obtained, show the algorithms presented to be faster than any previous implementations for the data sets used.

In [P8] we study the behaviour of an algorithm which compresses relational tables by representing common subspaces as Cartesian products. The

output produced allows space to be saved while preserving the functionality of many relational operations such as select, project and join. We describe an implementation of an existing algorithm, propose a slight modification which with high probability produces the same output, and present a performance study showing that for all test instances used both adaptations are considerably faster than the current implementation in a commercial software product.

Bjarke Buur Mortensen

My involvement has been with the CPH STL project. I have been concerned with space-efficient double-ended queues, known as dequeues. In a first project [R6] we investigated this by exploring and combining the ideas presented in previous literature. We showed that it was possible to improve some of the core functions of the data structure while also improving space efficiency. Operations concerned with inserting and erasing element in the middle of the data structure was improved by an order of magnitude compared to an existing implementation. This was achieved by using circular arrays as the basic building blocks for element storage. Furthermore, through performance tuning we significantly reduced the running time for random element access. Other core operations, insertion and removal of elements in both ends of the deque, were slower than the existing implementation.

In a second project [R9] we presented a new realization of a space-efficient deque which is constructed from three singly-resizable arrays. Our data structure is space efficient and fulfills the time requirements of the C++ standard. The structure presented is not fully standards compliant, but neither is any existing implementation we know of. In fact, our data structure fixes an important aspect of standards compliance by introducing a minor (in our view) non-compliance issue. Furthermore, it provides the same improvements in measured performance as the data structure implemented in the first project. Again, insertion and removal in the ends were slower. This work was presented at the *5th Workshop on Algorithm Engineering* [P9].

The two projects have shown how to build a fast deque that is also space efficient. It is also clear from the two projects how to improve the time efficiency of existing deque implementations, e.g., the one that exists in the SGI STL. This should be done by 1) using circular arrays as building blocks for element storage, which will improve element insertion and removal from the middle of the data structure by an order of magnitude, and 2) using performance engineering to improve performance for random access operations. This task is marked as future work for the CPH STL project.

Morten Nicolaj Pedersen

In recent years researchers have taken an interest in the word-RAM machine model, which better reflects the capabilities of contemporary computers than

the somewhat dated comparison-based model. Briefly, the word RAM is a random-access machine with an unlimited number of memory cells containing λ -bit integers, which supports (at least) comparison, addition, left and right shifts, and bitwise boolean instructions. My thesis [T3] is a study on the practical significance of word-RAM algorithms for integer sorting.

The modern sorting algorithms are generally designed as follows:

- 1) Use range reduction to reduce the problem of sorting a sequence of n b -bit keys, $b \leq \lambda$, to the problem of sorting (at most) n keys from a smaller range.
- 2) Solve the reduced problem of sorting short keys by packing several keys in one machine word in order to exploit the inherent bitwise word-level parallelism of machine instructions. Call this packed sorting.
- 3) Use the solution of the reduced problem to solve the original problem.

This range expansion complements the range reduction. Such algorithms are able to sort n keys in $o(n \log n)$ time on a word RAM, which is an improvement over all the classical radix-sorting and comparison-based algorithms. The modern algorithms are therefore theoretically important, but are they useful in practice? I tried to answer this question.

I gave a theoretical survey of word-RAM algorithms for internal integer sorting, including radix-sorting algorithms. From a discussion of the practical value of the research presented, and a summary of existing empirical evaluations of it, it became evident that packed sorting has no potential practical significance on the computers currently mass-produced ($\lambda = 64$). I therefore chose to experiment with exponential range reduction and expansion combined with Adaptive Radixsort; an algorithm called KR Sort.

In comparison with programs, which are considered efficient, I found my careful implementation of KR Sort to perform unsatisfactorily. The comparison was made on three performance indicators, running time, number of cache misses, and number of instructions executed, since that is required to reason about the causes of the behaviour of a program. From the test results one could conclude that KR Sort loses due to bad memory behaviour; its implementation uses more space than the competing programs, and the memory access pattern is inherently random. This results in an excessive number of cache misses, which are expensive on a real computer. The word-RAM model does not reflect this fact, and that is perhaps its biggest flaw.

Maz Spork

In my final year I worked on experimental algorithmics and cache-conscious programs. Experimental algorithmics concerns models that accurately describe the interaction between software and hardware. Algorithmics is usually considered the top-level abstraction in computer science, with computational semantics below it. Performance-tuning and compiler-optimization techniques are typically at the lowest level. Computing machinery has physical characteristics which interact with the software, and properties which

are possible to describe formally. Examples of such properties of microarchitectures are instruction-level parallelism, deep pipelines, and hierarchical memory subsystems. Mostly, the formal models are concerned with latencies induced indirectly by instruction streams or memory reference patterns.

My work was specifically on the interaction between applications of sorting and searching, and the higher levels of the memory hierarchy. The main results were refinements of a unit cost model for incorporating constants to complexities in a meaningful way, pure C with a memory reference latency. The cost model was applied to heap construction and stable mergesort.

Finally, I described a theory and a framework for constructing sequential-access algorithms for use on a speculative computer able to deliver data with zero-latency in “strides” of access, and applied this theory on the problem of computing the connected components of an undirected graph.

Lars Yde

My efforts under the auspices of the PE-lab were two-fold:

- Partaking in and co-administering the efforts undertaken under the CPH STL heading, namely conveying — through teaching and symposia — the ideas and intentions underlying the project.
- Developing experimental software to spearhead (ultimately, if not currently) a new approach to artifact browsing in a shared workspace. So far, this initiative has generated only sporadic attention although the prototype software — called PeerView — is a fully working application, albeit without the industrial strength scalability needed to go much beyond laboratory conditions. Plans, although tentative, are to remedy this when conditions are more benign to the development and launch of a commercial application based on the ideas propounded by PeerView and the M. Sc. thesis underlying it [T9]. The specifics of this future venture are also to be found in the aforementioned thesis, and are thus in the public domain, boosting the chances of future efforts spawned by this work.

In practice, this two-pronged effort has involved equal parts manual and intellectual labour, driving home (to me at least) the point that software development in practice is subject to the same economic and sociological determinants as more institutionalized forms of manual labour. In itself, this beckons a shift in modes of teaching software development vis-à-vis the established forms prevalent in universities and colleges. This realization was also the driving force behind plans to establish a permanent software development facility within the PE-lab; plans that I helped explore and develop together with Jyrki Katajainen, but which have so far failed to capture the attention of those responsible for the coffers that can lend this vision thrust. Hopefully, the array of scientific and developmental achievements generated by PE-lab associates can help pave the way for a more attentive reception of future petitions.

Publications

Scientific papers

- [P1] Jesper Bojesen and Jyrki Katajainen, Interchanging two segments of an array in a hierarchical memory system, in *Proceedings of the 4th International Workshop on Algorithm Engineering*, *Lecture Notes in Computer Science* **1982**, Springer-Verlag (2000), 159–170
- [P2] Jesper Bojesen, Jyrki Katajainen, and Maz Spork, Performance engineering case study: heap construction, in *Proceedings of the 3rd International Workshop on Algorithm Engineering*, *Lecture Notes in Computer Science* **1668**, Springer-Verlag (1999), 301–315
- [P3] Jesper Bojesen, Jyrki Katajainen, and Maz Spork, Performance engineering case study: heap construction, *The ACM Journal of Experimental Algorithms* **5** (2000), Article 15
- [P4] Gerth Stølting Brodal and Jyrki Katajainen, Highly flexible nodes in external heaps, unpublished typescript (2000)
- [P5] Hervé Brönnimann, John Iacono, Jyrki Katajainen, Pat Morin, Jason Morrison, and Godfried Toussaint, In-place planar convex hull algorithms, *Proceedings of the 5th Latin American Theoretical Informatics Symposium*, *Lecture Notes in Computer Science* **2286**, Springer-Verlag (2002)
- [P6] R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola, Heaps and heapsort on secondary storage, *Theoretical Computer Science* **220** (1999), 345–362
- [P7] Viliam Geffert, Jyrki Katajainen, and Tomi Pasanen, Asymptotically efficient in-place merging, *Theoretical Computer Science* **237** (2000), 159–181
- [P8] Jyrki Katajainen and Jeppe Nejsum Madsen, Performance tuning an algorithm for compressing relational tables, *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, *Lecture Notes in Computer Science*, Springer-Verlag (2002)
- [P9] Jyrki Katajainen and Bjarke Buur Mortensen, Experiences with the design and implementation of space-efficient dequeues, in *Proceedings of the 5th Workshop on Algorithm Engineering*, *Lecture Notes in Computer Science* **2141**, Springer-Verlag (2001), 39–50
- [P10] Jyrki Katajainen and Tomi A. Pasanen, In-place sorting with fewer moves, *Information Processing Letters* **70** (1999), 31–37
- [P11] Jyrki Katajainen and Tomi A. Pasanen, A randomized in-place algorithm for positioning the k th element in a multiset, *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, *Lecture Notes in Computer Science*, Springer-Verlag (2002)

Technical reports

The ISSN of the CPH STL Report Series is 1602-1150. All the CPH STL reports are available at <http://www.cphstl.dk/>.

- [R1] Jyrki Katajainen and Lars Yde (Editors), Project proposal: the Copenhagen STL, CPH STL Report **2000-1**, Web Document (2000), 5 pp.
- [R2] Lars Yde and Jyrki Katajainen, Supporting intellectual work through artifact rendering and group review, DIKU Report **2000-11**, Department of Computing, University of Copenhagen (2000), 14 pp.
- [R3] Jyrki Katajainen and Kimmo E. E. Raatikainen, Instructions to use DIKU style files, CPH STL Report **2001-1**, Web document (2001), 7 pp.

- [R4] Simon Thamdrup Jensen, [Random_shuffle\(\) in the Copenhagen STL](#), CPH STL Report **2001-2**, Web Document (2001), 19 pp.
- [R5] Brian S. Jensen, [Priority queue and heap functions](#), CPH STL Report **2001-3**, Web Document (2001), 22 pp.
- [R6] Bjarke Buur Mortensen, [The deque class in the Copenhagen STL: First attempt](#), CPH STL Report **2001-4**, Web Document (2001), 46 pp.
- [R7] Christian Boesgaard and Jacob Chr. Poulsen, [Copenhagen STL — hash map](#), CPH STL Report **2001-5**, Web Document (2001), 33 pp.
- [R8] Jacob Gorm Hansen and Asger Kahl Henriksen, [The \(multi\)?\(map|set\) of the Copenhagen STL](#), CPH STL Report **2001-6**, Web Document (2001), 20 pp.
- [R9] Jyrki Katajainen and Bjarke Buur Mortensen, [Experiences with the design and implementation of space-efficient deques](#), CPH STL Report **2001-7**, Web document (2001), 47 pp.
- [R10] Philip Bille, [A simple implementation of set algorithms for the STL](#), CPH STL Report **2001-8**, Web Document (2001), 7 pp.
- [R11] Steffen Nissen, [Permutation algorithms in the Copenhagen STL](#), CPH STL Report **2001-9**, Web Document (2001), 19 pp.
- [R12] Henrik Skovby, [Implementation of a circular singly linked list: slist](#), CPH STL Report **2001-10**, Web Document (2001), 14 pp.
- [R13] Jeppe Nejsum Madsen, [bitset<N> in the Copenhagen STL](#), CPH STL Report **2001-11**, Web Document (2001), 8 pp.
- [R14] Magdalena “Lenka” Otap, [Webdesign for Copenhagen STL](#), CPH STL Report **2001-12**, Web Document (2001), 86 pp.
- [R15] Jyrki Katajainen and Tomi A. Pasanen, [A randomized in-place algorithm for positioning the kth element in a multiset](#), CPH STL Report **2001-13**, Web document (2001), 11 pp.
- [R16] Jesper Holm Olsen and Søren Skov, [A comparative analysis of three different priority deques](#), CPH STL Report **2001-14**, Web Document (2001)
- [R17] Jyrki Katajainen (Editor), [Research proposal: software tools for program library development](#), CPH STL Report **2001-15**, Web document (2001), 7 pp.
- [R18] Jeppe Nejsum Madsen, [Algorithms for compressing and joining relations](#), CPH STL Report **2002-1**, Web Document (2002), 101 pp.
- [R19] Jesper Nielsen, [CPH STL designdokumenter på webben](#), CPH STL Report **2002-2**, Web Document (2002), 7 pp.
- [R20] Claus Jensen, [Webbaseret logokonkurrence](#), CPH STL Report **2002-3**, Web Document (2002), 43 pp.
- [R21] Lars Yde, [Performance engineering the nth_element function](#), CPH STL Report **2002-4**, Web Document (2002), 58 pp.
- [R22] Jyrki Katajainen (Editor), [Project performance engineering: final report](#), CPH STL Report **2002-5**, Web Document (2002), 11 pp.

Theses supervised by the principal investigator

Most of the theses are available at <http://www.diku.dk/research-groups/performance-engineering/Perfeng/theses.html>.

- [T1] Maz Spork, [Design and analysis of cache-conscious programs](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (1999), vii+102 pp. 40 app. pp.
- [T2] Tomi Pasanen, [In-place algorithms for sorting problems](#), Ph.D. Thesis, Department of Computer Science, University of Turku (1999), 105 pp.

- [T3] Morten Nicolaj Pedersen, [A study of the practical significance of word RAM algorithms for internal integer sorting](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (1999), ii+74 pp. 77 app. pp.
- [T4] Anders Sewerin Johansen, [SENSE — delt lager på netværk af arbejdsstationer](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (1999), 79 pp. 111 app. pp.
- [T5] Adam Arndt, [Anvendelse af internet til intellektuelt samarbejde](#), M. Sc. Thesis, The IT University of Copenhagen (2000), 99 pp. 10 app. pp.
- [T6] Jesper Bojesen, [Managing memory hierarchies](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (2000), iv+93 pp. 5 app. pp.
- [T7] Rasmus Borch, [SQL99s objektmodel: teori og praksis](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (2001), vi+108 pp. 5 app. pp.
- [T8] Sofus Mortensen, [Refining the pure-C cost model](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (2001), viii+86 pp. 33 app. pp.
- [T9] Lars Yde, [Supporting intellectual work through rendering and review](#), M. Sc. Thesis, Department of Computing, University of Copenhagen (2001), 77 pp. 423 app. pp.
- [T10] Hans-Henrik Stærfeldt, Protein-protein interaction: Heuristic search in databases of 3d protein structures, M. Sc. Thesis, Department of Computing, University of Copenhagen (2002), 54 pp. 106 app. pp.

On behalf of the project group

Copenhagen, 30 April 2002

Jyrki Katajainen
Assoc. Prof., Docent, Ph. D.