Memory-hierarchy optimal matrix multiplication-programs

Riko Jacob

June 26, 2007

1 Abstract

In many applications that work on huge data sets, from areas like bioinformatics, data-mining, network analysis, optimization, and simulation, the computation time is a major concern. To shorten this time, we have to take into account the many aspects that determine the running time of a program on a modern computer. One of these aspects is the range of different type of memory, from fast but small to huge but slow. More precisely, there are registers on the CPU, various kinds of caches, main memory and disk, the levels of the memory hierarchy. The data transfer between these levels happens by means of so called I/O-operations that can be very slow, and should be minimized to achieve fast programs.

The core computation of many applications that require huge amounts of data can be modeled by sparse matrices, such that general purpose high performance software libraries for abstract operations can be used. One of the basic such operations is the multiplication of a huge sparse matrix with a vector. It has been recognized that this is one of the operations where modern computers operate significantly below their peak CPU-performance, indicating that indeed the data transfer in the memory hierarchy is the bottleneck.

In very recent work, we showed a lower bound on the number of data transfers that are necessary to compute the product of an arbitrary sparse matrix with a vector, and a sorting-based algorithm that is asymptotically optimal. Fortunately, in many applications the sparse matrices do have a structure that can be exploited to perform the multiplication with fewer I/O's.

The focus of this project is to (automatically) analyze the structure of a huge sparse matrix with respect to the amount of data transfer that is required to multiply with this matrix. In other words, for a given sparse matrix A we are interested in a program that transforms a vector x into the product Ax with the fewest possible I/O-operations.

2 Summary of the Project

The development of computer hardware over the last decades has been tremendous, mostly visibly in terms of CPU-speed and storage capacity. Part of this progress stems from organizing the memory in a hierarchical fashion, where the different kinds of memory implement different trade-offs between speed and size.

This ranges from the few registers of the CPU that are used at CPU-speed to the hard-disk that can store basically as many intermediate results as desired, but is very slow to access in comparison to the CPU-speed. Between these two extremes, there are other levels of this memory hierarchy, namely the level-1 and level-2 caches, and the main memory. In one way or the other, the slow speed of the large memories is due to a certain delay that is associated with chosing some arbitrary position of the memory. In contrast, the bandwidth to load a chunk of data is much closer to the processing speed of the CPU. To exploit this effect, the data transfer between the different kinds of memory is organized in blocks.

This situation naturally leads to the so-called I/O-model [1], where one level of the hierarchy is modeled as an internal memory that can hold M data-items, and an external memory that consists of infinitely many blocks of B items. In this model, a program can work only on the data-items in internal memory, or it can write a block of B items to the external memory, or read a block of B items from external memory. The performance of an algorithm is measured only in the number of I/O-operations (i.e. read and write) it performs and the operations in internal memory are not counted. Accordingly, this model focuses completely on the data-transfer between the levels of the memory, which is adequate for applications where this is the performance bottleneck, like huge databases that are stored on disk.

There is a rich literature on the performance of algorithms in the I/O-model, one of the first and most illustrative examples being sorting [1]. Here, the multi-way merge sort is known to be optimal, and in contrast to the CPU-based models of computation, it is not only optimal among the comparison based algorithms, but it is optimal for most reasonable values of M and B, even if the rank (resulting position) is known a priori for every element. This means that the input consists of elements together with their position in the output. The task is called permuting and is very easy to achieve with linearly many CPU-operations. Hence, in the I/O-model this seemingly innocent task leads to interesting considerations. For other computations, like the multiplication of two dense matrices, the floating point calculations contribute to the overall running time, but it is also important to minimize the data transfer in the memory hierarchy. Since there exist algorithms that are optimal (under some assumptions) or best known in terms of floating point operations, and simultaneaously optimal in terms of I/O, it is no surprise that these algorithms are also very good in practice. This practice is the world of linear algebra libraries, a very successful toolbox in scientific computing and other applications. Here, the number of CPU-operations can often be reduced significantly by exploiting that the matrices are sparse, i.e., only a small fraction of the entries is not zero.

One of the most basic linear algebra operations is the multiplication of a matrix with a vector. This is an interesting task because it is one of the cases where the results on permuting show that the I/O operations cannot decrease as much as the CPU-operations. This is consistent with the observation that sparse matrix programs operate significantly below the maximum floating point performance of the CPU.

In very recent work [2], we could show that the results about permutation matrices actually generalize to sparse matrices. Similarly to permuting, we found asymptotically I/O-optimal algorithms that are based on sorting to compute a general matrix-vector product, i.e., the situation where both the matrix and the vector are considered input. In the light of the lower bounds on the number of I/O's for general matrices, the reported performance of even the naive sparse matrix-vector multiplication software sounds impossible.

The explanation of this paradoxical situation is that we have been comparing apples with oranges. Even though the multiplication software can handle aribitrary sparse matrices, its performance is reported on matrices stemming from applications. Now, since the lower bound applies with high probability to a randomly selected sparse matrix, the paradoxon disappears if we assume that the matrices arising in practice are actually highly structured.

The focus of this research project is to quantify the above explanation of the paradoxon. As the first and prime example, we will consider an arbitrary but fixed sparse matrix A and ask for the I/O-optimal program that computes the product Ax for an arbitrary vector x. In this setting, the program is allowed (or required) to use the structure of the non-zero elements of A cleverly to minimize the number of I/O-operations. This question is not only natural, but it is also of practical interest.

The best outcome that we aim for is to find an I/O-efficient and CPU-efficient algorithm that on input A produces the optimal program to transform a vector x into Ax. This would immediately settle the question of how to compute such products in all applications.

One such application that has been considered from different angles is the computation of PageRank, as presumably done on a regular basis by the search engine Google. Here, some of the known structure of the matrix A that represents web-pages and hyperlinks is exploited to compute the matrix-vector product I/O-efficiently, since this is the inner loop of an iterative eigenvector computation. With full success, these considerations would be done automatically.

The above explained theoretical considerations always immediately lead to the question if they really capture the important aspects of such computations on modern computers. Accordingly, the second focus of the project is to test the theoretical results in practice. This will give important feedback to the theoretical analysis, for example to exclude certain unrealistic parameter settings from the analysis. It can also lead to more accurate models, and provide an intuition for novel ideas.

For a basic building block like sparse matrix-vector multiplication, this algorithms-engineering approach naturally starts from already existing collections of matrices from diverse applications. This approach promises to show the performance difference of the various algorithms and thus allows to find the practically relevant structural differences of the matrices that make a matrix easy or difficult for a particular algorithm.

3 Relation to Previous Research

3.1 State of Research

At the basis of our theoretical consideration is the I/O-model introduced in [1]. Since the problem of permuting a sequence of items can also be understood as multiplying by a permutation matrix, [1] also provides the first lower bound for Sparse-Matrix–Vector multiplication. This lower bound proves, that for memory size M and block size B of one level of the memory hierarchy that there exist permutations that require as many I/O's (up to a constant factor) as it takes to sort the elements. The goal of this project is to actually identify these difficult permutations.

In this I/O-model, the case of full matrices has been considered, and optimal algorithms are even known for the cache-oblivious model [7], i.e., the version of the I/O-model where the algorithm does not know about the parameters M and B of the model, and is analyzed for arbitrary such parameters. The lower bounds for matrix-multiplication go back to [10], where only the limit on the main-memory is considered, i.e., the case $B = 1$, and the computational task is specified as a circuit. In contrast to these results, this project is focused on sparse matrices where the block size plays a more important role.

There is an active line of research considering the computation of a sparse-matrix–vector product in a parallel setting, examples include [11, 3]. There are several connections between the I/O-model and parallel computing. The key difference to this project is that the transfer of data-items is ungrouped, i.e., there is no notion of a block or a track.

It has long been recognized that the memory hierarchy plays an important role in basic linear algebra computations. The particular difficulty of cache-efficiently computing sparse-matrix vector products is, for example, quantified in [16], where it is reported that for this task the CPU operated at roughly 10% of peak CPU speed, indicating that indeed the I/O is the bottleneck. There is a series of papers considering techniques like blocking of the matrix. This body of work is surveyed for example in [16, 5], and libraries of this kind are for example [16, 15, 9, 14, 12, 6].

There has been a lot of interest in efficiently computing PageRank, as mentioned in the introduction, see for example [4, 8]. This work tries to exploit the special structure of the web-graph, the concrete sizes of main-memory and the graph, and similar characteristics of the task at hand. This kind of work illustrates that it is indeed possible to algorithmically exploit structure of the input. Our ambitious goal is to complement such algorithms by lower bounds that also take the structure of the matrix into account, and ideally to find algorithmically the program inducing the least possible number of I/O-operations. This strict focus on the I/O-model and the dependence on the concrete web-graph matrix are certainly novel.

3.2 Own Previous Work

Initial Work on Random Sparse Matrices Together with Gerth Brodal (Aarhus, DK), Rolf Fagerberg (Odense, DK), Michael Bender (Stony Brooks, USA) and Elias Vicari (ETH Zurich), we conducted the already mentioned work on sparse matrix vector multiplication. Here, we define the parameter k as the average number of non-zero elements per column, i.e., the $N \times N$ matrix A has kN entries, the interesting values for k range between 1 and N. If A is stored in column major order, then computing the product with semiring operations only takes

$$\Theta\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{\max\{M,k\}}\right),\ kN\right\}\right)$$

I/O-operations, where the lower bound holds only for $k < \sqrt{N}$.

If the algorithm is free to choose the layout of the matrix (even if it has access to all entries of the matrix in some read-only memory), then the number of I/O-operations reduces to

$$\Theta\left(\min\left\{\frac{kN}{B}\left(1+\log_{M/B}\frac{N}{kM}\right)\,,\,kN\right\}\right)\,.$$

Here, the lower bound holds for $k \leq \sqrt[3]{N}$.

This cooperation started at the Dagstuhl seminar “Data Structures” in March 2006. The results are now written down as [2] and submitted for publication.

Vector–vector multiplication for non-distributive algebraic operations In joint work that is part of the PhD-project of Franz Roos [13], we consider the problem of searching measured mass-spectra of peptides in a huge set of possible peptides, either as given in a database or as all peptides stemming from a sequenced DNA. Abstractly, this task can be understood as computing the sum of all entries in a matrix given as the product of two vectors, where the algebraic operations are not distributive. In the setting of the I/O-model, where one measured peptide and a predicted peptide occupy precisely one cell, we give I/O-optimal algorithms that compute $M \cdot B$ comparisons between measured and predicted peptides per I/O.

So far, the focus is clearly on this single biological application. At this stage our current software outperforms some of the established standard software by roughly a factor 100, and is thus amongst the fastest programs for the task. A final comparison on the same data that we plan to conduct soon will quantify this comparison of different software.

Our preliminary, general conclusion is that the I/O-model is adequate to explain the performance of the software. Since a brute force approach of an all-against-all comparison is only feasible if a single comparison is not too CPU-demanding, cleverly scheduling the I/O's shows a visible improvement in running time. This work will appear soon as part of the PhD-thesis of Franz Roos.

References

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

[2] M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari. Optimal sparse matrix dense vector multiplication in the i/o-model. In *Proceedings 19th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 2007.

[3] R. H. Bisseling and W. Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*, 21:47–65, 2005. special issue on Combinatorial Scientific Computing.

[4] Y.-Y. Chen, Q. Gan, and T. Suel. I/o-efficient techniques for computing pagerank. In *CIKM*, pages 549–557. ACM, 2002.

[5] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, R. V. Antoine Petitet, R. C. Whaley, and K. Yelick. Self-adapting linear algebra algorithms and software. *Proceedings of the IEEE, Special Issue on Program Generation, Optimization, and Adaptation*, 93(2), February 2005.

[6] S. Filippone and M. Colajanni. Psblas: A library for parallel linear algebra computation on sparse matrices. *ACM Trans. on Math. Software*, 26(4):527–550, December 2000.

[7] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 285–297, New York, NY, October 17–19 1999.

[8] T. Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, Database Group, Computer Science Department, Stanford University, February 1999. Available at http://dbpubs.stanford.edu/pub/1999-31.

[9] E. J. Im. *Optimizing the Performance of Sparse Matrix-Vector Multiplication.* PhD thesis, University of California, Berkeley, May 2000.

[10] H. Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, New York, NY, USA, 1981. ACM Press.

[11] G. Manzini. Lower bounds for sparse matrix vector multiplication on hypercubic networks. *Discrete Mathematics & Theoretical Computer Science*, 2(1):35–47, 1998.

[12] K. Remington and R. Pozo. NIST sparse BLAS user's guide. Technical report, National Institute of Standards and Technology, Gaithersburg, Maryland, 1996.

[13] F. Roos. *Algorithms for peptide identification by tandem mass spectrometry.* PhD thesis, Eidgenssische Technische Hochschule ETH Zrich, 2006. Nr 16844.

[14] Y. Saad. Sparsekit: a basic tool kit for sparse matrix computations. Technical report, Computer Science Department, University of Minnesota, June 1994.

[15] R. Vudac, J. W. Demmel, and K. A. Yelick. *The Optimized Sparse Kernel Interface (OSKI) Library: User's Guide for Version 1.0.1b.* Berkeley Benchmarking and OPtimization (BeBOP) Group, March 15 2006.

[16] R. W. Vuduc. *Automatic Performance Tuning of Sparse Matrix Kernels.* PhD thesis, University of California, Berkeley, Fall 2003.