Cache-Optimal Data-Structures for Hierarchical Methods on Adaptively Refined Space-Partitioning Grids

Miriam Mehl

Institut für Informatik, TU München
Boltzmannstraße 3, D-85748 Garching, Germany
mehl@in.tum.de
http://www5.in.tum.de/persons/mehl/

Abstract. The most efficient numerical methods for the solution of partial differential equations, multigrid methods on adaptively refined grids, imply several drawbacks from the point of view of memory-efficiency on high-performance computer architectures: First, we loose the trivial structure expressed by the simple i, j-indexing of grid points or cells. This effect is even worsened by the usage of hierarchical data and – if implemented in a naive way – leads to both increased storage requirements (neighbourhoodrelations possibly modified difference stencils) and a less efficient data access (worse locality of data and additional data dependencies), in addition. Our approach to overcome this quandary between numerical and hardware-efficiency relies on structured but still highly flexible adaptive grids, the so-called space-partitioning grids, cell-oriented operator evaluations, and the construction of very efficient data structures based on the concept of space-filling curves. The focus of this paper is in particular on the technical and algorithmical details concerning the interplay between data structures, space-partitioning grids and space-filling curves.

1 Introduction

As soon as we want to simulate real world applications described by (systems of) partial differential equations, we are faced with two increasingly important requirements: First, there is a rising need for accuracy and, second, the computing time should stay within manageable borders. These requirements force us to reduce both the number of unknowns – typically by sophisticated grid adaptivity – and the complexity of our solver – typically with the help of multigrid methods. Unfortunately, both methods – grid adaptivity and multigrid – come along with several drawbacks. Adaptively refined grids in general imply a substantial overhead in terms of storage requirements due to the need to store neighbourhoud relations and/or specialised stencils for the particular local adaptivity pattern. But also with respect to the efficiency of data access, adaptively refined grids cause considerable deficits since in an irregular data structure it is much more difficult to bring together data dependencies with data locality in the physical memory space. Multigrid methods even worsen this effect as they induce additional data dependencies between data of different grid levels.

M. Gerndt and D. Kranzlmüller (Eds.): HPCC 2006, LNCS 4208, pp. 138–147, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In the following, we present our approach to overcome the quandary between numerical efficiency and efficient memory usage described above. Sect. 2 introduces the used grids. Sect. 3 describes the algorithm of our solver and Sect. 4 demonstrates the construction of suitable memory efficient data structures. Finally, we present results on the applicability and efficiency of our concept in Sect. 5.

2 Space-Partitioning Grids

As a first towards hardware efficiency, we use so-called space-partitioning grids which offer both a high flexibility in term of the adaptivity pattern and a strict structuredness which is already inherently storage saving as the storage of dependencies of grid elements becomes obsolete. Space-partitioning grids, mostly referred to in the form of octrees, are widely used as a tool to solve different subproblems in the simulation context, for example grid generation, geometry description, visualisation, load distribution, and as a computational grid. This shows the variety of functionalities and advantages of the underlying concept making space-partitioning grids a very attractive choice – also as an overall integrating concept in particular in larger context such as coupled simulations [4].

Fig. 1 shows examples for space-partitioning Cartesian grids. The construction principle is the recursive refinement of grid cells into a fixed number of congruent subcells. The number of subcells can be varied. In the case of the well-known quadtree (two-dimensional) or the octree (three-dimensional), a partitioning into two in each coordinate direction is performed. We instead perform a partitioning into three in each direction for reasons we will explain later on in the context of the construction of data structures (Sect. 4).

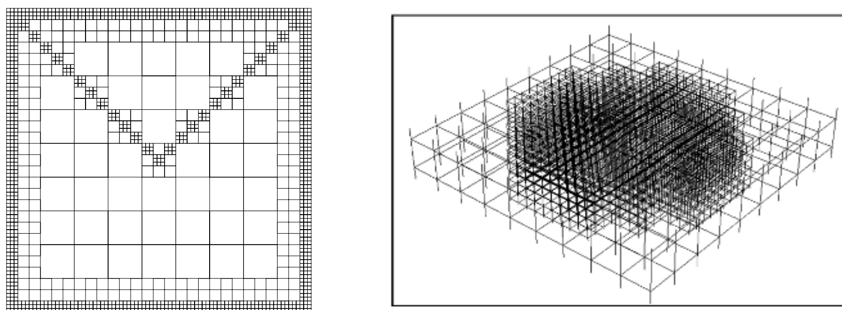

Fig. 1. Two- (left) and three-dimensional (right) examples for space-partitioning grids with a partitioning into three per coordinate direction

Arbitrarily local adaptive refinements can be mapped with space-partitioning grids[1]. The degrees of freedom are assigned to the vertices of the grid cells in our

[1] The only case we can not hadle without further efforts are unisotropic refinements. The generalisation of our concepts to unisotropic refinements is subject of our current work and in the implementation phase.

concept, but exceptions to this rule such as for the pressure in the Navier-Stokes equations [2,3] are possible. For reasons of clarity, we will restrict ourselves to vertex data in the following.

3 Algorithms

3.1 Operator Evaluation

Our main task – to achieve a high memory efficiency – requires the prevention of the storage of unnecessary informations and a high locality of data access (increasing the efficiency of the cache-usage). That is, we have to avoid the storage of neighbourhoodrelations and specialised difference stencils in dependence on the local adaptivity pattern, wich are, in our case, only caused by the vertex-oriented operator evaluation (processesing the grid vertex by vertex and using values at neighbouring vertices for operator evaluation) compute the value of the corresponding difference operator with the help of the values at neighbouring vertices. In addition, values at neighbouring verties might be arbitrarily far away from the current point in the physical memory space – and, therefore, are currently not in the cache with a high probability.

Thus, we switch to the so-called cell-oriented operator evaluation, a well-known concept in the finite element context [5]. It decomposes the operator into cell-parts which accumulate to the complete operator during a run over all cells contributing to the operator value. For the evaluation of the cell-part of an operator we allow only the usage of data owned to the cells, that is the degrees of freedom stored at the cells vertices. At boundaries between different refinements depths, the operator results from the accumulation of cell-parts of neighbouring cells appropriately combined with restriction and interpolation operators. Such, there is no need to store specialised stencils.

3.2 Multigrid

For the description of our multigrid methods, we only concentrate on the algorithmic realisation and leave out mathematical details as far as they have no algorithmic impacts.

Additive Multigrid. If we process our cell tree in the natural top-down-depth-first order, we cannot finish the evaluation of an operator at the current level before we switch to the next (coarser or finer) level, but we can compute the residual on all levels based on the same data. Thus, from the algorithmic point of view, the additive multigrid method is the natural choice: We interpolate all values to the finest level (dehierachisation) in the down-traversal (coarse to fine) of the cell tree, compute the cell-parts of the residual and smooth on the finest level, and restrict the fine grid residuals (those computed before fine-grid smoothing!) to the coarser grids during the up-traversal. Simultanously, we smooth on all grid levels as soon as the accumulation of residual parts (resulting from operator evaluation or restriction) is finished.

Multiplicative Multigrid. In contrast to additive multigrid methods, an iteration of a multiplicative multigrid method cannot be done within one top-down-depth-first sweep over the corresponding cell tree. We have to perform a run over the whole cell tree for each operation (smoothing, interpolation, restriction) but stop at the current grid level to prevent an overall cost of $O(N \log N)$ for a grid with N unknowns on the finest level. As a result, we have to swap out fine grid data to intermediate data structures whenever they are not needed.

The F-Cycle. If we now combine our multigrid methods with dynamical adaptivity, we end up with the so-called full-multigrid methods or F-cycles which start with an a priori defined preliminary and in general quite coarse grid, compute a first solution on this grid, apply some adaptivity criteria, refine or coarsen the grid according to these criteria, and, finally, compute the solution on the new grid. This results in an overall iterative process producing solutions on a sequence of incrementally enhanced grids.[2]

4 Data Structures

As mentioned above, our algorithm processes the grid in a top-down depth-first order and performs a cell-oriented evaluation of the discrete operators (difference stencils, restriction, interpolation). The cell-oriented view helps us to avoid the storage of unneccessary data (pointers to sons/fathers, specialised stencils at boundaries between different refinement depths) and, second, enhances the locality of data accesses if we additionally provide suitable data structures for vertex data[3].

4.1 Space-Filling Curves as an Ordering Mechanism

To achieve an optimal locality of data access and, thereby, a high cache-efficiency, we define a unique 'suitable' processing order of our grid cells and examine the resulting processing order of vertex data.

Space-filling curves [11] are used as an established tool for parallelisation and balanced load distribution [7,8,9,10] for PDE solvers on adaptive grids. The quasi-optimal locality of the class of self-similar recursively defined space-filling curves yields quasi-minimal communication costs [10]. Exactly this quasi-optimal locality is the property which is decisive for the optimisation of time locality of data usage in our algorithm, too. In general, self-similar recursively defined space-filling curves are given by a generating template connecting the *cells* of a first decomposition of the unit square (2D) or unit cube (3D) and a set of rules

[2] In the case of time-dependend problems, we have to adapt the grid to the temporal changes after each or after some time steps, in addition.

[3] Whenever we need cell-centered data such as the pressure in the Navier-Stokes equation, the allocation of the respective data structure is trivial: data are simply stored in a stream corresponding to the cell-stream resulting from our top-down-depth-first tree traversal.

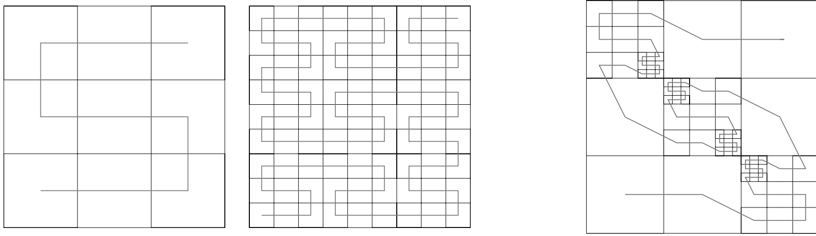

Fig. 2. Template, first two iterate and iterate for an adaptively refined space-partitioning grid of the the two-dimensional Peano curve

describing the recursive application of this template (possibly mirrored and/or rotated) on each of the subdomains in case of further refinement. Fig. 2 shows the generating template and the first iterate of the two-dimensional Peano curve as an example. The space-filling curves themselves are define as the limit of this recursive refinement process.

As we look for ordering mechanisms for the cells of a discrete grid, we are in fact not interested in the space-filling curves itself but in their iterates. Since the refinement rules of the curves are purely local, these iterates can be naturally generalised to adaptively refined space-partitioning grids. See Fig. 2 for an example. The high locality of the resulting oder of cells results from the fact that the curve visits all son (and grandson,...) cells of a father cell at once before they proceed to other cells not contained in the repective father cell. Such, all work to be done at one vertex of the grid (during the evaluation of operators) is finished within a short time period in the average case.

The resulting oder of grid cells can be easily generalised to an order of the cells of *all* refinement levels (needed hierarchical methods such as multigrid). Following the recursive definition of space-filling curves, we start with the coarsest cell (covering the whole computational domain), apply the curve's template in this cell, visit the son cells according to this template, their son cells (if existing) and so on. Thus, we end up with a particular – uniquely defined – top-down-depth-first odering as already presumed in Sect. 3.

4.2 Space-Filling Curves and Stacks

In the literature, we already find hints [12] on substantial improvements of the cache-preformance and, thus, the runtime of a PDE solver by the pure reordering of data according to their usage during the run along a space-filling curve. We go one step further and consequently use the properties of the Peano curve to construct data structures with optimal spatial locality perfectly supplementing the time locality described above. We will explain the underlying idea for two-dimensional regular grids with nodal data. Fig. 3 shows the Peano curve in two-dimensional regular grids. Moving along the curve, we see that the grid points at the left-hand side of the curves are processed in one direction during the first pass of the curve and exactly in the opposite direction during the second pass of the curve. The same holds for the points at the right-hand side of the curve.

This directly corresponds to a very simple type of data structures, the so-called stacks, which allow only two basic operations: `put` a datum on top of the stack and `pop` a datum from the top of the stack. In our examples, we can perform one iteration over all data with four data structures: one input stream containing all data in the order of their first usage, two stacks – corresponding to the left- and the right-hand side of the curve – on which we `put` data after the first pass of the curve and from which we `pop` them during the second pass, and one output stream collecting all vertex data after their last usage. The output stream can directly be used as an input stream for the next solver iteration if we process the grid along the same curve but in the opposite direction.

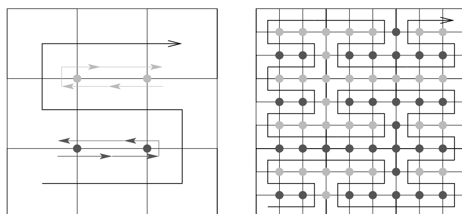

Fig. 3. Assignment of vertices and their data to two groups: left- and right-hand side of the Peano curve corresponding to two stacks used for the intermediate storage of data during solver iterations [4]

In the three-dimensional case, this concept works only if an analogue forward and backward processing of vertex data also holds on the two-dimensional faces of the cubic cells. We could not find any Hilbert curve fulfilling this requirement, whereas it is trivial to show for the Peano curve due to their dimension-recursive definition [13,14]. The generalisation to adaptively refined grids with hierarchical data is quite technical both in the two-dimensional and in the three-dimensional case. Therefore, we refer to other publications [15,13,14,16] for further details. We only state the results: we need eight intermediate stacks in the two-dimensional case and 27 stacks in the three-dimensional case, both independent of the current refinement depth of our grid.

5 Evaluation and Numerical Results

Before we present results on the hardware efficiency of our concept – measured by the cache-efficiency and the runtime – we would like to point out that our algorithms fulfill all numerical requirements, in particular offer a flexible and dynamical adaptivity and multigrid performance of the solvers. For detailed results see [15,13,14,6].

5.1 Cache-Efficiency and Storage Requirements

As described above, the storage of administrational data is almost obsolete due to the structueredness of the grid and the cell-oriented operator evaluation. There-

fore, we end up with a very low storage requirement of only about five Byte per degree of freedom in the case of the three-dimensional Poisson equation (with unknowns stored as `float` variables), for example [13].

If we look at the cache-efficiency of our programs, we observe an extremely high hit-rate in the L2-cache of above 99.9% in two and in three dimensions [15,13]. Similar hit-rates could be achieved as well for all algorithmic extensions such as dynamical adaptivity [6], higher order discretisations [14], parallelization [17,18,19], and the Navier-Stokes solver [3,2]. In addition, we could show that the actual number of L2-cache-misses is only about 10% larger than the theoretical minimum given by the need to load data to the cache at least once per solver iteration [15,13][4]. Besides those absolute values, we would like to point out the robustness of our concept with the help of three observations:

- Both the storage- and the cache-efficieny do not depend of the degree of adaptivity of our grid.
- Our approach can be successfully applied without any knowledge of the cache parameters (cache size, cache line length, ...) and, thus, can be easily ported to different platforms. In the literature, such methods are denoted as cache-oblivious [22,23].
- The cache hit-rates scale perfectly with the number of degrees of freedom. That is, there are no size effects deteriorating the efficiency above a certain problem size.

5.2 Runtime and Parallel Efficiency

The runtime of a program is surely the last and most important criterion of efficiency. To give some indication of the potential of our program in comparison with other cache-optimized PDE solvers, we compare the runtimes with DiME [24], a cache- and runtime-optimised multigrid solver for partial differential equations on regular grids. Concretely, we compare one DiME-V-cycle with one pre- and one postsmoothing step to one iteration of our additive multigrid method for the solution of the two-dimensional Poisson equation on a regular grid. Table 1 shows the resulting runtimes. Our program is about five times slower than DiME, which is quite good if we take into account that we can handle fully adaptive grids, the runtime per degree of freedom is independent on the adaptivity pattern of the grid [15,13], the storage requirements of our program are very low (less than seven Bytes per degree of freedom even in the three-dimensional case [13] whereas DiME needs more than 27 Bytes per degree of freedom in the two-dimensional case), and there are still numerous further optimisation potentials for our program (see Sect. 6).

As mentioned above, space-filling curves are a well-known tool for balanced parallelisation of algorithms on adaptive grids. We stated in addition, that we can easily generalize our data structures to processes working on a part of the grid

[4] We measured and simulated the cache-efficieny with the help of the tools *cachegrind* [20] (simulation), *perfmon* [21] and *hpcmon* (hardware performance counter).

Table 1. Comparison of the runtimes per degree of freedom and multigrid iteration for our code (left hand side) and DiME [24] (right hand side) on an AMD Athlon XP 2400+ (1.9 GHz) processor with 256 KB cache and 1 GB RAM using the *gcc3.4* compiler with options `-O3 -Xw` (from [1])

grid res.	# deg. of freedom	runtime per it and dof
243	$5.95 \cdot 10^4$	$1.70 \cdot 10^{-6}$ sec
729	$5.33 \cdot 10^5$	$1.71 \cdot 10^{-6}$ sec
2187	$4.79 \cdot 10^6$	$1.72 \cdot 10^{-6}$ sec

grid res.	# deg. of freedom	runtime per it and dof
257	$6.60 \cdot 10^4$	$5.78 \cdot 10^{-7}$
513	$2.63 \cdot 10^5$	$4.23 \cdot 10^{-7}$
1025	$1.05 \cdot 10^6$	$3.85 \cdot 10^{-7}$
2049	$4.20 \cdot 10^6$	$3.74 \cdot 10^{-7}$

Table 2. Parallel speedup achieved for the solution of the three-dimensional Poisson equation on a spherical domain on an adaptive grid with $23,118,848$ degrees of freedom. The computations were performed on a myrinet cluster consisting of eight dual Pentium III processors with 2 GByte RAM per node [17,25].

processes	1	2	4	8	16
speedup	1.00	1.95	3.73	6.85	12.93

only [17]. Table 2 shows the achieved parallel speedups for the three-dimensional Poisson equation on an adaptive grid on a spherical domain with $23,118,848$ degrees of freedom [17].

6 Conclusions and Outlook

We presentet a new approach for the hardware- and, in particular, memory-efficient implementation of state-of-the-art numerical methods, that is dynamically adaptive multigrid solvers. Without loosing mathematical functionality and/or efficiency, we maintain an extremely high cache-efficiency and a very low storage requirement per degree of freedom. In terms of the runtime, the last and crucial criterion, we are about a factor of five to ten slower than highly optimised solvers working on regular grids. Although this is already a good result taking into account that we work on adaptive grids with arbitrarliy local refinements, this is a clear hint that there is still a potential for further improvements for example based on methods similar to those presented in [26,27] or with the help of streaming SIMD extensions (SSE). Furthermore, the simplification of stack definition and administration is a currently active – and promising – field of our research as it will reduce stalls of floating point operations due to administrational integer operations.

Besides the detailed examination of the applicability of these optimisation possibilities, an important focus of our future work is on the implementation of more complicated equations and systems of equations, in particular the three-dimensional Navier-Stokes equations. Hereby, we can already start from an existing implementation of the two-dimensional Navier-Stokes equations [2,3]. Such 'real' applications will also show the effective potential of our method in comparison to other (adaptive or regular) solvers.

References

1. M. Mehl, T. Weinzierl, Ch. Zenger. A cache-oblivious self-adaptive full multigrid method. To appear in: special issue Copper Mountain Conference on Multigrid Methods 2005, *Numerical Linear Algebra with Applications*, Wiley Interscience.
2. T. Neckel. *Einfache 2D-Fluid-Struktur-Wechselwirkungen mit einer cache-optimalen Finite-Element-Methode.* Diploma thesis, Institut für Informatik, Technische Universität München, (2005).
3. T. Weinzierl. *Eine cache-optimale Implementierung eines Navier-Stokes Lösers unter besonderer Berücksichtigung physikalischer Erhaltungssätze.* Diploma thesis, Institut für Informatik, Technische Universität München, (2005).
4. M. Brenk, H.-J. Bungartz, M. Mehl, and T. Neckel. Fluid-Structure Interaction on Cartesian Grids: Flow Simulation and Coupling Interface. In Bungartz and Schfer (eds.), *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, LNCSE series, Springer, to appear.
5. D. Braess. *Finite Elements. Theory, Fast Solvers and Applications in Solid Mechanics*, Cambridge University Press, (2001).
6. N. Dieminger. *Kriterien für die Selbstadaption cache-effizienter Mehrgitteralgorithmen.* Diplomarbeit, Institut für Informatik, Technische Universität München, (2005).
7. M. Griebel and G. Zumbusch. Hash based adaptive parallel multilevel methods with space-filling curves. In: Rollnik and Wolf (eds.), *NIC Series* **9**, (2002), 479-492.
8. A. K. Patra, J. Long, and A. Laszloff. Efficient Parallel Adaptive Finite Element Methods Using Self-Scheduling Data and Computations. In: Banerjee, Prasanna, and Sinha (eds.), *High Performance Computing – HiPC'99, 6th International Conference, Calcutta, India, December 17-20, 1999, Proceedings*, HiPC, Lecture Notes in Compter Science **1745**, (1999), 359-363.
9. S. Roberts, S. Klyanasundaram, M. Cardew-Hall, and W. Clarke. A key based parallel adaptive refinement technique for finite element methods. In: Noye, Teubner, and Gill (eds.), *Proceedings Computational Techniques and Applications: CTAC '97*, World Scientific, Singapore, (1998), 577-584.
10. G. W. Zumbusch. *Adaptive Parallel Multilevel Methods for Partial Differential Equations*. Habilitationsschrift, Universität Bonn, (2001).
11. H. Sagan. *Space-Filling Curves*. Springer, New York, (1994).
12. M. J. Aftosmis, M. J. Berger, and G. Adomavivius. *A Parallel Multilevel Method for Adaptively Refined Cartesian Grids with Embedded Boundaries*, American Institute of Aeronautics and Astronautics-2000-808, Aerospace Science Meeting and Exhibit, 38th, Reno, Nevada, Jan 10-13, (2000).
13. M. Pögl. *Entwicklung eines cache-optimalen 3D Finite-Element-Verfahrens für große Probleme*. Doctoral thesis, Institut für Informatik, TU München, (2004).
14. A. Krahnke. *Adaptive Verfahren höherer Ordnung auf cache-optimalen Datenstrukturen für dreidimensionale Probleme*. Doctoral thesis, Institut für Informatik, TU München, (2005).
15. F. Günther. *Eine cache-optimale Implementierung der Finite-Elemente-Methode*. Doctoral thesis, Institut für Informatik, TU München, (2004).
16. F. Günther, M. Mehl, M. Pögl, Ch. Zenger. A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves. *SIAM Journal on Scientific Computing*, in review.

17. M. Langlotz. *Parallelisierung eines Cache-optimalen 3D Finite-Element-Verfahrens.* Diplomarbeit, Institut für Informatik, TU München, (2004).
18. W. Herder. *Lastverteilung und parallelisierte Erzeugung von Eingabedaten für ein paralleles Cache-optimales Finite-Element-Verfahren.* Diplomarbeit, Institut für Informatik, TU München, (2005).
19. F. Günther, A. Krahnke, M. Langlotz, M. Mehl, M. Pögl, and Ch. Zenger. On the Parallelization of a Cache-Optimal Iterative Solver for PDEs Based on Hierarchical Data Structures and Space-Filling Curves. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users Group Meeting Budapest, Hungary, September 19 - 22, 2004.* Proceedings, Lecture Notes in Computer Science, Vol. 3241/2004, Springer, Heidelberg, (2004).
20. J. Seward, N. Nethercote, and J. Fitzhardinge. *cachegrind: a cache-miss profiler*; `http://valgrind.kde.org/docs.html`
21. HP invent. *perfmon: create powerful performance analysis tools wich use the IA-54 Performance Monitoring Unit (PMU)*; `http://www.hpl.hp.com/research/linux/perfmon/index.php4`.
22. M. Frigo, C. E. Leierson, H. Prokop, and S. Ramchandran. Cache-oblivious algorithms. In: *Proceedings of the 40th Annual Sympoisium on Foundations of Computer Science*, New York, (1999), 285-297.
23. E. D. Demaine. Cache-Oblivious Algorithms and Data Structures. In: *Lecture Notes from the EEF Summer School on Massive Data Sets, University of Aarhus, Denmark, June 27-July 1*, Lecture Notes in Computer Science, Springer, (2002).
24. M. Kowarschik, C. Weiß. DiMEPACK – A Cache-Optimal Multigrid Library. In: Arabnia (ed.), *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Application (PDPTA 2001), Las Vegas, USA* **I**, (2001).
25. M. Langlotz, M. Mehl, T. Weinzierl, and C. Zenger. SkvG: Cache-Optimal Parallel Solution of PDEs on High Performance Computers Using Space-Trees and Space-Filling Curves. In: A. Bode und F. Durst (eds.), High Performance Computing in Science and Engineering, Garching 2004, Springer-Verlag, Berlin Heidelberg New York, (2005), 71-82.
26. C. C. Douglas, J. Hu, M. Kowarschik, U. Rüde, C. Weiß. Cache Optimization for Structured and Unstructured Grid Multigrid. *Electronic Transactions on Numerical Analysis* **10**, (2000), 21-40.
27. M. Kowarschik, C. Weiß. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In: Meyer, Sanders, and Sibeyn (eds.), *Algorithms for Memory Hierarchies – Advanced Lectures*; Lecture Notes in Computer Science **2625**, Springer, (2003), 213-232.