Cache-oblivious parallel multigrid solvers on adaptively refined grids

Miriam Mehl
mehl@in.tum.de
Institut für Informatik, TU München
Boltzmannstraße 3, 85748 Garching

Christoph Zenger
zenger@in.tum.de
Institut für Informatik, TU München
Boltzmannstraße 3, 85748 Garching

Abstract

In many implementations of modern solvers for partial differential equations, the use of multigrid methods in particular in combination with dynamically adaptive grids causes a non-negligible loss of efficiency in data access and storage usage due to an increasing complexity of data structures. We propose a concept based on space-partitioning grids, space-filling cuves, and stacks with the task to establish an efficiently parallelizable, storage saving, and, in particular, cache-oblivious multigrid solver. The side condition for this work is that we do not restrict the possibilities for arbitrarily local grid refinements.

1 Space-Partitioning Grids and Hierarchical Multilevel Data

Most modern numerical methods like multigrid techniques, dynamically adaptive solvers, and higher order methods based on extrapolation call for the usage of adaptive hierarchical multilevel data, and, in consequence, a flexible and efficient method to represent such data. A structured and, therefore, storage saving possiblity are recursivly space-partitioning grids similiar to those described in [7]. Such grids allow a recursive refinement of each single grid cell and, therefore, allow arbitrary (isotropic) refinements and are inherently hierarchical in terms of grid cells. In addition, there are established fast algorithms for geometry modelling with the help of recursive space-partitioning structures [3, 7, 6, 13].

To establish an algorithm for a solver of a partial differential equation or a system of partial differential equations, we have to define a visiting mechanism for the grid elements. The natural thing for space-partitioning grids is a top-down-depth first traversal of the grid cells (not the vertices!). This implicates a cell-wise evaluation of all operators needed in the solver, that is only the usage of data owning to the current cell is allowed. In the case of vertex data, this results in an accumulative operator evaluation. For finite element methods,

this is a well known approach [4]. In each cell we calculate the operator parts for all associated vertices ny performing the finite elelent integration over the respective cell only instead of the whole support of the basis functions. The big advantages of the cell-wise handling of the grid and the operator evaluations are, first, a strictly local data usage and, second, an automatic generation of operators at the boundaries between different refinement depths without any storage of specialized stencils [2, 9, 14]. The structuredness of space-tree grids makes the storage of pointers to father and/or sons obsolete. In fact, the tree of grid cells can be stored in a linearized way.

The top-down depth first traversal of grid cells allows a very natural implementation of additive multigrid methods [9, 14, 10, 5], but also the implementation of a multiplicative multigrid method can be done very efficiently by simple modifications of the control for the in- and output stream of solver iterations.

2 The Role of Space-Filling Curves

In the previous section, we shortly described the organization of hierarchical multilevel data in adaptively refined space-tree grids. An important component missing for the completion of a solver algorithm is a uniquely defined processing order for the grid cells. As we have to handle adaptively refined grids, simple concepts like index-oriented lexicographic ordering are not applicable. The main criteria for a good ordering mechanism are a high flexibility with respect to the adaptivity of the grid, compatibility with parallelization strategies, low storage costs, and a high time locality in data usage (which optimizes the cache-efficiency).

Recursively defined space-filling curves [16] or, actually, their discrete iterates fulfill all these criteria and are perfectly in line with the concept of space-partitioning grids as they are defined by a generating template defined on the unit square/cube, which is – from the viewpoint of the grid – the root cell, and a locally defined refinement rule corresponding to local grid refinements. The specific advantages of space-filling curves in our context will be discussed in detail in Section 3 by means of numerical results.

In our algorithm, we use a particular space-filling curve, the Peano-curve which is defined for arbitrary dimensions in a dimension-recursive way (see Figure 1). The Peano curve has the important property – which could not be shown for any Hilbert-curve for example – that projections of the curve to the boundaries of a subdomain are Peano-curves (of lower dimension) again and, second, that data on such boundaries are processed in one direction during the run of the curve through the cells on one side of the boundary and in the opposite direction during the run of the curve through the cells on the other side of the boundary. This property corresponds in a natural way to the idea of stacks – data structures which allow only two operations: put a datum on top of the stack and get a datum from the top of the stack. [9, 14, 11] showed that the whole algorithm works with a refinement independent and small number of stacks for any dimension. Stacks are inherently optimal in terms of spatial locality of data access.

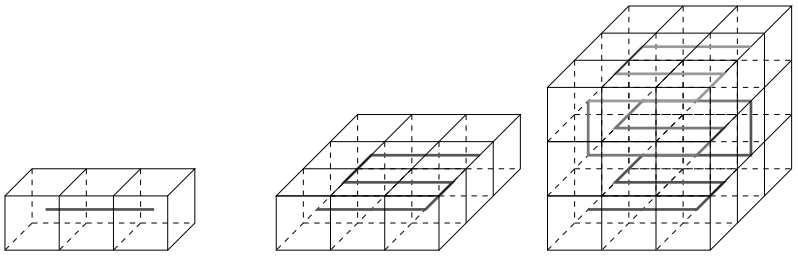

Figure 1: Dimension-recursive construction of the template of the Peano curve

3 Numerical Results

In this section, we will give some detailed results of our code considering several criteria for the efficiency and flexibility of the concept.

All computations were performed for the test equation

$$\Delta u(x) = -3\pi^2 \prod_{i=0}^{2} \sin(\pi x_i) \text{ for all } x \in \Omega \quad (1)$$

$$u = 0 \text{ at } \partial\Omega \quad (2)$$

on different domains Ω and on regular as well as adaptive grids.

3.1 Storage Requirements

The strict structuredness of the space-tree grids together with the cell-wise operator evaluation leads to very low storage costs per degree of freedom (Table 1): The storage of pointers to neighbours of a cell as well as extra-operators at boundaries between different refinement depths is obsolete.

domain	resolution	deg. of freedom	storage requ.	st. req. per dof
cube	243	14,702,584	76 MB	5.2 Byte
	729	400,530,936	2,000 MB	5.0 Byte
sphere	243(a)	855.816	6 MB	7.0 Byte
	729(a)	23.118.848	126 MB	5.5 Byte

Table 1: Storage requirements for the solution of the Poisson equation on a cubic and a spherical domain (taken from [14]). Test cases with adaptive grids are marked with '(a)'.

3.2 Parallelization

Besides low storage costs, an efficient parallelization of a simulation code is an important prerequisite for the handling of big and complex scenarios. Space-filling curves are a well-known tool for an easy and efficient balanced parallelization of algorithms working

on space-partiotioning grids [18]. The communication costs are quasi-minimal due to the locality properties of the curves. For speedups achieved with our code see Table 2.

# processes	1	2	4	8	16
speedup	1	1.95	3.77	7.04	13.65

Table 2: Parallel speedup in dependence on the number of processes on a myrinet cluster of 8 dual Pentium III PCs for an adaptively refined spherical domain with $2.59 \cdot 10^7$ degrees of freedom (see [12]).

3.3 Adaptivity

To keep the computational costs within a realizable limit also for complex problems and, in particular, for problems with singularities, we need a high flexibility of our grids. Space-partitioning grids – in contrast to for example block refined grids – allow arbitrarily local refinements, and, therefore, are suitable for singular problems (see Table 3). As a test case, we solved (1) and (2) on a unit cube with a resected edge ($\Omega =]0;1[^3 \setminus]1/3;1/3[^3$), the three-dimensional analogon to the well-known two-dimensional L-shape domain.

	error tolerance	ref. depth	number of degrees of freedom
regular grid	$5.9525 \cdot 10^{-3}$	3	$17,339$
	$1.1734 \cdot 10^{-3}$	4	$509,656$
adaptive grid	$5.9525 \cdot 10^{-3}$	3	$17,339$
	$1.1734 \cdot 10^{-3}$	4	$61,267$

Table 3: Comparison of the number of degrees of freedom required to achieve a given accuracy by a regular and an adaptive refinement of a regular start grid for the Poisson equation on the domain $\Omega =]0;1[^3 \setminus]1/3;1/3[^3$ (see [5]).

3.4 Cache-Efficiency

A big problem, many simulation programs suffer from, in particular as soon as they have to handle adaptive multilevel data, is a very inefficient utilization of the cache-hierarchy of modern computers. Solvers on such data – which are multigrid solvers in the typical case – have to establish connections between data in different coordinate directions and between several refinement levels. This causes in general frequent and big 'jumps' within the pysical memory space, and, thus, a high probability for cache-misses. Our algorithm avoids this inefficiency as it ensures a very high time and space locality of data by the combination of the cell-wise operator evaluation (no access to neighbours) with the usage of space-filling curves (ensuring time locality due to the locality properties of the curves [1, 9, 18]), and the concept of stacks (with inherently maximal spatial locality in data access)[1]. Table 4

[1] Algorithms and data structures which are cache-optimal by concept and not by fitting parameters (block sizes etc.) to the specific architecture are called cache-oblivious [8, 15]

shows that our code causes only about 10% more cache-misses for the level 2 cache than the theoretical minimum (for more details see [9, 14, 10]).

c_l	$nit \cdot cms_{min}$	cms_{meas}	rate
64 Byte	$248,483$	$271,416$	1.09
128 Byte	$124,241$	$136,599$	1.10
256 Byte	$62,121$	$69,036$	1.11

Table 4: Cache-optimal behaviour for different cache-line lengths c_l measured by the cache-simulator *cachegrind* [17] for the Poisson equation on the unit cube with $530,000$ degrees of freedom (see [14])

3.5 Runtime

In terms of runtime, our program is not optimzed yet. In particular a more efficient implementation of the stack administration can be assumed to give some substantial gains. In spite of this, there is still one main remarkable result: The computational time per degree of freedom is independent from the degree of adaptivity/irregularity and the size of the grid (see Table 5).

domain	resolution	runtime per dof and it
cube	243	$5.77 \cdot 10^{-6}$ sec
	729	$5.66 \cdot 10^{-6}$ sec
sphere	243(a)	$6.96 \cdot 10^{-6}$ sec
	729(a)	$6.05 \cdot 10^{-6}$ sec

Table 5: Runtimes per degree of freedom and per solver iteration for the Poisson equation solved on an Intel Dual Xeon with 2.4 GHz, 4 GByte RAM, using the Intel Compiler 8 with options `-O3 -xW` (see [14]). Test examples which used adaptiv grids are marked by '(a)'.

3.6 Large Problems

For large three-dimensional problems, we are often faced with the difficulty that the available storage is not sufficient. As our program works with a very special kind of data structures and, in particular, the in- and output stack of each iteration are processed in a strict linear order, we can store these data on the hard disk and load them to the main memory in a demand-driven way using buffers [14]. Independent of the size of the buffers and of the architecture, this didn't lead to a worsening of runtimes. In contrary, for many configurations, the runtime became even shorter (up to 10%) compared to the version without usage of the hard disk. This phenomenon could not be completely explained yet.

resolution	degrees of freedom	buffer size	RAM/hard disk	runtime per dof and it
729	$400,530,936$	32 MB	157 MB / 2 GB	$4.8 \cdot 10^{-6}$ sec
2187	$10,846,541,792$	64 MB	470 MB / 55 GB	$5.3 \cdot 10^{-6}$ sec

Table 6: Runtimes per degree of freedom and per solver iteration (additive multigrid) for the Poisson equation on a unit cube solved on a Intel Dual Xeon with 2.4 GHz, 4 GByte RAM with outsourcing of data to the hard disk (see [14])

4 Conclusion

We could show that the special combination of adaptive space-partitioning grids with multilevel data associated to the vertices of the cell, space-filling curves, and stacks as data structures leads to a highly efficient and flexible program ensuring a flexible local adaptivity suitable even for singular problems, a low storage requirement independent of the degree of adaptivity/irregularity, a good parallel efficiency, an extremely high cache-efficiency, runtimes which are independent of the adaptivity of the grid, and, last but not least the possibility to handle very large problems by an outsourcing of data to the hard disk without loosing efficiency. Thus, we conclude that our concept is very valuable as soon as the application calls for a sophisticated dynamical adaptivity combined with multigrid methods.

References

[1] *Aftosmis, M.J., Berger, M.J., and Adomavivius, G.:* A Parallel Multilevel Method for adaptively Refined Cartesian Grids with Embedded Boundaries. AIAA Paper, 2000.

[2] *Blanke, C.:* Kontinuitätserhaltende Finite-Element-Diskretisierung der Navier-Stokes-Gleichungen. Diploma thesis, Fakultät für Informatik, Technische Universität München, 2004.

[3] *Brenk, M., Bungartz, H.-J., Mehl, M., Mundani, R.-P., Düster, A., and Scholz, D.:* Efficient Interface Treatment for Fluid-Structure Interaction on Cartesian Grids. In Proc. of the ECCOMAS Thematic Conf. on Comp. Methods for Coupled Problems in Science and Engineering. International Center for Numerical Methods in Engineering (CIMNE), 2005.

[4] *Braess:* Finite Elements. Theory, Fast Solvers and Applications in Solid Mechanics. Cambridge University Press, 2001.

[5] *Dieminger, N.:* Kriterien für die Selbstadaption cache-effizienter Mehrgitteralgorithmen. Diploma thesis, Fakultät für Informatik, Technische Universität München, 2005.

[6] *Bungartz, H.-J., Frank, A., Meier, F., Neunhoeffer, T., and Schulte, S.:* Efficient treatment of complicated geometries and moving interfaces for CFD problems. In H.-J.

Bungartz, C. Zenger und F. Durst (Hrsg.), High Performance Scientific and Engineering Computing, Lecture Notes in Computational Science and Engineering, pp. 113-123. Springer, Berlin, Heidelberg, August 1999.

[7] *Frank, A.:* Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung. Doctoral thesis, TU München, 2000.

[8] *Frigo, M., Leierson, C.E., Prokop, H., Ramchandran, S.:* Cach-oblivious algorithms. In: Proceedings of the 40th Annual Sympoisium on Foundations of Computer Science, pages 285-297, New York, October 1999.

[9] *Günther, F.:* Eine cache-optimale Implementierung der Finite-Elemente-Methode. Doctoral thesis, TU München, Mai 2004.

[10] *Kranke, A.:* Adaptive Verfahren höherer Ordnung auf cache-optimalen Datenstrukturen für dreidimensionale Probleme. Doctoral thesis, TU München, 2005.

[11] *Hartmann, J.:* Entwicklung eines cache-optimalen Finite-Element-Verfahrens zur Lösung d-dimensionaler Probleme. Diploma thesis, Fakultät für Informatik, Technische Universität München, 2005.

[12] *Langlotz, M.:* Parallelisierung eines Cache-optimalen 3D Finite-Element-Verfahrens. Diploma thesis, Fakultät für Informatik, Technische Universität München, 2004.

[13] *Mundani, R.-P., Bungartz, H.-J., Rank, E., Romberg, R., and Niggl, A.:* Efficient Algorithms for Octree-Based Geometric Modelling. In Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Comp.. Civil-Comp Press, 2003.

[14] *Pögl, M.:* Entwicklung eines cache-optimalen 3D Finite-Element-Verfahrens für große Probleme, Fortschritt-Berichte VDI, 10 Informatik Kommunikation. Doctoral thesis. VDI Verlag, Düsseldorf, 2004.

[15] *Prokop, H.:* Cache-Oblivious Algorithms. Master Thesis, Massachusetts Institute of Technology, 1999.

[16] *Sagan, H.:* Space-Filling Curves. Springer-Verlag, New York, 1994.

[17] *Seward, J., Nethercote, N., Fitzhardinge, J.:* cachegrind: a cache-miss profiler. `http://valgrind.kde.org/docs.html`

[18] *Zumbusch, G.W.:* On the quality of space-filling curve induced partitions. Z. Angew. Math. Mech., 81:25-28, 2001. Suppl. 1, also as report SFB 256, University Bonn, no. 674, 2000.