APCOM'07 in conjunction with EPMESC XI, *December 3-6, 2007, Kyoto, JAPAN*

Benefits of Structured Cartesian Grids for the Simulation of Fluid-Structure Interactions

Miriam Mehl[1]*, Markus Brenk[1], Ioan L. Muntean[1], Tobias Neckel[1], Tobias Weinzierl[1]

[1] *Department of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany*

e-mail: mehl@in.tum.de, brenk@in.tum.de, muntean@in.tum.de, neckel@in.tum.de, weinzier@in.tum.de

Abstract Due to the different physical effects involved and the complicated and changing geometries, the simulation of fluid-structure interactions still is a highly challenging task. Thus, simplicity and efficiency are essential for the corresponding software codes. In this paper, we consider the potential of structured Cartesian grids with respect to numerical and hardware efficiency as well as the development of user friendly tools for partitioned simulations of fluid-structure interactions. In contrast to the so-called monolithic approach, the partitioned approach does not establish a new complete solver for the whole scenario but instead embeds two existing solver – for fluid flow and structural mechanics – into a bigger environment handling the combination of both fluid and structure. Our fist task concerns the fluid solver. Here, we exploit the structuredness and spatially recursive properties of structured adaptive Cartesian grids to bring together both numerical efficiency (in the form of multigrid solvers on adaptively refined grids) and hardware efficiency (in particular memory efficiency). The second task is to provide tools for coupling the two solvers involved in an easy, modular, and flexible way. For this purpose, we developed a coupling client FSI*ce with clearly defined interfaces and a central description of the coupling surface between fluid and structure hiding the two solver grids from each other and, thus, making the solver completely independent from each other. In addition, FSI*ce controls the whole coupled simulation which leads to a central implementation of the coupling strategy (instead of an implementation in one of the solvers). Finally, libraries for general tasks such as data interpolation and projection are established. Here, our Cartesian grids come into play again as they are used as a very efficient tool for the administration and modification of the geometric information such as neighborhood relations between nodes of the spatial solver grids and the central surface description (lower dimensional triangulation).

Key words: partitioned fluid-structure simulation, coupling tool, adaptive Cartesian grids, octrees, flow solver

INTRODUCTION

In this introduction, we will address the basic questions concerning structured Cartesian grids in combination with fluid-structure interactions:

- What are adaptive structured Cartesian grids?
- What do we need for partitioned fluid-structure simulations?
- How can adaptive structured Cartesian grids be helpful in this context?

What are adaptive structured Cartesian grids? The adaptive Cartesian grids we use consist of square or cubic grid cells in the two-dimensional or three-dimensional case, respectively. A generalization would be to allow for rectangular cells as well. The grids are generated via subsequent recursive subdivision of the grid cells. That is, we start a recursive refinement procedure with a very big cell containing the whole computational domain. In each refinement step and in each cell, we decide whether or not to refine the cell further according to a suitable refinement criterion. In case of further refinement, the cell is subdivided into a fixed number of equal son cells. Due to this spacepartitioning property of the refinement process, we call such grids spacepartitioning grids in the following. As we can use arbitrary local refinement criteria, these grids allow for very flexible and local grid adaptivity. Fig. 1 shows examples for spacepartitioning grids with a subdivision of cells into three parts per coordinate direction and refinement step.

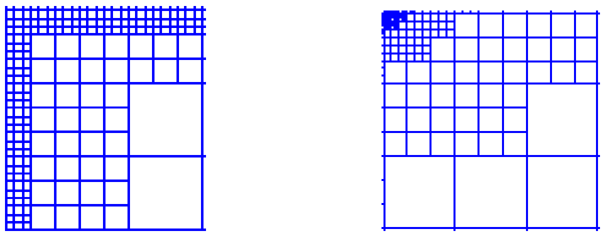


Fig. 1. Examples for two-dimensional spacepartitioning grids with partitioning into thee parts per refinement step and coordinate direction.

What do we need for partitioned fluid-structure simulations? To efficiently simulate fluid-structure interaction scenarios with a partitioned approach, we need solvers that use grids capable of handling complex and strongly changing geometries. This holds in particular for the fluid solver as, in general, the changes of the fluid domain are much larger and topology changes are much more frequent, here. To illustrate this, Fig. 2 show as scenario with particles moving in a fluid where the structure (particle) geometry doesn't change at all whereas the fluid domain is undergoing very substantial modifications.

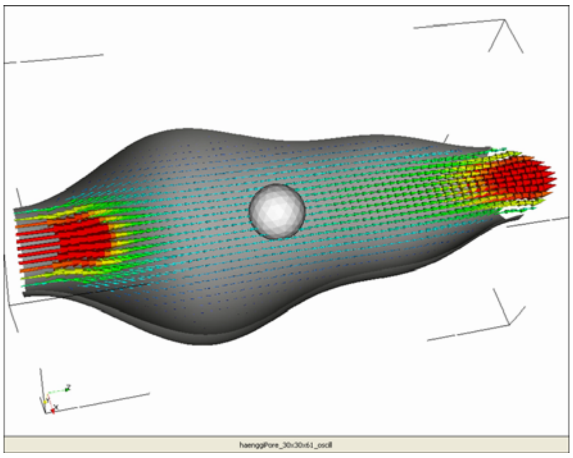


Fig. 2. Sphere (structure) transported in a flow channel with oscillating diameter. The movement of the particle implies strong changes on the fluid domain whereas the structure domain remains unchanged (if the midpoint of the sphere is considered as the origin of the coordinate system).

For the coupling of the two solvers involved in a partitioned simulation of fluid-structure interactions, a tool managing the communication between the solvers is required. In the ideal case, the tool is constructed in a way that enables the user to independently exchange solvers and the coupling strategy, that is to replace one of these three components without having to change the other two remaining components. In addition, efficient functions supporting the mapping of data from one solver grid to the other one should be provided.

How can Cartesian grids be helpful in this context? As mentioned above, in particular the flow solver has to handle large changes of the computational domain up to topology changes. In such cases ALE (Arbitrary Lagrangian Eulerian [1]) approaches reach their limits. Grid deformations induced by the structure movements become too large to maintain a numerically satisfying shape of the grid cells. In this case as well as in the case of topology changes of course, the generation of a new grid becomes necessary, which is quite costly for many grid types such as unstructured grids. If we use spacepartitioning grids, however, the generation of the grid becomes very efficient and cheap as we can use octree-like algorithms exploiting the inherent spatial recursivity and, thus, inheritance properties of the grid structure. Thus, using an Eulerian (fixed grid) approach in combination with spacepartitioning grids has a big efficiency

potential if we have to handle large deformations or movements of the structure within the fluid domain [2,3]. In addition, spacepartitioning grids allow for a hardware efficient implementation of state of the art numerical solver that is in particular multigrid solvers on dynamically refined adaptive grids and a simple realization of a balanced parallelization with the help of space-filling curves [4].
If we look at the coupling aspect of partitioned fluid-structure simulations, the spatial recursivity comes into play again as it allows – again based on inheritance – for a very efficient determination of neighborhood relations between grid cells, nodes, and elements of the spatial solver grid and the central representation of the lower-dimensional representation of the coupling surface in the coupling client. These neighborhood relations are essential for the mapping of data between the grids (i.e. interpolation or projection) [2].
In the following section, we will present the general algorithm for the hardware-efficient implementation of a numerically efficient PDE solver on spacepartitioning grids at the example of the Navier-Stokes solver. The regeneration of the spacepartitioning grids after geometry changes as well as the usage of spacepartitioning grids for the data mapping between different grids will be presented in the third section. Here, we will also shortly describe the coupling environment FSI*ce. Finally, we present some fist results.

EFFICIENT USAGE OF CARTESIAN GRIDS IN PDE SOLVERS

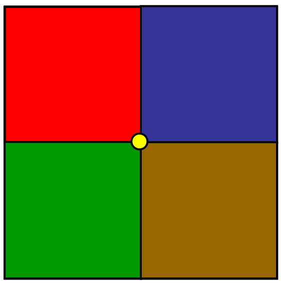
To face the requirements posed by highly complex applications, PDE solvers should nowadays combine numerically efficient methods such as multigrid methods on adaptively refined grids with hardware efficiency, that is in particular efficient data structures and data access mechanisms. The common element of all iterative solvers or time stepping algorithms is a node-by-node or cell-by-cell processing of the computational grid. Depending on the solver or the current action such as evaluation of difference stencils, interpolation, restriction etc., different computations are performed in each cell or at each node of the grid. In this section, we will describe the underlying data storage and data processing at the example of a Navier-Stokes solver. We solve the Navier-Stokes equations for incompressible viscous fluids:

$$\vec{u}_t = \frac{1}{re}\Delta\vec{u} - (\vec{u}\cdot\nabla)\vec{u} - \nabla p \quad \text{in } \Omega,$$
$$\nabla^T\vec{u} = 0 \;\text{ in } \Omega.$$

As a basis of the spatial discretization, we use a semi-staggered grid, that is assign the velocity degrees of freedom to the nodes of the grid and the pressure to the midpoints of the grid cells [3]. For the spatial discretization, we use a finite element discretization [3,5]. However, the algorithms described in the following can be applied to arbitrary (local) difference operators. For time stepping, we apply an explicit Euler in combination with Chorins projection method [6,7]. Also this discretization is only an example and can be replaced by any other time stepping scheme.

First step towards efficiency: cell-wise operator evaluation. The first step towards memory efficiency is to consequently choose cellwise operator evaluation, that is to decompose all operators (difference operators, restrictions, interpolations) in cell-parts such that each cell-part only requires data owned to the cell (data assigned to nodes or the midpoint of the cell). The complete operator is accumulated from all cell-parts involved. Fig. 3 shows how this works for the common five-point stencil discretization of the two-dimensional Laplace operator. The advantages of this cell-wise operator evaluation are, first, that we do not have to store any neighborhood relations of cells in the adaptively refined grid as we only need data belonging to the respective current cell and, second, that the evaluation of operators at boundaries between different adaptive refinement levels becomes very straight-forward. The cell-parts look the same in each cell independently from the refinement depth of the neighboring cells. The only thing we have to do is to correctly transport cell-parts between the grid levels via suitable interpolation and restriction operators at such jumps in the refinement depth. Thus, the cell-wise operator evaluation saves a lot of memory space as we do neither store neighborhood relations of grid elements nor special operators at refinement depth jumps in the adaptive grid [3].

To be able to apply this principle also for the evaluation of the Laplace operator of the cell-centered pressure values, we simply decompose the Laplace operator into two parts: In a first step, we compute the pressure gradients at the cell vortices and in a second step, we use these gradients to compute the Laplacian at the midpoints of the cells [8].



$$\begin{bmatrix} & \frac{1}{2} & \\ \frac{1}{2} & -1 & \\ & & \end{bmatrix} + \begin{bmatrix} & \frac{1}{2} & \\ & -1 & \frac{1}{2} \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ \frac{1}{2} & -1 & \\ & \frac{1}{2} & \end{bmatrix} + \begin{bmatrix} & & \\ & -1 & \frac{1}{2} \\ & \frac{1}{2} & \end{bmatrix} = \begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$$

Fig. 3. Cell-wise evaluation of the five-point stencil discretizing the two-dimensional Laplace operator.

Second step towards efficiency: processing order and data structures. What we need now in a second step is of course a sophisticated data access and data storage mechanism, as the solution to store data owned to the grid cells can obviously not be to store all data (i.e. velocities) assigned to cell nodes in each cell neighboring the respective node separately that is four times (in 2D) or eight times (in 3D), respectively. To establish such an access and data storage mechanism, we start with defining a processing order of the cells of our grid. Here, we use iteratives of a particular self-similar recursively defined space-filling curve [9], namely the Peano-curve. The recursive definition of such curves perfectly fits the spatially recursive construction of our adaptive grids. Fig. 4 shows two-dimensional spacepartitioning grids with a partitioning into three parts per coordinate direction together with the associated iteratives of the Peano curve. The reasons why we use the Peano curve and, therewith, accept the partitioning into three cells per coordinate direction and refinement step instead of the common partitioning into two, are certain properties of the Peano curve necessary for the construction of data structures as described in the following.

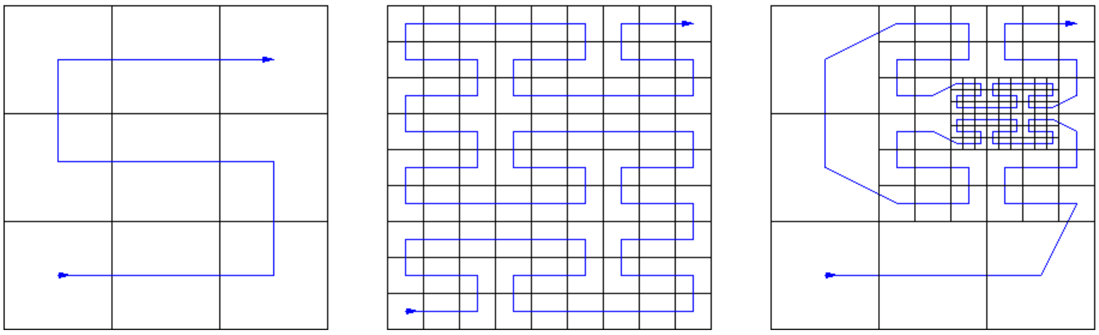


Fig. 4. Two-dimensional space-partitioning grids with a portioning into three per coordinate direction and refinement step together with the associated iterate of the Peano curve defining the processing order of the grid cell in our cell-oriented algorithms.

If we have a detailed look at the order of data usage resulting from the usage of the Peano curve at the hand of a regular two-dimensional grid as displayed in Fig. 5, we can define two groups of grid nodes with a very nice property: those lying ‘left’ from the Peano curve (red in Fig. 5) and those lying ‘right’ from the curve (green in Fig. 5). The nodes of each of these groups are used in a certain direction during the first pass of the curve and exactly in the opposite direction during the second pass of the curve. According to this, we can use highly efficient data structures to store them during one run over the grid cell (for operator evaluation, e.g.): stacks are data structures only allowing for two operations: put a datum on top of the stack or get a datum from the top of the stack. Such, no jumps in the physical memory space are possible (optimal spatial locality of data usage). In addition, the locality properties of the Peano curve [10] guaranty a quasi-optimal time locality of data usage that is a short time period between the first and the last usage of

a datum during the run over the grid cells. Both, high spatial and time locality, lead to a very efficient cache usage of the algorithm and, thus, a minimization of data access times. This principle of constructing data structures can be generalized to adaptively refined multilevel data (with a slightly high number of stacks in order to prevent the mutual hiding of grid points of different refinement levels within the stacks, [4,11]) and also to the three-dimensional case (here, we need the so-called projection and palindrome properties of the Peano curve, [4,12,13,14]).
In addition to data assigned to grid nodes, we also need data assigned to midpoints of the grid cells (pressure) in the context of the Navier-Stokes equation. However, this doesn't pose any problems at all since the handling of midpoint data during a run over the grid along the Peano curve is trivial: Each of these data is used only once such that we simply treat them as a data stream with an ordering defined by the Peano curve [8].

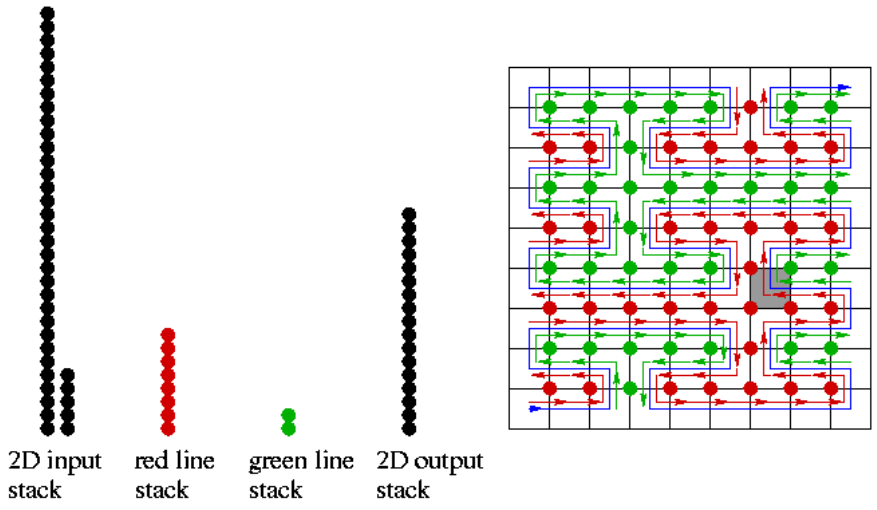


Fig. 5. Regular two-dimensional grid with the respective iterate of the Peano curve and the data structured required during one sweep over the grid along the curve: input stack holding data not yet needed, two line stacks (red and green) storing data between their first and last usage during the grid processing, output stack holding the data already used for the last time in this sweep over the grid. The grey cell marks the current position of the algorithms corresponding to the displayed fillment of the stacks.

Third step towards efficiency: balanced and efficient parallelization. Our third step towards a state of the art Navier-Stokes solver is the parallelization of the code. It is well known that space-filling curves such as the Peano curve that we already used for data access and the construction of data structures are a very efficient and simple tool to achieve a balanced and efficient domain decomposition for adaptively refined structured grids [10,15]. To define the domain decomposition, we simply line up all grid cells along the Peano curve and cut the resulting cell queue into equal pieces. Here, the locality properties of the Peano curve are helpful again as they lead to a domain decomposition with quasi-minimal surfaces of the partitions and, thus, quasi-minimal communication costs [10]. As the Peano curve doesn't define an order for the cells of one grid level only but – due to its inherent hierarchical construction – for the grid cells on all levels, even a multigrid solver on the adaptively refined grid can be parallelized in a very natural and efficient way [16,17].

GEOMETRY CHANGES AND DATA MAPPING WITHIN FSI*ce

Client Server Approach. As mentioned, we use a client-server approach fort the coupling of the fluid and the structure solver in a partitioned fluid-structure simulation. The so-called coupling client holds the complete information about the coupling surface (i.e. a central geometric description of the surface and all data stored thereon such as displacements, forces, velocities, tensions) and controls the simulation (i.e. the time stepping and possibly the iterations within single time steps) [18,19]. Fig. 6 shows the schematic concept together with the placement of the geometry handling and data mapping tools.

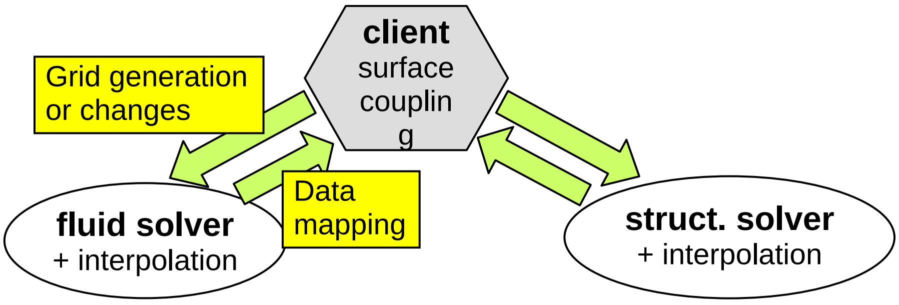


Fig. 6. Schematic view of the environment for partitioned fluid-structure simulations with the coupling client as central unit, two solvers acting as servers and grid generation/change as well as data mapping tools between the fluid solver and the coupling client.

Geometry generation and changes. Our spacepartitioning grids for the fluid solver in the context of fluid-structure interactions are generated from the central coupling surface description of the coupling client FSI*ce. We use an octree-like algorithm strongly relying on inheritance of informations stored in the cell tree such as 'is in fluid', 'is in structure', or 'contains a part of the surface'. It is well known that this results in a very fast algorithm (see [3,20] and the results section. Fig. 7 shows a very simple two-dimensional example for a surface description together with a corresponding spacepartitioning grid and the associated cell tree. In case of changes of the fluid geometry, the trivial possibility would be to generate a new grid according to the new surface description. Even this alternative would be very fast in particular compared to Lagrangian approaches where we have to solve equations on the whole domain to compute the new positions of the grid nodes [1]. However, we can handle changes even more efficiently if we restrict to areas (low-level or coarse grid cells of the cell tree) where changes really happen and only change the corresponding subtree(s) of the whole cell tree.

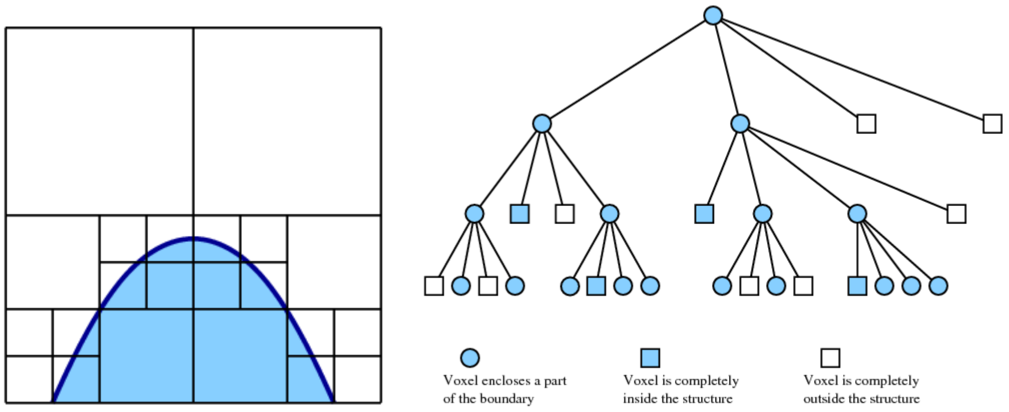

Voxel encloses a part of the boundary
Voxel is completely inside the structure
Voxel is completely outside the structure

Fig. 7. Surface model (blue line) together with the derived spacepartitioning grid and the corresponding cell tree (quadtree in this case, right, taken from [18]).

Data mapping. For the mapping of data between the surface grid of the coupling client and the solver grid, we of course need functions defining the particular interpolation or projection methods. These functions use data and the positions of the respective points in space as an input. Thus, data mapping becomes an (technically) easy task as soon as these positions are known. Using octree-like structures, the algorithm for finding neighbourhood relations is very similar to the algorithms for grid generation from a surface description of the computational domain as described above [3]. In both algorithms, we start on the coarsest grid level (large cell containing the whole computational domain) and recursively go down to deeper/finer levels. In doing so, we inherit all information already determined for the father cell to the son cells. In the case of grid generation it is the 'in/out/on the boundary' information, in case of neighbourhood relations it is the information about grid elements (nodes, faces, cells) of both grids involved that are contained in the current cell. Such, we only have to refine or update the inherited information in the son cells. For example, to find out which elements of the triangulation describing the coupling surface in FSI*ce are contained in the current cell, we only have to look for candidates among the cells already contained in the father cell instead of all triangulation cells. This makes the finding of neighbouring nodes/elements with the help of octree-like structures such as spacepartitioning grids extremely efficient.

RESULTS

In this section, we present some results obtained with the codes and tools described above. Some of these results have a very preliminary character. More results will be presented at the APCOM'07 in Kyoto, December 3-6, 2007.

Flow Simulations. To show the potential of spacepartitioning grids in connection with stacks as data structures and space-filling curves as ordering mechanism, we start with some simple scenarios. The first one is a spherical domain on which we solve the Poisson equation using a multigrid solver for adaptively refined spacepartitioning grids. Fig. 8 shows the two-dimensional domain and the adaptive grid. The computations were performed on an Itanium 2 IA-64 processor with 1.3 GHz, 256 MByte L2-cache, and 8 GByte RAM.
Table 1 shows the measured performance values. As expected, the cache-hitrate is very high whereas the memory requirement is very low (compare with unstructured grids, e.g.). Another remarkable fact is that the memory per degree of freedom as well as the computing time per iteration and degree of freedom do not depend on whether the grid is regular or not. If we compare the achieved runtimes with highly optimized solvers on regular grids, our solver is slower, but has the advantage that it allows for arbitrarily local adaptive grid refinements without loosing efficiency and, thus, enables the user to substantially reduce the number of computed degrees of freedom. In particular in the case of singularities, one can not do without this feature at all since sufficiently fine regular grids would by far exceed the available memory as well as acceptable runtimes.

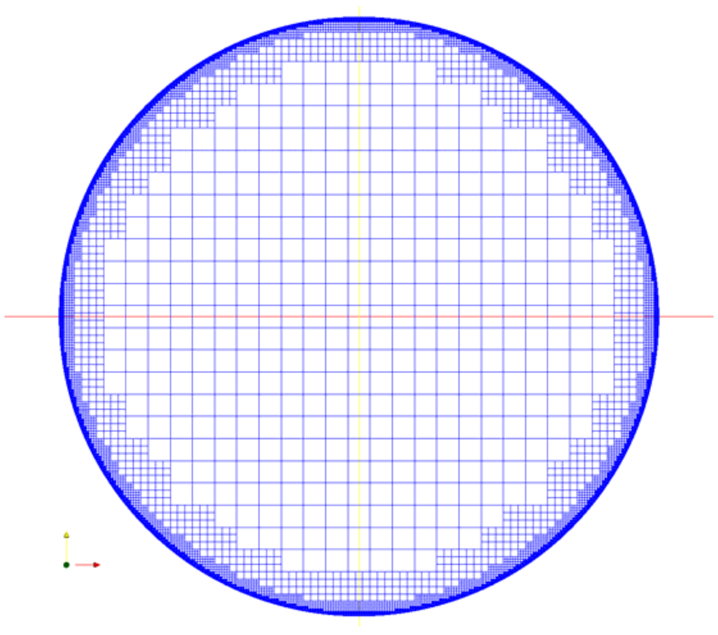


Fig. 8. Adaptively refined spacepartitioning grid in a spherical computational domain.

Table 1. Performance of a multigrid Poisson solver on an adaptively refined spacepartitioning grid in a spherical domain (Fig. 8). The measured values are L2-cache-hitrate, required memory per degree of freedom, and computing time per degree of freedom. The computations were performed on an Itanium 2 IA-64 processor with 1.3 GHz, 256 MByte L2-cache, and 8 GByte RAM.

L2-cache-hitrat e	memory/dof	time/(dof*it)
99%	8 Bytes	4e-6 sec

The second scenario is the DFG cylinder benchmark that is the flow around a cylinder placed in a channel with a slight asymmetry with respect to the middle of the channel, i.e. a little below the middle. This results in two force components acting on the cylinder: a drag and a lift force. Fig. 9 shows a cutoff of the computational domain with the adaptive spacepartitioning grid and the computed forces. The results agree very well with the benchmark values.

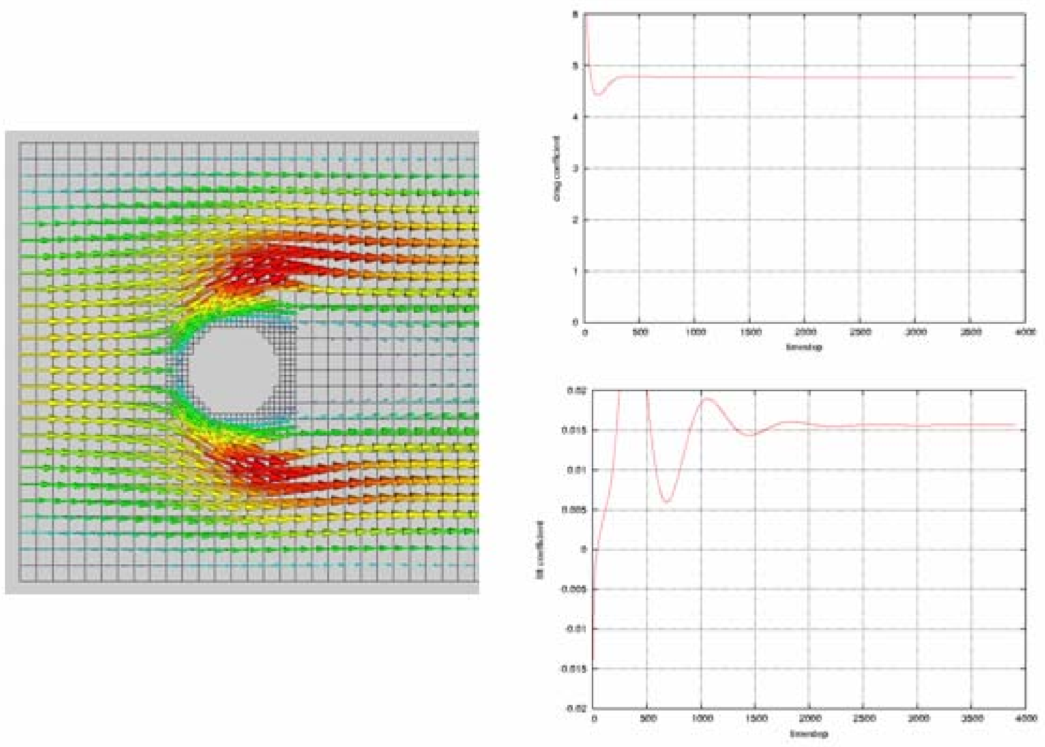
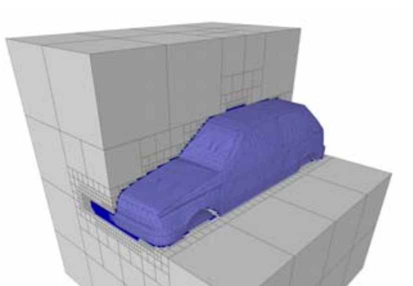


Fig. 9. DFG cylinder benchmark setup (steady-state) on an adaptively refined grid for Reynoldsnumber 20. Left: cutoff of the computational domain with the used adaptive spacepartitioning grid. Right: computed drag (upper graph) and lift (lower graph) forces.

Grid Generation . To show the efficiency of the grid generation algorithm, we use a surface triangulation of a car geometry and generate an octree grid with different maximal refinement depths from it. Here, the local refinement was completely controlled by the geometry, that is the refinement stopped as soon as either the cell belong completely to the inner or outer part of the domain or the maximal refinement depth was reached. Fig. 10 shows the triangulation together with a generated octree and the numbers of nodes and the runtime for different maximal refinement levels.

Coupled Simulations. As an application example for a coupled simulation, we show the simulation of a movement of a particle in asymmetrically shaped pore at the length scale of micrometers (compare Fig. 2). The fluid is pumped forward and backward within this pore by a pressure pump. Experiments showed that the asymmetry of the pore geometry together with the oscillating fluid flow lead to a directed movement of



ref. depth	time	number of

		nodes
7	0.8 sec	203,905
9	4.9 sec	3,288,225
11	48.2 sec	52,662,337
13	662 sec	842,687,105

Fig. 10. Right: surface triangulation of a car together with an octree grid generated from this triangulation; left: runtimes and numbers of nodes for different maximal refinement depths of the generated octree. The computations were performed on a Pentium 4 2.4 GHz, 512 kByte cache.

small particles suspended in the fluid whereas the direction of the movement depends for example on the size of the particle. Such, these pores can be used to separate small particles of different sizes [20]. First simulations of the respective scenario seem to approve the experimental results: Fig. 11 shows the movement of the particle moving an on the axis of the pore during some oscillations of the pressure. We observe an oscillating particle movement with a slight drift of the deflection of the particle.

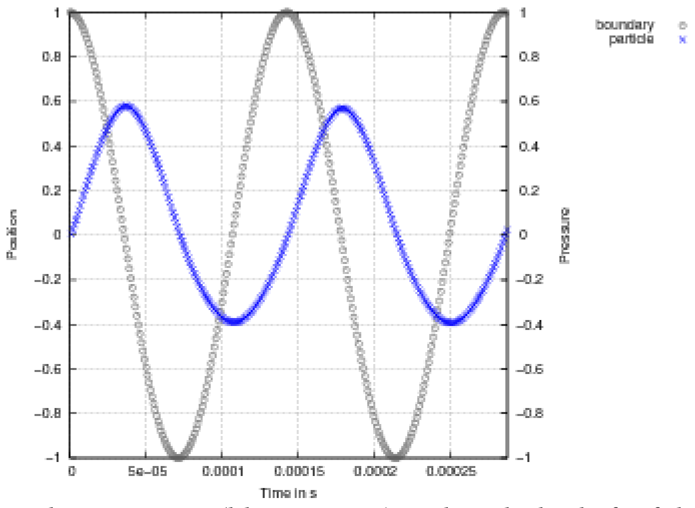


Fig. 11: Oscillating particle movement (blue crosses) with a slight drift of the deflection for a particle moving on the axis of the micro pore shown in Fig. 2 under oscillating pressure.

Acknowledgements The support of the German Research Foundation (project HA-1517 25/1 and HA-1517 25/2 and research group 493) is gratefully acknowledged.

REFERENCES

[1] Y. Di, T. Tang, and P. Zhang, *Moving mesh finite element methods for the incompressible Navier–Stokes equations.* SIAM J. Sci. Comput., 26 (2005),1036–1056.

[2] K. Daubner, *Geometric Modelling with Octrees and Visualization of Simulation Data from Computational Fluid Dynamics (german: Geometrische Modellierung mittels Oktalbäumen und Visualisierung von Simulationsdaten aus der Strömungsmechanik).* Student work, Universität Stuttgart (2005).

[3] H.-J. Bungartz, M. Mehl, *Cartesian Discretisations for Fluid-Structure Interaction – Efficient Flow Solver.* Proceedings ECCOMAS CFD 2006, European Conference on Computational Fluid Dynamics, Egmond an Zee, September 5th-8th, (2006).

[4] F. Günther, M. Mehl, M. Pögl, and Ch. Zenger, *A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves.* SIAM Journal on Scientific Computing 28 (2006), 1634-1650.

[5] C. Blanke, *A Continuity Preserving Finite-Element Discretization of the Navier-Stokes Equations (german: Kontinuitätserhaltende Finite-Elemente-Diskretisierung der Navier-Stokes-Gleichungen).* Diploma thesis, Technische Universität München, (2004).

[6] A. J. Chorin, *Numerical solution of the Navier-Stokes equations.* Math. Comp. 22 (1968), 745-762.

[7] M. Griebel, T. Dornseifer, T. Neunhoeffer, *Numerical Simulation in Fluid Dynamics, a Practical Introduction.* SIAM, Philadelphia, (1997).

[8] T. Weinzierl, *A Cache-Optimal Implementation of a Navier-Stokes Solver under consideration of Physical Conservation laws (german: Eine cache-optimale Implementierung eines Navier-Stokes Lösers unter besonderer Berücksichtigung physikalischer Erhaltungssätze).* Diploma thesis, Technische Universität München, (2005).

[9] H. Sagan, *Space-filling curves.* Springer, New York, (1994).

[10] G. W. Zumbusch, *Adaptive Parallel Multilevel Methods for Partial Differential Equations.* Habilitationsschrift, Universität Bonn, (2001).

[11] F. Günther, *A Cache-Optimal Implementation of the Finite-Element Method (german: Eine cache-optimale Implementierung der Finite-Elemente-Methode).* Doctoral thesis, Technische Universität München, (2004).

[12] M. Pögl, *Development of a Cache-Optmal 3D Finite-Element-Method for Large Problems (german: Entwicklung eines cache-optimalen Finite-Element-Verfahrens für große Probleme.* Doctoral thesis, Technische Universität München, (2004).

[13] A. Krahnke, *Adaptive Higher Order Methods on Cache-Optimal Datastructures for Three-Dimensional Problems (german: Adaptive Verfahren höherer Ordnung auf cache-optimalen Datenstrukturen für dreidimensionale Probleme).* Doctoral thesis, Technische Universität München, (2005).

[14] M. Mehl, T. Weinzierl, and Ch. Zenger, *A cache-oblivious self-adaptive full multigrid method.* Numerical Linear Algebra with Applications, 13, Special Issue: Multigrid Methods, R. D. Falgout (ed.), Wiley Interscience, (2006), 275-291.

[15] M. Griebel, G. W. Zumbusch, *Hash-based parallel multilevel methods with space-filling curves.* In: Rollnik, Wolf (eds.): NIC Symposium 2001, NIC Series 9 (2002), 479-492.

[16] M. Langlotz, *Parallelization of a Cache-Optimal 3D Finite-Element-Methods (german: Parallelisierung eines Cache-optimalen 3D Finite-Element-Verfahrens*). Diploma Thesis, Technische Universität München, (2004).

[17] H.-J. Bungartz, M. Mehl, and T. Weinzierl, *A Parallel Adaptive Cartesian PDE Solver Using Space-Filling Curves.* In: Euro-Par 2006 Parallel Processing, Lecture Notes in Computer Science, 4128, Springer, Berlin Heidelberg, (2006), 1064-1074.

[18] M. Brenk, H.-J. Bungartz, M. Mehl, and T. Neckel, *Fluid-Structure Interaction on Cartesian Grids: Flow Simulation and Coupling Environment.* In: Bungartz, Schäfer (eds.), LNCSE 53 Fluid-Structure Interaction series, Springer, (2006), 233-269.

[19] M. Brenk, H.-J. Bungartz, and T. Neckel, *Cartesian Discretisations for Fluid-Strukture Interaction -- Consistent Force].* In: Wesseling, Onate, Periaux (eds.), ECCOMAS CFD 2006, European Conference on Computational Fluid Dynamics, (2006).

[20] R. D. Astumian, P. Hänggi, *Brownian motors.* Phys. Today 55 (2002), 33–39.