CFD simulations using an AMR-like approach in the PDE framework Peano

◯ Tobias Neckel, Inst. f. Informatik, TU München, 85748 Garching, Germany, E-mail: neckel@in.tum.de
Miriam Mehl, Inst. f. Informatik, TU München, 85748 Garching, Germany
Hans-Joachim Bungartz, Inst. f. Informatik, TU München, 85748 Garching, Germany
Takyuki Aoki, GSIC, Tokyo Institute of Technology, 2-12-1 O-okayama, Tokyo, Japan

The proposed PDE framework Peano can provide a combination of both an easy access to dynamical adaptive Cartesian grids and a good cache performance. The AMR-like mechanism of spacetree formulations using space-filling curves and a stack data concept allows for the realisation of adaptivity with extremely low memory requirements. Results for a low-order finite element approach are presented for a laminar cylinder benchmark scenario. A first integration of a higher order IDO discretisation scheme for the continuity equation shows the flexibility and straight-forward usability of the framework Peano.

1. Introduction

Modern PDE simulations in general and CFD solvers in particular still face capital challenges concerning performance and re-usability aspects. Despite increasing computer power and parallisation of algorithms, applications such as high-resolution turbulent flows or coupled simulations demand a high performance of the solvers in use. Besides the performance issue, the aspect of development time needed for the corresponding programs plays an important rule. Completely different applications have similar needs such as to handle dynamical and possibly large changes in the underlying grid without a large computational overhead. These grid changes may be caused by individual dynamical adaptivity criteria for the accurracy of the solution or by geometry changes in the domain (in the case of fluid-structure interactions e.g.; see [4]). Therefore, PDE frameworks try to offer a common basis where one may plug in an application relying on a lot of features already available. If done properly this saves development time by avoiding reimplementation (and debugging) of the same problems while keeping a good performance. Our approach to tackle these tasks within the object-oriented C++ PDE framework Peano is the usage of adaptive Cartesian grids via spacetrees, space-filling curves and stack data structures.

2. Adaptive Cartesian Grids using Spacetrees

The basic idea of our approach in Peano is to create an adaptive Cartesian grid via a spacetree approach. This is similar to AMR (cf. [3, 2]) but provides a full grid hierarchy in the complete domain. The grid construction starts with a hypercube root cell containing the complete computational domain and then subdivides the root in a recursive manner by splitting up each cell into three parts along each coordinate axis [1]. Thus, adaptive grids can be constructed easily and fast also for complicated geometries. Figure 1 shows an example of a two-dimensional, implicitely defined pore geometry with a circular particle inside.

In order to serialise the underlying spacetree for a run over the grid, a certain traversal order is necessary. There are, of course, a lot of possibilities, but some of them have certain advantages. The so called space-filling curves (cf. [14]), which have been used in parallel computations due to good dynamic load balancing properties, offer additional features concerning an efficient usage of caches in modern computer architectures. These self-similar recursively defined curves fit perfectly to our construction of the spacetree and grid. In Figure 2, a discrete iterative of the Peano curve is visualised with the corresponding adaptive Cartesian grid on three refinement levels.

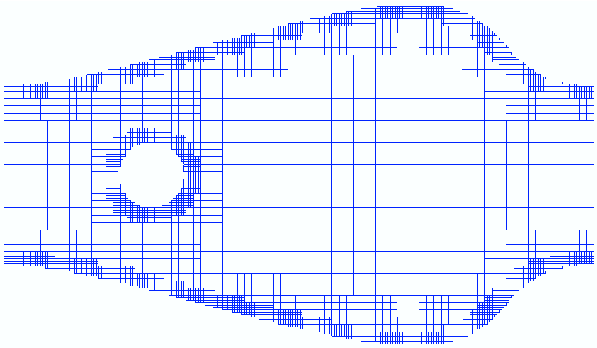


Fig. 1: Visualisation of a 2D adaptive Cartesian grid constructed from a pore geometry in Peano.

The access of data associated with cells is completely straight forward since the Peano curve gives the ordering and each cell is visited only once. For accessing nodal data, we developed a very cache-efficient mechanism using simple stack data structures (see [8]).
A stack is a pile of data packages that allows just two operations: *push* a new package to the top of the pile, and *pop* the top-most package for usage. Each data package holds all degrees of freedom and grid administration data necessary on a particular node. Investigation of the data access order during a traversal of the grid along the Peano curve shows exactly such a linear "pile-up-pile-down" structure. In Figure 3, this is visualised for the case of a regularly refined grid. Two stacks are needed, besides the input and output stream: one for the nodal data located on the left-hand side of the Peano curve (red vertices), and one for data on the right-hand side of the curve (green vertices). After one grid run, input and output stacks are switched and the next grid run in opposite direction along the Peano curve may start. In case of hierarchical data, two stacks are no longer sufficient, but the same mechanism works with a moderate number of stacks (cf. [13]). The reason for the usage of the Peano curve and the associated three-partitioning of the grid cells for refinement are the special projection and palindrome properties of this curve, that are necessary to port the stack data concept also to 3D.

Using the stacks in combination with the Peano curve assures temporal and spatial locality of data access. The locality properties of the Peano curve guarantee a good temporal locality, that is a short time period between

[1] The reason for this three-partioning instead of the common two-partitioning (such as used for octrees, e.g.) will be explained below.

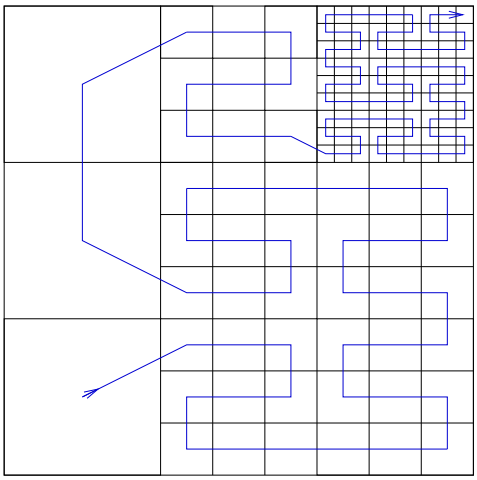


Fig. 2: Example of a 2D adaptive Cartesian grid and its corresponding discrete Peano curve.

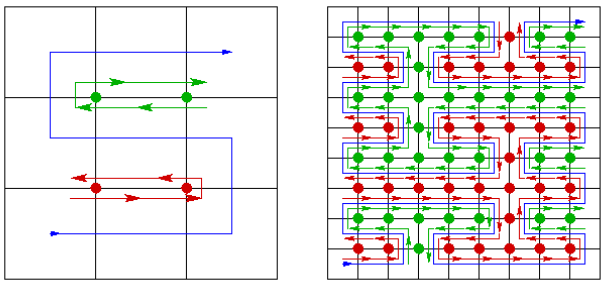


Fig. 3: Visualisation of the linear data access suitable for stacks in combination with the Peano curve. Nodes left (red) and right (green) of the curve can be accessed in an "up-down" manner on two different stacks.

the first and the last usage of a datum during a grid run (cf. [17]). The stacks provide a strictly linear data access in physical memory needing only a minimum number of jumps (when all data of a cache line is completely treated, e.g.), thus assuring an optimal spatial locality. These two effects lead to a very efficient cache usage with cache-hit rates of 98 – 99%, independent of the dimension, the architecture or the concrete type of application in use.

This data access concept depends on a strict cell-wise operator evaluation. This avoids the storage of any neighbourhood informations in particular at boundaries between different refinement levels and, thus, saves a lot of memory. A short description of the operator mechanism in flow simulations is given in Section 3.. In addition, the stacks allow for a straight-forward dynamic modification of the grid. If on a certain leaf of the spacetree (i.e. a non-refined grid cell) during a grid run a further refinement is necessary, the corresponding child cells simply are created with all corresponding nodal data and the new data packages are just put onto the respective stacks. In an analogous manner, coarsening of grid parts can be handled. Furthermore, the spacetree concept corresponds exactly to the need of geometric multigrid solvers. The necessary grid hierarchy as well as the corresponding degrees of freedom are already available throughout a run over the grid (cf. [13]).

The approach of adaptive Cartesian grids generated via spacetrees using the space-filling Peano curve in combination with stack data structures is completely independent both of the concrete PDE one wants to solve and of the type of spatial discretisation (FD, FE, FV, etc.). Therefore, Peano represents a general framework for different solvers, offering grid generation, data handling etc. Currently, several solvers exist, such as for the Poisson equation, for heat equation simulations, for the continuity equation, and for the incompressible Navier-Stokes equations.

3. Peano's Flow Solver

In this section, we will briefly describe our approach for CFD simulations in the Peano framework. The flow solver component of Peano has been developed in order to solve the two- and three-dimensional incompressible Navier-Stokes equations

$$\vec{u}_t + (\vec{u}\cdot\nabla)\vec{u} - \frac{1}{Re}\Delta\vec{u} + \nabla p = 0 \qquad (1)$$

$$\nabla\cdot\vec{u} = 0. \qquad (2)$$

We currently tackle mainly laminar flow scenarios, but there exists also a Smagorinsky LES model.

3.1 Discretisation Scheme

The algorithm in use is a semi-implicit Chorin-like scheme with a MAC approach (cf. [15]). In the partially staggered Cartesian grids of Peano, we use d-linear finite elements for the velocities in the momentum equations (1) associated to grid nodes, while pressure cell values serve as discrete Lagrangian multipliers for respecting the continuity equation (2). The time integration is realised via explicit methods. Currently, we use forward Euler and classical Runge-Kutta. For each time step, we need to solve the corresponding Pressure Poisson equation (PPE). We use the PETSc toolkit[2] to solve this linear system of equations. This offers the possibility to evaluate a variety of different solvers and preconditioners. We currently work on the integration of the built-in geometric multigrid method (available in the Poisson Equation solver component) into the flow solver.

3.2 Adaptive Grids and CFD Operators

As adaptively refined grids imply the existence of boundaries between different refinement depths of the grid and, therewith, of hanging nodes (see Figure 4), we briefly explain how the finite element discretisation of the differential operators in the Navier-Stokes equations (1) and (2) is realised in a natural and efficient way in Peano. In particular, there is no need to explicitely compute and store specialised discretisation stencils at such boundaries. Instead, the same cell-wise discretisation operators are applied to each cell, which is possible since the cell-wise operators only rely on the existence of the cells vertices and not on neighbouring cells. After the accumulation of the cell-wise operator contributions at all cell vertices, we simply have to restrict operator values at hanging nodes (which do not hold real degrees of freedom) to the respective father nodes. This can be done in a fast and natural way within the same run over the grid tree as the evaluation of the cell-wise operators itself. Unlike other approaches, this works without restrictions also for arbitrary level hanging nodes, that is for hanging nodes whos father nodes are hanging nodes again (see Figure 4).

3.3 Consistent forces

It is often necessary to compute forces exerted by the fluid onto solid obstacles. Especially in fluid-structure interaction simulations, the force values at the geometric interface carry the coupling information from the fluid to the structure(s). This implies that a very accurate computation of the forces is important. Therefore, we use the method of consistent forces (see [5, 7]). Hereby,

[2] http://www.mcs.anl.gov/petsc

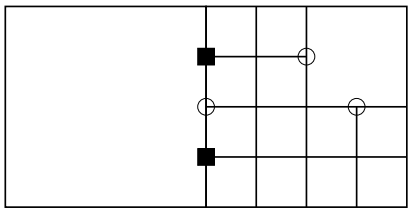


Fig. 4: Visualisation of level-1 (◯) and level-2 (■) hanging nodes in two dimensions.

we just have to use the semi-discrete momentum equations at the boundary nodes where the forces shall be computed. The occurring operators are evaluated already for the flow simulation analogue to the operator evaluation at inner fluid nodes. Thus, the consistent force computation allows us to compute accurate force data without any additional costs. This method is used for the results in Section 5.2.

4. Integration of the IDO Concept into Peano

This Section contains a brief survey on the method of Interpolated Differential Operators (IDO), the motivation for its integration into the Peano framework, and the first realisation steps.

4.1 Survey on IDO

The IDO method has been developed as an alternative scheme for higher order spatial discretisations (cf. [1]), different from common methods such as finite differences or finite volumes. The basic idea is to represent the dependent variables as higher order piecewise polynomials on a discrete spatial grid. Combining both dependent variables and their spatial derivatives in a suitable manner allows for a construction of a higher order approximation in space of the differential operators of the PDE. This approach fits well to the usage of Cartesian grids, where just the mesh sizes of the different cells may be different.

There have been several applications and developments for the IDO scheme (see [9, 11, 10], e.g.). Concerning flow problems, in particular the recent development of a conservative IDO form (IDO-CF, see [12]) seems to be very promising. The conservation of both momentum and energy is an important feature, in particular in turbulent flow problems.

4.2 Motivation of IDO for Peano

In the context of the Peano framework, the IDO concept is very interesting for several reasons. It has been developed on Cartesian grids in 2D and 3D which fits well to the Peano setup. Furthermore, IDO represents a higher order discretisation scheme in space compared to the low-order FEM that Peano's flow solver currently supports. To compare results for several scenarios using identic grids (and the identic solver framework) allows for direct "fair" measurements concerning the cost-benefit ratio of these two spatial discretisations of different order. And finally, the usually nodal-based evaluation of the higher-order IDO operators is quite different from the basic cell-wise manner for which Peano and in particular its flow solver have been set up. Thus, the integration of IDO represents a proof of concept that Peano is sufficiently flexible for other type of discretisations.

4.3 Realisation of IDO in Peano

In a first step, we have implemented the IDO (central) discretisation scheme for the two-dimensional continuity equation for regular Cartesian grids in 2D. The continuity equation

$$\frac{\partial f}{\partial t} = -\frac{\partial (f \cdot u)}{\partial x} - \frac{\partial (f \cdot v)}{\partial y} \tag{3}$$

is considered on a unit square domain $\Omega = [0,1]^2$ with periodic boundary conditions on all $\delta\Omega$. This intermediate step has been chosen for several reasons. First, the continuity equation is simpler to simulate due to direct updates without the need to solve a linear system of equations etc. Second, it allows for a comparison of the numerical results after one periodic cycle with the initial solution. Third, this scenario comprises already all basic important IDO features, such as storage locations and data access structure for the discrete operators, that are necessary for a discretisation of the Navier-Stokes equations.

The handling of the necessary degrees of freedom has been included into Peano. Thus, not only corner and cell data may now be processed, but also edge data, necessary for IDO, can be accessed while physically still lying on vertices. Furthermore, the IDO scheme originally uses a nodal operator evaluation with stencils that comprise more than one direct neighbour due to the higher order approach. Figure (5) gives an exemplary survey of the data access structure for one of the several discrete operators in use. The red circles show the data used to update a corresponding nodal value visualised with a black box. We modified each discrete operator to be evaluated in a cell-wise manner fitting to the Peano's needs. The focus of the ongoing work is now on the integration of the IDO-CF scheme to solve the full Navier-Stokes equations.

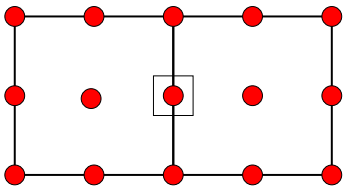


Fig. 5: Visualisation of the nodal IDO data access structure on edge nodes. The discrete operator uses data on neighbouring nodes (red circles) for an update of a target edge node (black square).

5. Numerical Results

In this section, we present results on the memory requirements of the CFD solver in Peano as well as different simulation runs for a laminar cylinder flow benchmark. Finally, first results for the continuity equation obtained with the IDO scheme in Peano are given.

5.1 CFD Memory Requirements

As all data in Peano is located either in vertex- or cell-associated data packages that are stored on stacks, it is interesting to compare and validate the corresponding low memory requirements. Since degree of freedom data and grid management data are located in the same package for each cell or vertex—i.e. there is only one corresponding class holding these data—the construction of the C++ classes by hand would be troublesome and error-prone. Therefore, the tool DaStGen (cf. [6]) has been developed, that performs this flattening of data from (possibly hierarchical) input files into C++ data classes in an automatic manner. DaStGen not only simplifies the development of your degrees of freedom but also allows for data compression of `bool` or `enum` type attributes which usually are allocated at certain memory borders (a `bool`, e.g., usually is not just one bit as one would assume but one byte). These data compression features have been used for the following results in

Peano's flow solver on a 32 bit architecture.

The main interest, besides obtaining insight into the absolute storage necessities, is to compare the amount of data for the grid administration with the data that is necessary due to our flow algorithm (and would also be needed on any other type of grid). Therefore, we first list the latter one. Table 1 shows the different variables on a fluid cell necessary for the Chorin-like update scheme. The pressure (p), the right-hand side of the Pressure Poisson equation (rhs PPE), and the type of a cell represent the minimum amount of data for the algorithm to work. We additionally store its number due to the explicit solution of the PPE which will become obsolete for multigrid usage. In total, we end up with 24 bytes of CFD data on a cell, both in 2D and 3D.

Tab. 1: Variables in use for a cell degree of freedom in Peano's flow solver. The same data is needed in 2D and 3D. The last column shows if the data is supplementary to the minimum amount needed by the algorithm.

variable	type	bytes	suppl.
p	`double`	8	-
rhs PPE	`double`	8	-
type	`int`	4	-
number	`int`	4	yes
total		24	

Concerning a fluid vertex degree of freedom, we list the corresponding variables in Table 2. Again, the velocities (u), the intermediate velocities (F), the pressure gradient (∇p), the support (or mass matrix contribution, A) and the type of the vertex are always necessary for the flow simulation. The vertex number is stored due to postprocessing purposes. The type of boundary as well as the force data are currently saved on each node as we do not yet distinguish boundary node data from inner nodes. The total amount of necessary storage, thus, is 80 bytes in two and 108 bytes in three dimensions.

Tab. 2: Variables in use for a cell degree of freedom in Peano's flow solver. The last column shows if the data is supplementary to the minimum amount needed by the algorithm.

variable	type	bytes 2D	bytes 3D	suppl.
u	`Vector`	16	24	-
F	`Vector`	16	24	-
∇p	`Vector`	16	24	-
A	`int`	4	4	-
type	`int`	4	4	-
number	`int`	4	4	yes
boundary	`int`	4	4	yes
force	`Vector`	16	24	yes
total		80	108	

Now, we compare that total amount of data with the measurements. A survey of these data for two and three dimensions can be found in Tables 3 and 4. The second column shows the measured data size of the corresponding classes generated via DaStGen, the third column holds the amount of grid management data when the necessary degree of freedom for CFD computations is subtracted.

This clearly indicates the low memory requirements of our approach for adaptive, dynamic Cartesian grids in Peano: Only 40 and 45 per cent of the overall data on each cell in 2D and 3D is needed for grid administration. On vertices, this amount reduces even to 9 and 10 per cent for the corresponding dimensions.

Tab. 3: Memory requirements in bytes for Peano's CFD component in 2D. For a cell or vertex, the total size is given as well as the data for grid management and CFD computations (the latter split into total amount of memory, minimum number required by the algorithm, and supplementary data, see Tables 1 and 2).

data type 2D	total	grid	CFD		
			total	min	suppl.
cell	40	16	24	20	4
vertex	88	8	80	56	24

Tab. 4: Memory requirements in bytes for Peano's CFD component in 3D. For a cell or vertex, the total size is given as well as the data for grid management and CFD computations (the latter split into total amount of memory, minimum number required by the algorithm, and supplementary data, see Tables 1 and 2).

data type 3D	total	grid	CFD		
			total	min	suppl.
cell	44	20	24	20	4
vertex	120	12	108	76	32

5.2 Cylinder Benchmark Simulations

In this section, we describe flow results obtained with Peano for the scenario 2D–1 of the benchmark collection "laminar flow around a cylinder" (cf. [16]). We have chosen this benchmark since it allows to compare the flow results with hard reference data. In particular, the drag and lift coefficients c_d and c_l serve as reference data for the evaluation of the simulations' accuracy. All simulations have been performed on a priori refined grids with the geometry surface as the refinement criterion. The serial runs have been carried out on an Intel Pentium 4 architecture with a single processor of 3.4 GHz, 1 MB level-2 cache, and 2 GB RAM. We used, the gcc 4.1.1[3] with aggressive optimisation.

The benchmark scenario 2D–1 represents a steady-state flow around a cylinder placed in the front part of a two-dimensional channel at Reynolds number 20. The channel has a height and length of 0.41 and 2.46, respectively. The accuracy depends of course both on the resolution of the cylinder (i.e. the maximum grid level of the spacetree) and on the solution accuracy in the whole domain (depending on the discretisation order, the mesh resolution, and the adaptivity pattern).

Table 5 shows the results for a suite of scenarios that have all the same minimum spacetree level five corresponding to the coarsest mesh size $h_{\max} = 0.0455$. Around the cylinder, the grid is refined up to a maximum refinement level of nine (i.e. $h_{\min} = 0.00056$). Figure 6 shows a cut-off of the corresponding grid for a maximum grid level of eight. The "steep" adaptivity at the cylinder geometry (and there only) results in a quite moderate raise in the number of degrees of freedom.

[3] See http://gcc.gnu.org/.

Tab. 5: Simulation results of the benchmark 2D–1 [16] computed on grids with a steep adaptive refinement around the cylinder. The maximum and minimum level of refinement (l_{max} and l_{min}) represent the finest and coarsest mesh size in use. The drag (c_d) and lift (c_l) coefficients of the cylinder as well as the runtime of one time step in seconds are shown in the last three columns.

l_{max}	l_{min}	cells	vertices	c_d	c_l	time
6	5	1622	1309	5.656	0.0324	0.060
7	5	1953	1578	5.521	0.0220	0.075
8	5	3017	2466	5.569	0.0158	0.100
9	5	5469	4438	5.540	0.0156	0.180
ref.	-	-	-	5.580	0.0107	-

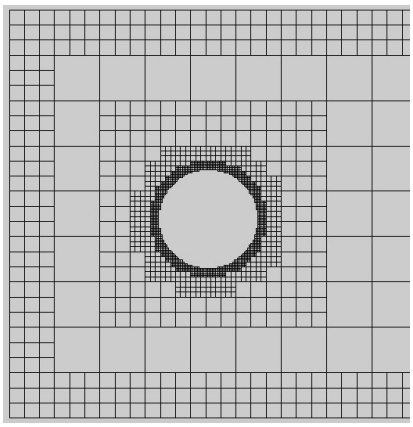


Fig. 6: Visualisation of the adaptive grid refinement of the benchmark flow around a cylinder 2D–1 [16]. The maximum refinement level is 8 with coarsest level 5.

Since the coarsest mesh cells are quite close to the cylinder (see Figure 6), the coarse (and, therefore, inaccurate) flow data have a non-negligible negative impact on the force coefficients, even if the resolution of the cylinder geometry is quite fine. This may be remedied by the usage of a posteriori error estimators for dynamic adaptivity. Apart from accuracy considerations, Table 5 also shows that the runtimes per time step grow proportionally to the number of degrees of freedom. Thus, the adaptivity pattern and, in particular, the balancedness of the underlying spacetree do not have any influence on the runtime per degree of freedom or, in other word, we do not pay a price for complicated adaptivity patterns in terms of runtime.

We have run an additional sequence of simulations with a more regular refinement in the overall domain (see Figure 7) in order to observe the effect of a more accurate solution.

Tab. 6: Results of adaptive simulations of the benchmark 2D–1 [16] with smaller coarsest mesh size (i.e. higher minimum spacetree refinement level l_{min}).

l_{max}	l_{min}	cells	vertices	c_d	c_l	time
6	6	4342	4167	5.678	0.0443	0.08
7	7	39073	38494	5.558	0.0135	0.69
8	7	39973	39222	5.684	0.0147	0.71
9	7	42501	41270	5.591	0.0113	0.78
ref.	-	-	-	5.580	0.0107	-

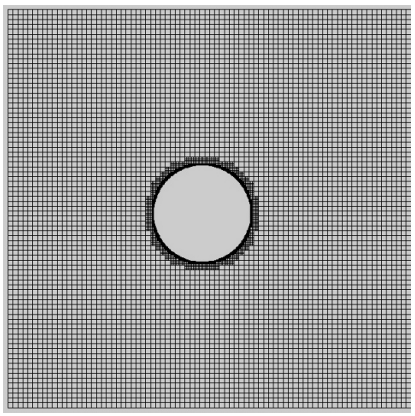


Fig. 7: Visualisation of the adaptive grid refinement of the benchmark flow around a cylinder 2D–1 [16]. The maximum refinement level is 9 with coarsest level 7.

The results of these simulations are shown in Table 6. The first two scenarios with level six and seven, respectively, represent a regular Cartesian grid (treated with the adaptive algorithm). The second two scenarios with maximum spacetree refinement level eight and nine, respectively, both keep the minimum level of seven and use additional adaptive refinement at the cylinder. We observe a good convergence of the force coefficient towards the reference values. As expected, a further adaptive refinement at the cylinder results in a higher accuracy of the forces if the difference between the maximum and minimum refinement level is not too large (compare level nine in Tables 5 and 6).

To get a feeling for the benefit of grid adaptivity, we compare setups of Tables 5 and 6 with similar accuracies of the force coefficients: Lines one and two in Table 6 (i.e. level six and seven) correspond to lines one and three in Table 5 (i.e. maximum level six and eight). The number of required degrees of freedom reduces by a factor of about three and fifteen, respectively, in the case of real adaptivity.

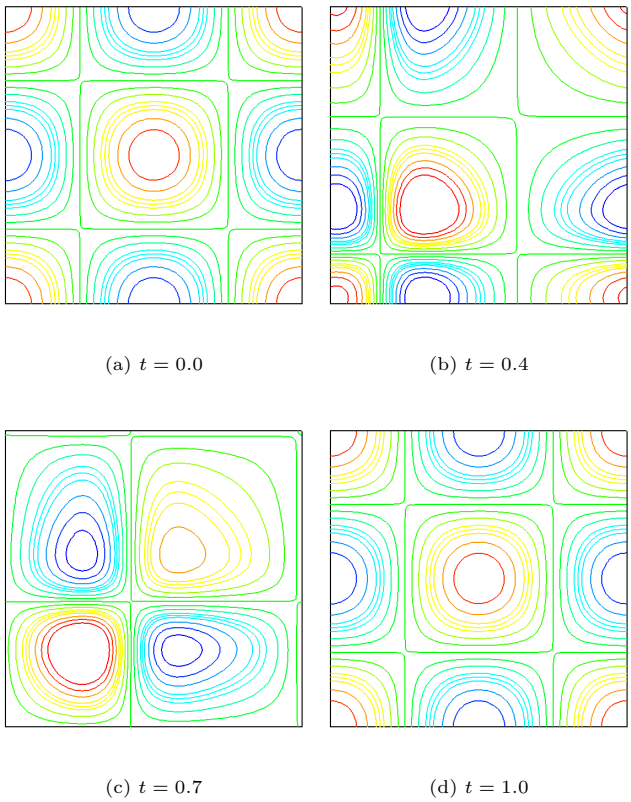
5.3 IDO: First Results

As a first step for using IDO in Peano for flow simulations, we integrated this concept for the 2D continuity equation (3) on regular Cartesian grids (see Section 4.). We used a compressing and decompressing velocity field $u(x, y)$,

$$\begin{aligned} u_x(x) &= \frac{1}{1 + a \cdot \sin(k_x \cdot x)} \\ u_y(y) &= \frac{1}{1 + a \cdot \sin(k_y \cdot y)}, \end{aligned}$$

with an amplication factor $a = 0.2$ and $k_x = k_y = 2\pi$. The initial value for the dependent variable f is set to $f_0(x, y) = \cos(k_x \cdot x) \cdot \cos(k_y \cdot y)$. The time step size for the explicit classical Runge-Kutta scheme has been chosen as $\tau = 0.1 \cdot h_{min}$, due to stability reasons of this explicit update scheme. We simulate one periodic cycle with a final time of $t_{end} = 1.0$.

For a grid of 100×100 cells on the unit square $[0, 1]^2$, Figure 8 shows contour lines at startup, for $t = 0.4$, $t = 0.7$, and at $t = t_{end}$. The colours correspond to the values of the dependent variable f. The final IDO solution recovers the initial startup data f_0 well, which may also be seen from the absolute discrete L2-error of $\sqrt{9.03e-07}$.



(a) $t = 0.0$ (b) $t = 0.4$

(c) $t = 0.7$ (d) $t = 1.0$

Fig. 8: Simulation results of the continuity equation (3) on $[0, 1]^2$ obtained with IDO in Peano. The contour lines are given at $t = 0.0$ (a), $t = 0.4$ (b), $t = 0.7$ (c), and $t = 1.0$ (d), where the colours represent the value of the dependent variable f.

6. Conclusion

In this paper, we presented the flexible framework Peano for PDE simulations on Cartesian grids in general, and for CFD in particular. We showed that Peano's spacetree mechanism combined with the space-filling Peano curve and a stack data structure concept is suitable for flow simulations on adaptive Cartesian grids resulting in high cache-hit rates and very low memory consumption. The results of the cylinder benchmark computations show that the flow solver converges well to the reference data on the suites of a priori adaptive grids. First results using the IDO scheme in Peano for the continuity equation indicate that the current integration of IDO-CF into Peano's flow solver will not encounter major technical problems. Thus, Peano is a very promising framework for CFD simulations, and for solving PDEs in general, as it combines easy access to dynamic adaptive Cartesian grids for different sorts of discretisation schemes with good performance results.

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.

参考文献

1. T. Aoki. Interpolated Differential Operator (IDO) scheme for solving partial differential equations. *Comput. Phys. Comm.*, 102:132–146, 1997.

2. M. J. Berger and P. Collela. Local adaptive mesh refinement for shock hydrodynamics. *Journ. Comput. Phys.*, 82:64–84, 1989.

3. M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journ. Comput. Phys.*, 53:484–512, 1984.

4. M. Brenk, H.-J. Bungartz, M. Mehl, and T. Neckel. Fluid-structure interaction on cartesian grids: Flow simulation and coupling environment. In H.-J. Bungartz and M. Schäfer, editors, *Fluid-Structure Interaction*, number 53 in LNCSE, pages 233–269. Springer, 2006.

5. M. Brenk, H.-J. Bungartz, and T. Neckel. Cartesian discretisations for fluid-structure interaction – consistent forces. In P. Wesseling, E. Oñate, and J. Périaux, editors, *ECCOMAS CFD 2006, European Conference on Computational Fluid Dynamics*. TU Delft, 2006.

6. H.-J. Bungartz, W. Eckhardt, M. Mehl, and T. Weinzierl. Dastgen - a data structure generator for parallel c++ hpc software. In Bubak, van Albada, Sloot, and Dongarra, editors, *ICCS 2008 Proceedings*, Lecture Notes in Computer Science, Heidelberg, Berlin, June 2008. Springer-Verlag.

7. P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method*. John Wiley & Sons, 1998.

8. F. Günther, M. Mehl, M. Pögl, and C. Zenger. A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves. *SIAM J. Sci. Comput.*, 28(5):1634–1650, 2006.

9. Y. Imai and T. Aoki. Accuracy study of the IDO scheme by Fourier analysis. *Journ. Comput. Phys.*, 217:453–472, 2006.

10. Y. Imai and T. Aoki. A higher-order implicit IDO scheme and its CFD application to local mesh refinement method. *Comput. Mech.*, 38:211–221, 2006.

11. Y. Imai and T. Aoki. Stable coupling between vector and scalar variables for the IDO scheme on collocated grids. *Journ. Comput. Phys.*, 215:81–97, 2006.

12. Y. Imai, T. Aoki, and K. Takizawa. Conservative form of interpolated differential operator scheme for compressible and incompressible fluid dynamics. *Journ. Comput. Phys.*, 227:2263–2285, 2008.

13. M. Mehl, T. Weinzierl, and C. Zenger. A cache-oblivious self-adaptive full multigrid method. *Numer. Linear Algebr.*, 13(2-3):275–291, 2006.

14. H. Sagan. *Space-filling curves*. Springer, New York, 1994.

15. M. F. Tomé and S. McKee. GENSMAC: A computational marker and cell method for free surface flows in general domains. *J. Comp. Phys.*, 110:171–186, 1994.

16. S. Turek and M. Schäfer. Benchmark computations of laminar flow around a cylinder. In E. H. Hirschel, editor, *Flow Simulation with High-Performance Computers II*, number 52 in NNFM. Vieweg, 1996.

17. G. Zumbusch. Adaptive parallel multilevel methods for partial differential equations. Habilitationsschrift, Universität Bonn, 2001.