Parallel and Adaptive Methods for Fluid-Structure-Interactions

Josef Ballmann, Marek Behr, Kolja Brix, Wolfgang Dahmen, Christoph Hohn, Ralf Massjung, Sorana Melian, Siegfried Müller, and Gero Schieffer

Abstract The new flow solver Quadflow, developed within the SFB 401, has been designed for investigating flows around airfoils and simulating the interaction of the structural dynamics and aerodynamics. This article addresses the following issues arising in this context. After identifying proper coupling conditions and settling the well-posedness of the resulting coupled fluid-structure problem, suitable strategies for successively applying flow and structure solvers needed to be developed that give rise to a sufficiently close coupling of both media. Based on these findings the overall efficiency of numerical simulations hinges, for the current choice of structural models, on the efficiency of the flow solver. In addition to the multiscale-based grid adaptation concepts, proper parallelization concepts are needed to realize for such complex problems an acceptable computational performance on parallel architectures. Since the parallelization of dynamically varying adaptive discretizations is by far not straightforward we will mainly concentrate on this issue in connection with the above mentioned multiscale adaptivity concepts. In particular, we outline the way the multiscale library has been parallelized via MPI for distributed memory architectures. To ensure a proper scaling of the computational performance with respect to CPU time and memory, the load of data has to be well-balanced and communication between processors has to be minimized. We point out how to meet these requirements by employing the concept of space-filling curves.

K. Brix, W. Dahmen, S. Melian, S. Müller
Institut für Geometrie und Praktische Mathematik, RWTH Aachen University, Templergraben 55, 52056 Aachen, e-mail: dahmen,melian,brix,mueller@igpm.rwth-aachen.de

M. Behr, G. Schieffer
Lehrstuhl für Computergestützte Analyse Technischer Systeme, RWTH Aachen University, Schinkelstraße 2, 52062 Aachen, e-mail: schieffer@cats.rwth-aachen.de

J. Ballmann, Ch. Hohn
Lehr- und Forschungsgebiet für Mechanik, RWTH Aachen University, Schinkelstraße 2, 52062 Aachen, e-mail: ballmann,hohn@lufmech.rwth-aachen.de

1 Introduction

In this paper we present an overview of the work conducted in our research group concerning the numerical simulation for fluid-structure interaction problems in the context of aeroelasticity. There have been two major focus points, namely (i) the coupling of fluid and structure solvers, and (ii) parallelization of the fluid solver.

As for (i), a typical approach to solving the coupled fluid-structure problem is the successive application of highly developed solvers for each separate task coupled through appropriate interface conditions. We shall first briefly recall the essence of our findings from an early stage of the research program concerning some foundational issues of coupling mutual fluid and structure response. In particular, this covers well posedness of a coupled problem formulation based on proper coupling conditions as well as the development of discretization concepts that ensure a correct energy balance at the interface. Needless to stress that the latter issue is essential for a time accurate integration of the inherently nonstationary processes. Once suitable coupling strategies for flow and structure solvers had been identified the focus of the work shifted towards improving the performance of the individual solvers.

Here the primary demands arise on the fluid side which brings us to topic (ii). In order to be able to meet the requirements of highly accurate nonstationary simulations based on the full Navier Stokes equations as a fluid model a new fully adaptive solver Quadflow has been developed [6, 7]. More specifically, it has been designed to handle (i) unstructured grids composed of polygonal(2D)/polyhedral(3D) elements [5] and (ii) block-structured grids where in each block the grid is determined by local evaluation of B-Spline mappings [17]. While the solver can handle also grids provided by an external grid generator grid adaptation has been implemented only for block-structured grids where in each block the grid is locally refined using the concept of multiscale-based grid adaptation [19]. In order to treat both settings, two different data structures were developed for the flow solver and the grid adaptation, respectively. The time evolution is performed on *one* unsorted list of *all* cell averages not distinguishing between data that might correspond to different blocks. On the other hand, grid adaptation is carried out for *each* block separately sweeping through the different refinement levels. It turned out that hash maps are well suited for this purpose rather than tree structures. Therefore, in each adaptation step the data have to be transferred back and forth between unsorted lists and hash maps.

Although multiscale-based grid adaptation leads to a significant reduction of the computational complexity (CPU time and memory) in comparison to computing on uniform meshes, this by itself is ultimately not sufficient to warrant an acceptable efficiency when dealing with realistic 3D computations for complex geometries. In addition, parallelization techniques are indispensible for further reducing the computational time to an affordable order of magnitude. In a first step, the unstructured finite volume solver was parallelized via MPI [12, 13] for distributed memory architectures. Here load-balancing was performed by graph partitioning techniques using the Metis software [16, 15] together with PETSc [4, 2, 3]. This parallelized flow solver was used together with the concept of adaptive, block-structured grids. However, in this case all data had to be transferred to *one* processor. Since this ruins

the overall performance, we have parallelized the multiscale library, which realizes local grid refinement in each block by means of the multiscale-based grid adaptation concept.

The performance of a parallelized code crucially depends on load-balancing and minimal interprocessor communication. Since due to hanging nodes, the underlying adaptive grids are unstructured this task is by no means trivial. In contrast to the flow solver, instead of employing graph partitioning methods, we use space-filling curves [25]. Here the basic idea is to map level-dependent multiindices identifying the cells in the grid hierarchy to a one dimensional line. The interval is then split into different parts each containing the same number of entries. For this mapping procedure we employ the same cell identifiers as in case of the hash maps.

After briefly recalling in Section 2 the results concerning fluid-structure coupling, the main objective of the present work is to present the subsequent developments concerning the parallelization of multiscale-based grid adaptation. For this purpose, we first summarize in Section 3 the basic ingredients of the multiscale library: (i) the multiscale analysis of the discrete cell averages and grid adaptation, (ii) algorithms and (iii) data structures.

The key issues of parallelization are load-balancing and interprocessor communication. These issues are addressed in Section 4. An optimal balancing of the computational work load can be realized using the concept of *space-filling curves*. Mainly in connection with local multiscale transformations we discuss the data transfer at processor boundaries. Finally in Section 6, we present some performance studies with the parallelized version of the multiscale-based grid adaptation and show first adaptive, parallel 3D computations of a Lamb-Oseen vortex.

2 Fluid-Structure Coupling

This section is devoted to discretization issues concerning the numerical solution of fluid-structure interaction problems. Numerical solution methodologies for solving such problems are typically based on employing an already available fluid solver and a given structural solver. The task is then to incorporate them into a fluid-structure solver or, referring to our application background, into an aeroelastic solver. Similar to constructing efficient and reliable fluid solvers and structural solvers, by making those solvers obey certain discretization principles, one faces the question, how to configure a good fluid-structure solver from the given individual fluid and structure solvers, or expressed shortly, how to realize a good fluid-structure coupling. For this purpose we shall explain first the setting in which this question has been analyzed.

Consider first the following physical model. The aeroelastic system consisting of an aircraft wing in transonic flow can develop the nonlinear vibration phenomenon of limit cycles which exhibit constant amplitude structural vibrations. It is important to know the parameter ranges, in which these unstable phenomena occur. The limit cycles are self-excited oscillations caused by the aerodynamic coupling of structural modes in linear theory. Among other effects, shock movements in the flow play a

significant role in the destabilizing mechanisms which finally activate nonlinearities. These features can also be found in the panel flutter problem [9], which serves as our model problem. The panel flutter problem analyzed in this work consists of a plate (*panel*) over which a compressible fluid (air) flows at transonic speed. We assume that the panel is infinitely long in the spanwise direction (z–coordinate), so that a 2D flow in the x-y plane passes over a strip of the panel, compare Figure 1. The flow is modeled by the 2D Euler equations of gas dynamics and the structure by a strip of a von–Karman plate. The panel is supported at its ends in fixed hinges. It is placed on the x–axis between solid walls as drawn in Figure 1. Numerical results visualized in [18] show structural deflections and shock movements present in typical limit cycle oscillations for the panel flutter problem.

Second, let us make more precise, what the fluid-structure coupling is about. Typically the fluid-structure solver is based on the following 5 ingredients.

- Fluid solver
- Structural solver
- Geometrical transfer (at nonconforming grid interface)
- Load transfer (at nonconforming grid interface)
- Coupling Scheme $=$ rule to process above 4 steps in time

Here we assume that fluid and structural solvers are given. In the fluid, which is compressible in our case, we have chosen a Finite Volume MUSCL approach and for the structure we have chosen the Finite Element Method, which are currently the most popular methods in use in the respective disciplines. What remains, is to specify geometrical transfer, load transfer and coupling scheme. Together all 5 items define the discrete fluid-structure system with its intrinsic properties. In particular, the grid interface of fluid and structure are typically nonconforming, i.e. the structural nodes and the fluid nodes do not match, and the deflected structure and the deforming fluid grid do not coincide as spatial objects at the interface. As an illustration of this point see Figure 2, where the situation is shown for the panel flutter problem. Here, at least each fluid node can be fixed to a material point of the structure, a property we have employed in our discretization, and which can already be interpreted as the geometrical transfer. Note that the nonconformity is much more involved, when considering a flow around an aircraft wing, having in mind the way in which aircraft wings are modeled, and that deformations in 3D can be more complex. The load transfer determines how the load distribution for the structure is determined from the discrete fluid quantities. Finally as an example for a coupling scheme we show the simplest choice, namely the loose coupling illustrated in Figure 3. Here the fluid state, respectively the structural deflection, on time level n, are denoted by U^n, respectively w^n. One fluid-structure time step is performed by first applying the load transfer (*Step 1*), then advancing the structural solution in time (*Step 2*), then applying the geometrical transfer (*Step 3*), which means to deform the fluid grid according to the structural deformation, and finally by advancing the fluid in time (*Step 4*) on the grid that moves during the time step from its position at the beginning to that at the end of the time step.

The property which we impose on the coupling is that the discrete fluid-structure system satisfies the same energy balance as the continuous system, see (6) below. Obviously this is an important property. A consistent energy transfer between fluid and structure is essential, in particular, when the objective is to accurately determine where unstable behaviour occurs in parameter space, or more generally, when determining bifurcations of the system. Using a Finite Volume Method in the fluid and a Finite Element Method in the structure, we have constructed in [18] a geometrical transfer and a load transfer such that the discrete fluid-structure system satisfies the same energy balance as the continuous system. In fact we have given a discretization of the equations (1)–(5), resulting in a discrete fluid-structure system, implicit in time, such that for each time step a coupled set of fluid-structure equations has to be solved. This is done approximately by iterating the loose coupling within each time step until a convergence criterion is met. In order to give an idea about the energy argument, we restrict ourselves to a description of the continuous model and its energy balance for the time-dependent interaction of a compressible inviscid and non-heatconducting fluid and a linear elastic plate with geometrical nonlinearity according to von-Karman for a plate with fixed ends in two space dimensions.

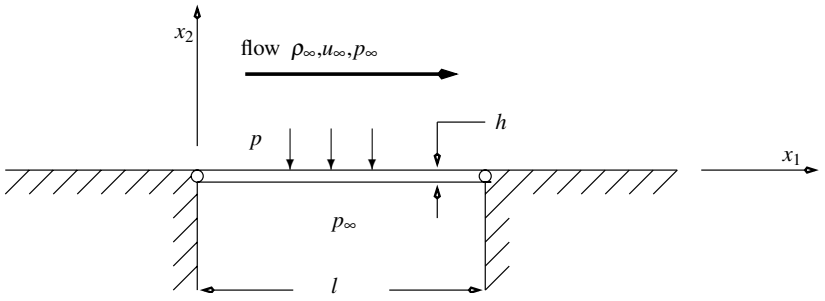

Fig. 1 Geometry of the 2D panel flutter problem.

The fluid equations are

$$\frac{d}{dt}\int_{\Omega(t)} U\,dx_1\,dx_2 \;+\; \int_{\partial\Omega(t)} U\,(\mathbf{v}-\dot{\mathbf{x}})^T\,\mathbf{n}\,ds \;=\; \int_{\partial\Omega(t)} \begin{pmatrix} 0 \\ -p\,n_1 \\ -p\,n_2 \\ -p\,\mathbf{v}^T\mathbf{n} \end{pmatrix} ds \qquad (1)$$

for arbitraryly moving domains $\Omega(t)$. The conservative state vector $U = (\rho, \rho\,u, \rho\,v, \frac{1}{2}\,\rho\,|\mathbf{v}|^2 + \frac{p}{\gamma-1})^T$ involves the density ρ, the velocity vector $\mathbf{v} = (u,v)^T$ and the pressure p. On a point of the boundary $\partial\Omega(t)$, $\mathbf{n} = (n_1, n_2)^T$ is the outward unit normal of $\Omega(t)$ and $\dot{\mathbf{x}}$ is the velocity of a boundary point $\mathbf{x}(t)$.

For the formulation of the plate equation we introduce the space $V := H^2(0,l) \cap H_0^1(0,l)$. The variational formulation is to find the plate deflection $w(t,x)$ satisfying

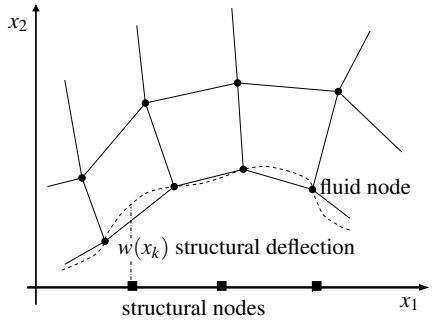

Fig. 2 Representation of the interface in fluid and structure.

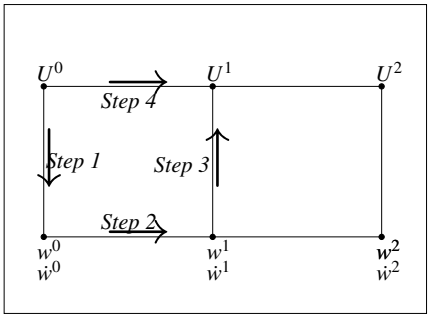

Fig. 3 Loose coupling.

$$D(w_{xx},\varphi_{xx}) + N(w)\cdot(w_x,\varphi_x) + m(\ddot{w},\varphi) \;=\; (p_2 - p_1,\varphi) \tag{2}$$

for all test functions $\varphi \in V$, where the nonlinear term models a restoring force due to the mid-surface stretching with

$$N(w) \;=\; \frac{E\,h}{2\,l}\int_0^l w_x^2\,dx\,.$$

The constants in the panel equation are the stiffness $D = Eh^3/12(1-\nu^2)$, the mass per unit area $m = \rho_s h$, the panel thickness h and length l, the density ρ_s, Young's modulus E and the Poisson ratio ν.

Now the aeroelastic problem can be formulated. For convenience, all interface and boundary conditions are given in the strong sense.

- The fluid domain at time t is defined as the space above the deflecting panel and solid walls and is denoted by $\Omega_F(t)$, i.e. for each fluid interface point $\mathbf{x}(t)$ associated with a panel point $\xi \in [0,l]$,

$$\mathbf{x}(0) \;=\; (\xi,0)^T\,, \quad \mathbf{x}(t) \;=\; (\xi, w(t,\xi))^T\,. \tag{3}$$

- For all moving domains $\Omega(t) \subseteq \Omega_F(t)$ (1) holds.
- (2) holds with the r.h.s. determined from the fluid pressure on top of the panel and p_∞ on the bottom,

$$\begin{aligned} p_1(t,\xi) \;&\equiv\; p(t,\xi,w(t,\xi)), \qquad \xi \in [0,l], \\ p_2 \;&\equiv\; p_\infty\,. \end{aligned} \tag{4}$$

- At infinity we have $\rho = \rho_\infty$, $u = u_\infty$, $v = 0$, $p = p_\infty$.
- On the remaining boundary of $\Omega_F(t)$ the fluid velocity in normal direction $\mathbf{n}$ equals the velocity of the boundary in direction $\mathbf{n}$, in particular on the fluid–structure interface,

$$\mathbf{v}^T\mathbf{n} \;=\; \dot{\mathbf{x}}^T\mathbf{n} \;=\; \dot{w}\,n_2\,. \tag{5}$$

The energy equations for fluid and structure are given by

$$\frac{d}{dt}E_F \;+\; \int_{\partial\Omega_F(t)} \left(\frac{1}{2}\rho\,\mathbf{v}^2 + \frac{p}{\gamma-1}\right)(\mathbf{v}-\dot{\mathbf{x}})^T\mathbf{n}\,ds \;=\; \int_{\partial\Omega_F(t)} -p\,\mathbf{v}^T\mathbf{n}\,ds\,,$$

$$\frac{d}{dt}E_S \;=\; \int_0^l (p_2 - p_1)\,\dot{w}\,dx\,,$$

where we have denoted the energy in fluid and structure by

$$E_F(t) \;\equiv\; \int_{\Omega_F(t)} \frac{1}{2}\rho\,\mathbf{v}^2 + \frac{p}{\gamma-1}\;d\mathbf{x}\,,$$

$$E_S(t) \;\equiv\; \int_0^l \frac{m}{2}\dot{w}^2 + \frac{D}{2}w_{xx}^2\,dx \;+\; \frac{E\,h}{2\,l}\left(\frac{1}{2}\int_0^l w_x^2\,dx\right)^2\,.$$

The energy equation of the fluid is the fourth component of (1) and the energy equation of the structure is derived by plugging $\varphi = \dot{w}$ into (2).

Summation of the two energy equations yields the energy equation of the aeroelastic system. Here we restrict the fluid solution to a finite domain $\Omega_F(t)$ with non–moving inflow and outflow boundary. Due to the boundary and coupling conditions (3), (4), (5), we end up with

$$\begin{aligned} &E_F(t_{n+1}) + E_S(t_{n+1}) - E_F(t_n) - E_S(t_n) \;= \\ &= \int_{t_n}^{t_{n+1}} \left(-\int_{\partial\Omega_{\text{in/outflow}}} \left(\frac{1}{2}\rho\,\mathbf{v}^2 + \frac{p\,\gamma}{\gamma-1}\right)\mathbf{v}^T\mathbf{n}\,ds \;+\; \int_0^l p_\infty\,\dot{w}\,dx\right)\,dt\,. \end{aligned} \tag{6}$$

To illustrate the influence of the coupling on the accuracy when predicting bifurcations in the transonic regime, we show an example taken from [18], where further details are given. Increasing the dynamic pressure at a fixed Mach number of $M_\infty = 0.95$ we consider the panel flutter problem for an aluminium panel for a nondimensional dynamic inflow pressure λ in the range $2000 \le \lambda \le 4000$. A bifurcation in the system behaviour occurs in that range and we run calculations with several coupling schemes at various (fluid)-CFL numbers, to compare how the different schemes manage to track that bifurcation. The bifurcation points obtained are shown in Figure 4 and reveal a strong variation depending on the coupling scheme, when using moderate CFL numbers. Here a CFL number of 50 corresponds to 120 time-steps per limit cycle oscillation and shocks in the fluid moving half the grid size during a time-step. The dotted line shows the λ-value for which each coupling scheme predicts the bifurcation as $\Delta t \to 0$. The method denoted as FPI, solves in each time step the discrete system that satisfies (6), by a fixed-point iteration, which iterates the loose coupling until a convergence criterion is met. On average this happens after two iterations. In contrast the method denoted by LOOSE performs only one such iteration, i.e. it is the loose coupling. The method PRED performs a loose

coupling but uses a prediction of the fluid load data from previous time steps, in order to obtain a better approximation of the equations that satisfy (6). The details of all methods are given in [18].

In summary, one can conclude that the validity of (6) was identified as an important design criterion for the fluid-structure coupling, and that it can be achieved in conjunction with standard solution methods available for fluids and structures.

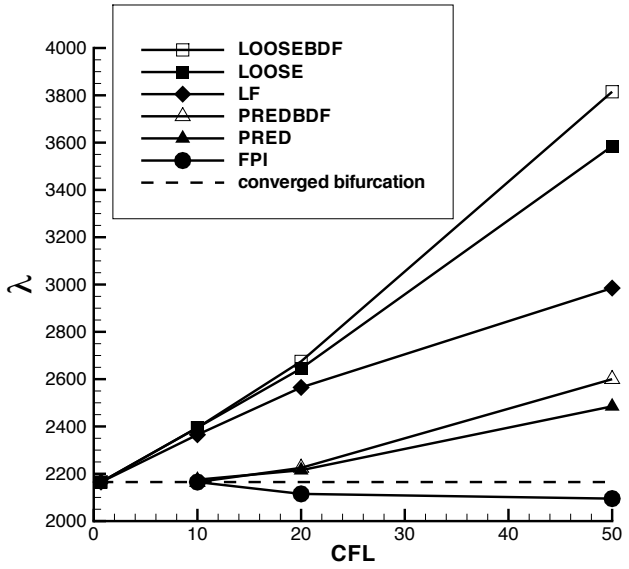

Fig. 4 Determination of bifurcation point with various time step sizes.

3 Multiscale Decompositions–Basic Ingredients

In this section we summarise the ingredients to successively decompose a sequence of cell averages given on a uniform fine grid (reference grid) into a sequence of coarse-scale averages and details. The details describe the update between two discretisations on successive resolution levels corresponding to a nested grid hierarchy. They reveal insight in the local regularity of the underlying function. In particular, whenever the details become negligible small in certain locations this gives rise to data compression. From the remaining *significant* details, an adaptive grid, i.e., a locally refined grid with hanging nodes can be determined. In principle, the concept can be applied to any hierarchy of nested grids, no matter whether these grids are structured or unstructured. However, here we will confine ourselves to structured grids and uniform dyadic refinement,s where on each refinement level the grids can

be determined by evaluating a grid mapping. This has been successfully realised in the multiscale library, see [19] for details, and incorporated into the Quadflow solver [6, 7].

3.1 Multiscale Analysis and Grid Adaptation

Grid mapping. The starting point is a smooth function $\mathbf{x} : R := [0,1]^d \to \Omega$, which maps the parameter domain R onto the computational domain Ω (which is typically one of several blocks in a composite grid). The Jacobian is assumed to be regular, i.e., $\det(\partial \mathbf{x}(\xi)/\partial \xi) \neq 0,\ \xi \in R$. In our applications we represent the grid function by B–splines, see [17]. This admits control of good local grid properties, e.g., orthogonality and smoothness of the grid, and a consistent boundary representation by a small number of control points depending on the configuration at hand.

Nested Grid Hierarchy. A nested grid hierarchy is defined from the grid mapping by means of a sequence of nested uniform partitions of the parameter domain. To this end, we introduce the sets of multi–indices $I_l := \prod_{i=1}^d \{0,\ldots,N_{l,i}-1\} \subset \mathbf{N}_0^d$, $l = 0,\ldots,L$, with $N_{l,i} = 2N_{l-1,i}$ initialised by some $N_{0,i}$. Here l represents the refinement level where the coarsest partition is indicated by 0 and the finest by L. The product denotes the Cartesian product, i.e., $\prod_{i=1}^d A_i := A_1 \times \cdots \times A_d$. Then the nested sequence of parameter partitions $\mathscr{R}_l := \{R_{l,\mathbf{k}}\}_{\mathbf{k}\in I_l},\ l = 0,\ldots,L$, is given by $R_{l,\mathbf{k}} := \prod_{i=1}^d [k_i h_{l,i}, (k_i+1)h_{l,i}]$, with $h_{l,i} := 1/N_{l,i} = h_{l-1,i}/2$, see Figure 5. Finally, a sequence of *nested grids* $\mathscr{G}_l := \{V_{l,k}\}_{\mathbf{k}\in I_l},\ l = 0,\ldots,L$, of the computational domain Ω is obtained by $V_{l,\mathbf{k}} := \mathbf{x}(R_{l,\mathbf{k}})$, see Figure 6 for an illustration. Each grid $\mathscr{G}_l$

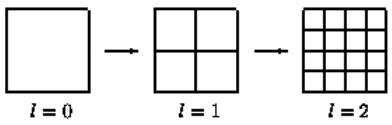

Fig. 5 Dyadic grid hierarchy in parameter space.

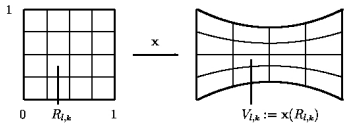

Fig. 6 Transformation from parameter to computational domain.

builds a partition of Ω, i.e., $\Omega = \bigcup_{\mathbf{k}\in I_l} V_{l,\mathbf{k}}$, and the cells of two neighbouring levels are nested, i.e., $V_{l,\mathbf{k}} = \bigcup_{\mathbf{r}\in\mathscr{M}_{l,\mathbf{k}}^0} V_{l+1,\mathbf{r}},\ \mathbf{k} \in I_l$. Because of the *dyadic* refinement, the refinement set is determined by $\mathscr{M}_{l,\mathbf{k}}^0 = \{2\mathbf{k}+\mathbf{i}\,;\,\mathbf{i}\in E := \{0,1\}^d\} \subset I_{l+1}$ of 2^d cells on level $l+1$ resulting from the subdivision of the cell $V_{l,\mathbf{k}}$.

Multiscale decomposition. For any scalar, integrable function $u \in L^1(\Omega,\mathbf{R})$ we define the average $u_{l,\mathbf{k}} := \langle u, \phi_{l,\mathbf{k}}\rangle_{L^2(\Omega)}$ as the inner product of u with the L^1–normalised box function $\phi_{l,\mathbf{k}}(\mathbf{x}) := |V_{l,\mathbf{k}}|^{-1}\chi_{V_{l,\mathbf{k}}}(\mathbf{x})$, $\mathbf{x}\in\Omega$, where $|V_{l,\mathbf{k}}| := \int_{V_{l,\mathbf{k}}} 1\,d\mathbf{x}$ denotes the cell volume and $\chi_{V_{l,\mathbf{k}}}$ the characteristic function on $V_{l,\mathbf{k}}$. With each grid $\mathscr{G}_l$ we can associate then the sequence of averages $\hat{\mathbf{u}}_l := \{\hat{u}_{l,\mathbf{k}}\}_{\mathbf{k}\in I_l}$. The nestedness

of the grids as well as the linearity of the integration functional imply the two–scale relation

$$\hat{u}_{l,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{M}^0_{l,\mathbf{k}}} m^{l,0}_{\mathbf{r},\mathbf{k}}\,\hat{u}_{l+1,\mathbf{r}}, \quad m^{l,0}_{\mathbf{r},\mathbf{k}} := \frac{|V_{l+1,\mathbf{r}}|}{|V_{l,\mathbf{k}}|}, \tag{7}$$

i.e., the coarse–grid average can be represented by a linear combination of the corresponding fine–grid averages. Consequently, starting on the finest level, the flow field represented by the corresponding array of cell averages can be successively downsampled by computing averages on coarser levels. Since information is, of course, lost by the averaging process, it is not possible to reverse (7). Therefore, we have to store the update between two successive refinement levels by additional coefficients, so-called *details* representing the fluctuation between two successive refinement levels. From the nestedness of the grid hierarchy we infer that the linear spaces $S_l := \mathrm{span}\{\phi_{l,\mathbf{k}}\ ;\ \mathbf{k}\in I_l\}$ are nested, i.e., $S_l \subset S_{l+1}$. Hence there exist complement spaces W_l such that $S_{l+1} = S_l \oplus W_l$. These are spanned by some basis, i.e., $W_l := \mathrm{span}\{\psi_{l,\mathbf{k},\mathbf{e}}\ ;\ \mathbf{k}\in I_l,\ \mathbf{e}\in E^* := E\backslash\{\mathbf{0}\}\}$ whose elements are oscilatory. For the construction of an appropriate *wavelet* basis we refer to [19]. In analogy to the cell averages, the details can be encoded by inner products $d_{l,\mathbf{k},\mathbf{e}} := \langle u, \psi_{l,\mathbf{k},\mathbf{e}}\rangle_{L^2(\Omega)}$ of the function u now with the *wavelet* $\psi_{l,\mathbf{k},\mathbf{e}}$. The rationale is that for us an "appropriate" basis means that it is a Riesz basis in L_2 say, and therefore has a companion Riesz basis consisting of elements $\tilde{\psi}_{l,\mathbf{k},\mathbf{e}}$ which form a *biorthogonal* system, i.e. $\langle \psi_{l,\mathbf{k},\mathbf{e}}, \tilde{\psi}_{l',\mathbf{k}',\mathbf{e}'}\rangle_{L_2(\Omega)} = \delta_{(l,\mathbf{k},\mathbf{e}),(l',\mathbf{k}',\mathbf{e}')}$. Therefore, the details are just the expansion coefficients of u with respect to the dual companion basis. Moreover, the choice of the companion basis determines, in particular, the order of vanishing moments of the $\psi_{l,\mathbf{k},\mathbf{e}}$ which, in turn, determine how small the details are when u is smooth on the respective wavelet support. In fact, a local Taylor argument readily shows that vanishing moments of order m imply that $|d_{l,\mathbf{k},\mathbf{e}}|$ is of the order 2^{-ml} when u has bounded derivatives of order m on the corresponding wavelet support.

Since the box functions and the wavelets are linearly independent, there exists a two–scale relation for the details, i.e.,

$$d_{l,\mathbf{k},\mathbf{e}} = \sum_{\mathbf{r}\in\mathscr{M}^{\mathbf{e}}_{l,\mathbf{k}}\subset I_{l+1}} m^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}}\,\hat{u}_{l+1,\mathbf{r}}. \tag{8}$$

On the other hand, we deduce from the change of basis the existence of an inverse two–scale relation

$$\hat{u}_{l+1,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{G}^0_{l,\mathbf{k}}\subset I_l} g^{l,0}_{\mathbf{r},\mathbf{k}}\,\hat{u}_{l,\mathbf{r}} + \sum_{\mathbf{e}\in E^*}\sum_{\mathbf{r}\in\mathscr{G}^{\mathbf{e}}_{l,\mathbf{k}}\subset I_l} g^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}}\,d_{l,\mathbf{r},\mathbf{e}}. \tag{9}$$

Note that the mask coefficients $m^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}}$ and $g^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}}$ in (7), (8) and (9) do not depend on the data but only on the bases and their underlying geometric information.

Grid Adaptation. The multiscale analysis outlined above allows us to generate a locally refined grid in the following way. First we perform the multiscale decomposition according to (7), (8), as illustrated in Figure 7. As indi-

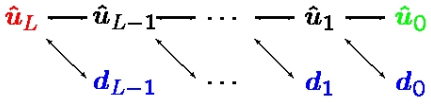

Fig. 7 Pyramid scheme of multiscale transformation.

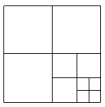

Fig. 8 Locally refined grid.

cated above, the details become small where the underlying function u is locally smooth. The basic idea is therefore to perform data compression on the vector of details using hard thresholding. This means we discard all detail coefficients $d_{l,\mathbf{k},\mathbf{e}}$ whose absolute values fall below a level-dependent threshold value $\varepsilon_l = 2^{(l-L)d}\varepsilon$ and only the *significant details* identified by the index set $\mathscr{D}_{L,\varepsilon} := \big\{(l,\mathbf{k})\,;\,|d_{l,\mathbf{k},\mathbf{e}}| > \varepsilon_l,\ \mathbf{k}\in I_l,\ \mathbf{e}\in E^*,\ l\in\{0,\ldots,L-1\}\big\}$ are retained. In order to account for the dynamics of a flow field, due to the time evolution, and to appropriately resolve all physical effects on the new time level, this set is to be inflated in such a way that the prediction set $\tilde{\mathscr{D}}_{L,\varepsilon} \supset \mathscr{D}_{L,\varepsilon}$ contains all significant details of the old and the new time level. Here one exploits roughly speaking the finite speed of information propagation in a (dominantly) hyperbolic problem. In a last step, we construct the locally refined grid, see Figure 8, and the corresponding cell averages. For this purpose, we proceed levelwise from coarse to fine, see Figure 7, and we check for all cells of a given level whether there exists a significant detail. Whenever we find one, we refine the respective cell, i.e., we replace the average of this cell by the averages of its children by locally applying the inverse multiscale transformation (9). The final grid is then determined by the index set $\tilde{\mathscr{G}}_{L,\varepsilon} \subset \bigcup_{l=0}^{L}\{l\}\times I_l$ such that $\bigcup_{(l,\mathbf{k})\in\tilde{\mathscr{G}}_{L,\varepsilon}} V_{l,\mathbf{k}} = \Omega$. Note that a locally refined grid always corresponds to a tree that represents the refinement history. The index sets resulting from the thresholding do not necessarily form a tree yet. Therefore we have to inflate the prediction set somewhat so as to form even a *graded* tree. This means that there is at most one hanging node at a cell edge which is not strictly necessary but greatly simplifies data management, see [19].

3.2 Algorithms

In order to benefit from the reduced complexity offered by the fact that the cardinality of the set of significant details and hence of the locally refined grid is typically much smaller than the corresponding uniform grid on level L, all transformations have to be performed locally. In particular, we are not allowed to operate on the full arrays corresponding to the uniformly refined grids, i.e., the summation in the transformations (7), (8) and (9) have to be restricted to those indices which correspond to non–vanishing entries of the mask coefficients. Introducing the mask matrices $\mathbf{M}_{l,\mathbf{e}} = (m^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}})_{\mathbf{r}\in I_{l+1},\mathbf{k}\in I_l}$ and $\mathbf{G}_{l,\mathbf{e}} = (g^{l,\mathbf{e}}_{\mathbf{k},\mathbf{r}})_{\mathbf{r}\in I_l,\mathbf{k}\in I_{l+1}}$, these two-scale relations may be rewritten in terms of matrix-vector products as $\mathbf{y}^T = \mathbf{x}^T\,\mathbf{A}$. Note that the mask matrices are sparse due to an appropriate choice of the wavelet basis. In order to confine

the summation in the matrix-vector product only to non-vanishing entries of the matrices it is helpful to introduce the following notion the *support* of matrix columns and rows

$$\mathscr{A}_k := \mathrm{supp}(\mathbf{A},k) := \{r\ ;\ a_{r,k} \neq 0\} = \text{ support of } k\text{th column of } \mathbf{A},$$

$$\mathscr{A}_k^* := \mathrm{supp}(\mathbf{A}^T,k) := \{r\ ;\ a_{k,r} \neq 0\} = \text{ support of } k\text{th row of } \mathbf{A}.$$

The support $\mathscr{A}_k$ of a column is comprised of all non–vanishing matrix elements that might yield a non–trivial contribution to the kth component of the matrix–vector product, i.e., y_k. Therefore $\mathscr{A}_k$ can be interpreted as the *domain of dependence* for y_k, i.e., the components x_r which contribute to y_k. The support $\mathscr{A}_k^*$ of a row contains all non–vanishing matrix entries of the kth row that might yield a non–trivial contribution to the vector $\mathbf{y}$ of the matrix–vector product. Therefore $\mathscr{A}_k^*$ can be interpreted as the *range of influence*, i.e., the components y_r which are influenced by the component x_k.

With this notation in mind, we now can formulate efficient algorithms for locally performing the decoding and encoding processes as they have been realised in the multiscale library:

Algorithm 1 *(Encoding) Proceed levelwise from $l = L-1$ downto 0:*

I. **Computation of cell averages on level** *l:*

1. *For each active cell on level $l+1$ determine its parent cell on level l:*
   $U_l^0 := \bigcup_{\mathbf{r}\in I_{l+1,\varepsilon}} \mathscr{M}_{l,\mathbf{r}}^{*,0}$ *where* $I_{l+1,\varepsilon} := \{\mathbf{k} \in I_{l+1} : (l+1,\mathbf{k}) \in \tilde{\mathscr{G}}_{L,\varepsilon}\}$
2. *Compute cell averages for parents on level l:*
   $\hat{u}_{l,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{M}_{l,\mathbf{k}}^0} m_{\mathbf{r},\mathbf{k}}^{l,0}\, \hat{u}_{l+1,\mathbf{r}},\ \mathbf{k} \in U_l^0$

II. **Computation of details on level** *l:*

1. *For each active cell on level $l+1$ determine all cells on level l influencing their corresponding details:*
   $U_l^{\mathbf{e}} := \bigcup_{\mathbf{r}\in I_{l+1,\varepsilon}} \mathscr{M}_{l,\mathbf{r}}^{*,\mathbf{e}},\ \mathbf{e} \in E^*$
2. *For each detail on level l determine the cell averages on level $l+1$ that are needed to compute the detail:*
   $P_{l+1} := \bigcup_{\mathbf{e}\in E^*} \bigcup_{\mathbf{k}\in U_l^{\mathbf{e}}} \mathscr{M}_{l,\mathbf{k}}^{\mathbf{e}} \backslash I_{l+1,\varepsilon}$
3. *Compute a prediction value for the cell averages on level $l+1$ not available in adaptive grid:*
   $\hat{u}_{l+1,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{G}_{l,\mathbf{k}}^0} g_{\mathbf{r},\mathbf{k}}^{l,0}\, \hat{u}_{l,\mathbf{r}},\ \mathbf{k} \in P_{l+1}$
4. *Compute the details on level l:*
   $d_{l,\mathbf{k},\mathbf{e}} := \sum_{\mathbf{r}\in\mathscr{M}_{l,k}^{\mathbf{e}}} m_{\mathbf{r},\mathbf{k}}^{l,\mathbf{e}} \hat{u}_{l+1,\mathbf{r}},\ \mathbf{k} \in U_l^{\mathbf{e}},\ \mathbf{e} \in E^*$

Algorithm 2 *(Decoding) Proceed levelwise from $l = 0$ to $L-1$:*

I. **Computation of cell averages on level** *$l+1$:*

1. *Determine all cells on level $l+1$ that are influenced by a detail on level l:*
   $I_{l+1}^+ := \bigcup_{\mathbf{e}\in E^*} \bigcup_{\mathbf{l}\in J_{l,\varepsilon}} \mathscr{G}_{l,\mathbf{l}}^{*,\mathbf{e}}$ *where* $J_{l,\varepsilon} := \{\mathbf{k} \in I_l : (l,\mathbf{k}) \in \tilde{\mathscr{D}}_{L,\varepsilon}\}$

2. *Compute cell averages for cells on level* $l+1$*:*
$\hat{u}_{l+1,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{G}^0_{l,\mathbf{k}}} g^{l,0}_{\mathbf{r},\mathbf{k}}\,\hat{u}_{l,\mathbf{r}} + \sum_{\mathbf{e}\in E^*}\sum_{\mathbf{r}\in\mathscr{G}^{\mathbf{e}}_{l,\mathbf{k}}} g^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}}\,d_{l,\mathbf{e},\mathbf{r}}, \quad \mathbf{k}\in I^+_{l+1}$

II. **Remove refined cells on level** *l:*

1. *For each cell on level* $l+1$ *determine its parent cell on level* l*:*
$I^-_l := \bigcup_{\mathbf{k}\in I^+_{l+1}} \mathscr{M}^{*,0}_{l,\mathbf{k}}$
2. *Remove the parent cells on level* l *from the adaptive grid:*
delete $\hat{u}_{l,\mathbf{k}}$, $\mathbf{k}\in I^-_l$*, i.e.,* $I_{l,\varepsilon} := I^+_l / I^-_l$ *(Note:* $I^+_0 = I_0$*)*

3.3 Data Structures

In order to fully benefit from the above principal complexity reduction, we need appropriate supporting data structures. These have to be designed in such a way that the computational complexity (storage and CPU time) remains proportional to the cardinality of the adaptive grid and hence of the significant details, respectively. For this purpose, the C++-template class library `igpm_t_lib` [20, 24] has been developed. This library provides data structures that are tailored to the requirements set by Algorithms 1 and 2, from which the fundamental design criteria are deduced, namely, (i) *dynamic memory operations* and (ii) *fast data access* with respect to inserting, deleting and finding elements.

Due to refinement and coarsening operations in the algorithm, memory operations are frequently performed and therefore should be very fast. This can be realised more efficiently by allocating a sufficiently large memory block and by managing the algorithm's memory requirements with a specific data structure. In addition, since the overall memory demand can only be estimated, the data structure should provide dynamic extension of the memory.

In order to facilitate an efficient memory management and a fast data access we use the well-known concept of *hash maps*, cf. [8], that is composed of two parts, namely, a vector of pointers, a so–called *hash table*, and a memory heap, see Figure 9. The hash table is connected to a *hash function* $f : \mathscr{U} \to \mathscr{T}$, which maps a *key*, here $(l,\mathbf{k})$, to the hash table of length $\#\mathscr{T}$, i.e., a number between 0 and $\#\mathscr{T}-1$. Here the set $\mathscr{U}$ can be identified with all possible cells in the nested grid hierarchy (universe of keys), i.e., $\mathscr{U} = \{(l,\mathbf{k}) : \mathbf{k}\in I_l,\ l = 0,\dots,L\}$, and $\mathscr{T}$ corresponds to the keys of the dynamically changing adaptive grid, i.e., $(l,\mathbf{k})\in\tilde{\mathscr{G}}_{L,\varepsilon}$.

The set of all possible keys is much larger than the length of the hash table, i.e., $\#\mathscr{T} \ll \#\mathscr{U}$. Hence, the hash function cannot be injective. This leads to collisions in the hash table, i.e., different keys might be mapped to the same position by the hash function. As collision resolution we choose chaining: the corresponding values of these keys are linked to the list that starts at position $f(key)$. Each element in the hash table is a pointer to a linked list whose elements are stored in the heap. Here each element of the list can be a complex data structure itself. It contains the key

and usually additional data, the so–called *value*. In general, the value consists of the data corresponding to a cell.

The performance of the hash map crucially depends on the number of collisions. In order to optimise the number of collisions, the length of the hash table $\#\mathscr{T}$ and the number of collisions $\#\{key \in \mathscr{U} : f(\{key\}) = c\}$ have to be well-balanced. Several strategies have been developed for the design of a hash function, see [23, 8]. For our purpose, choosing the modulo function and appropriate table lengths turned out to be sufficient, see [19].

Since the local multiscale transformations are performed level by level, see Algorithms 1 and 2, the hash map has to maintain the level information. For this purpose, the standard hash map is extended by a vector of length L. The idea is to have a linked list of all cells on level l: the lth component of the vector contains a pointer that points to the first element of level l put into the memory heap. Additionally, the value has to be internally extended by a pointer that points to the next element of level l. This is indicated in Figure 10. Then we can access all elements of level l by traversing the resulting singly linked list.

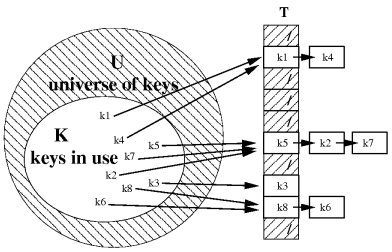

Fig. 9 Hashing (Courtesy of [25]).

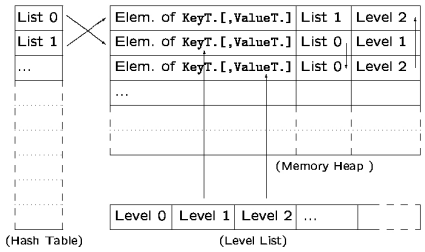

Fig. 10 Linked hash map.

4 Parallelisation

In the following we will present how we have parallelised the multiscale library by which we perform local grid adaptation in one block using multiscale techniques. For this purpose, we first outline the strategy of load-balancing using space-filling curves, see Section 4.1. In a second step, see Section 4.2, we explain how to perform the multiscale transformation in parallel. Here the crucial point is the handling of the cells on the partition boundary and interprocessor communication. Note that the multiscale-based grid adaptation consists of additional steps such as thresholding, prediction, grading and decoding. The parallelisation of these steps is in complete analogy and therefore not detailed here.

4.1 Load-Balancing via Space-Filling Curves

Regarding parallelization, a first essential issue is the mesh partitioning or the load-balancing problem. This is fairly straightforward for uniform grids. But having to deal with locally refined grids where not all cells on all levels of refinement are active, complicates matters significantly. A natural representation of a multilevel partition of a mesh is a global enumeration of the active cells. We need a method to do this at runtime, as the adaptive mesh is also created at runtime using the multi-scale representation techniques sketched above. Such an enumeration can be based on space-filling curves (SFC) by mapping a higher-dimensional domain to a one-dimensional curve. Specifically, the unit square or the unit cube is mapped to the unit interval. Using space-filling curves, each of the cells of the adaptive grid has a corresponding unique number on the curve. So, instead of assigning portions of the geometric domain to different processors, we only have to split the interval of numbers on the curve into parts that have roughly equal length. Each of these parts is mapped to a different processor, so that we obtain a well-balanced data partition. However, concerning the second major issue, we also have to pay the cost of inter-processor communications, because neighbours within a grid block may belong to different processors. We shall indicate below that the concept of space-filling-curves offers favorable features in this regard as well.

Space-Filling Curve. Space-filling curves have been originally created for purely mathematical purposes , cf. [22]. Nowadays, these curves have found several applications, one of them being a good load-balancing for numerical simulations on parallel computer architectures. They can be used for data partitioning and, due to self-similarity features, multilevel partitions can also be constructed.

In the mathematical definition, a space-filling curve is a surjective, continuous mapping of the unit interval $[0,1]$ to a compact d-dimensional domain Ω with positive measure. In our context, we restrict our attention to Ω being the unit square or the unit cube. In fact, as our grids have finite resolution, the iterates — so-called discrete space-filling curves — are applied, instead of the continuous space-filling curve. The construction of these curves is extremely inexpensive, as the SFC index for any cell in the grid can be computed using only local information, making it suitable for parallel computations.

Hilbert Space-Filling Curve. One of the oldest space-filling curves, the Hilbert curve, can be defined geometrically, cf. [25]. The mapping is defined by the recursive subdivision of the interval I and the square Q. In 3D, the Hilbert curve is based on a subdivision into eight octants. The construction begins with a generator template, which defines the order in which the quadrants are visited. Then the template (identical, mirrored or rotated) is applied to each quadrant and, by connecting the loose ends of the curve, the next iterate of the space-filling curve is obtained. Actually, the mapping between the cells of the adaptive grid and the space-filling curve is realised using the finest iterate of the curve, which is constructed by recursively applying the template to the subquadrants (2D) and suboctants (3D) until the number of refinement levels is reached. Figures 11 and 12 show the first iterates of a 2D and 3D Hilbert SFC, respectively. A detailed discussion on the construction of

the Hilbert space-filling curve is beyond the scope of this paper. For corresponding detailed expositions we refer the reader to [22, 25]. Here, we only summarise the procedure of the Hilbert curve construction and focus our attention on how the curve can contribute to an efficient parallelisation of the multiscale-based grid adaptation scheme.

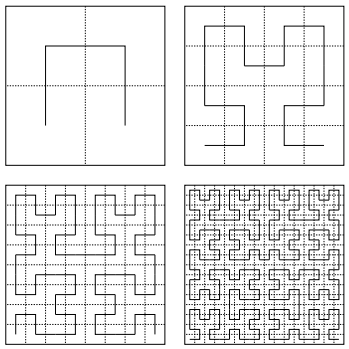

Fig. 11 First 4 iterates of 2D Hilbert SFC.

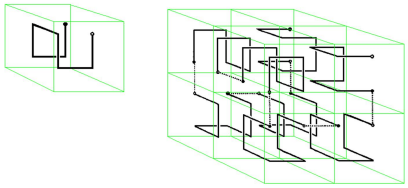

Fig. 12 1st and 2nd iterate of 3D Hilbert SFC (Courtesy of Gilbert [11]).

Encoding the Hilbert SFC ordering. By inverting the mapping induced by a discrete space-filling curve, multi-dimensional data can be mapped to a one-dimensional interval. The basic idea is to map each cell of our adaptive grid to a point on the space-filling curve, so that we obtain a global enumeration of the grid cells. In the data structures, we use the key comprised of the cell's refinement level and the cell's multidimensional index on that level $(l, \mathbf{k})$ to identify each cell. Since each cell of the adaptive grid is uniquely identified by this key, the aim is to use it in order to determine for each cell a corresponding number on the space-filling curve. Also, due to locality properties of the curves, each visited cell is directly connected to two face-neighbouring cells which remain face neighbours in the one-dimensional space spanned by the curve. In this way, the cell's children are sorted according to the SFC numbers and they will be nearest-neighbours on a contiguous segment of the SFC. Since we are dealing with multilevel adaptive rectangular grids, we restrict ourselves to recursively defined, self-similar SFC based on rectangular recursive decomposition of the domain.

Encoding and decoding the Hilbert SFC order requires only local information, i.e., a cell's 1D index can be constructed using only that cell's integer coordinates in the d-dimensional space and the maximum number of refinement levels L that exist in the mesh. In a 2D space, consider a $2^L \times 2^L$ square $(0 \leq X \leq 2^{L-1},\ 0 \leq Y \leq 2^{L-1})$ in a Cartesian coordinate system. Note that the variable L used for determining the Hilbert SFC order might be larger than the one introduced in Section 3.1 for the number of refinement levels, since $N_{0,i} > 1$ in general and for the space-filling curve construction we should have a coarsest mesh with $N_{0,i} = 1$. Any point can be expressed by its integer coordinates, (X, Y), where X, Y are two sequences of L-bit binary numbers, as follows:

$$X = x_1 x_2 \dots x_r \dots x_L, \quad Y = y_1 y_2 \dots y_r \dots y_L.$$

Each sequence of two interleaved bits $\{x_l, y_l\}_{l=1,\dots,L}$ determines on each level l one of the four quadrants the cell belongs to, recursively. Thus by simply inspecting the cell's integer coordinates and using a finite state machine, the cell's location on a curve can easily be computed, cf. [25]. In the 3D case we proceed similarly.

An important aspect that should be mentioned is that the construction of the space-filling curve on an adaptive mesh ensures that a parent cell $(l, \mathbf{k})$ has exactly the same number on the SFC as *one* of its children $(l+1, 2\mathbf{k}+\mathbf{e})$, $\mathbf{e} \in \{0,1\}^d$, cf. [25]. This leads to the minimisation of the interprocessor communication in the case of a parallel MST using the Hilbert space-filling curve as partitioning scheme, because in most of the cases the parent cell should be computed on the same processor as its children.

Load-Balancing. After computing the SFC indices for all the cells in the mesh, these indices are taken as sort keys and the mesh may be ordered along the curve using standard sorting routines, such as introsort in our case. Having all the cells sorted along the curve, the partition can be easily determined, just by choosing the number of cells that each processor should get. So the mesh cells are distributed to the different processors according to their index on the curve. Since the position of each cell on the curve can be computed very inexpensively at any time in the computation, there is no need to store all the keys. Instead, it is sufficient to store the separators between the elements of the partition of the interval, i.e., the first index on a processor, in order to determine for any cell's multidimensional index the corresponding processor number.

There are two options for handling the data partitioning and the load-balancing problem in the beginning of the computation, namely, (i) master-based partitioning and (ii) symmetric multiprocessing. In case of master-based partitioning, as its name says, the entire adaptive mesh is initialised on a master processor, according to the input file. Once the grid is initialised, the same master processor is also responsible for the entire partitioning procedure already described: the mapping of the cells to the SFC, the sorting of the keys, the load-balancing and separators' determination. After having performed these steps once, the cells can be distributed to the corresponding processors. This approach is straight forward if the starting point is a running serial algorithm, as no data transfer and no barrier points are needed before the distribution of the data to processors actually begins. On the other hand, this implies that there is only one processor active during all the initialisation and sorting of the SFC procedures, while the others are idle, waiting to receive the data from the master for initialising their own data structures.

The second possibility is symmetric multiprocessing: this avoids the use of a master processor, i.e., all processors should work in parallel, executing the same code and initialising only their corresponding part of the grid. For this, a set of initial separators on the space-filling curve has to be assumed, without knowing in advance anything about the structure of the adaptive grid. So the worst case has to be taken into account, when the grid would be uniformly refined, which is equivalent to the fact that, for each number on the discrete space-filling curve, there exists an

active cell in the grid. In the case of a fully refined grid, the number of cells in the grid ($2^L \times 2^L$ and $2^L \times 2^L \times 2^L$, in 2D and 3D, respectively) corresponds to the last number on the SFC and the guess of the initial separators is straight forward. The main drawback of this second approach is that this initial guess might and is very likely to be far from the optimal choice, so the possibility of not having a remarkable performance improvement in the initialisation part is very high, also due to the interprocessor communication costs that arise. A rebalancing of the initial data is then required in order to obtain a well-balanced distribution of data among processors and a new set of separators is computed for the new partition.

Applying either of these two strategies leads to a well-balanced data distribution as shown in Figure 13.

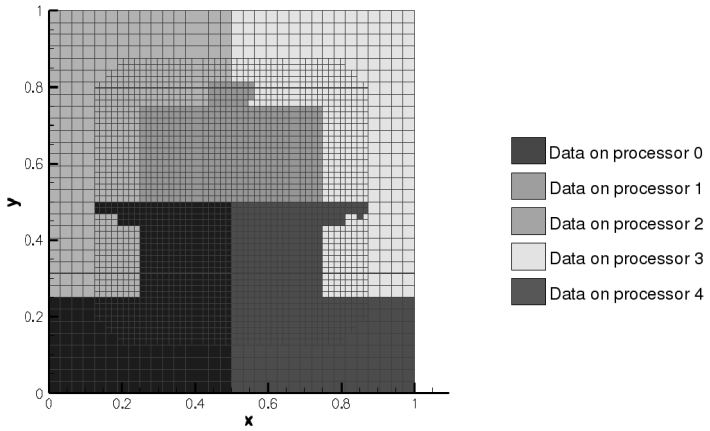

Fig. 13 Well-balanced distribution of a locally refined grid to 5 processors.

Parallel rebalancing. A reordering of the cells along the curve is also needed whenever the load-balancing is significantly spoiled due to the adaptivity of the grid. When this occurs, a new set of separators — that determine a new well balanced partition — has to be computed. There is no need to gather all cells on a master processor for the reordering, since all the cells on a processor p have smaller numbers on the SFC than the cells on processor $p+1$ for all $p = 0,\ldots,n_{\text{proc}}-2$. The parallel rebalancing is described in Algorithm 3.

Algorithm 3 *(Parallel Rebalancing)*

1. *Sort local cells along the SFC according to the old list of separators* $sep_{\text{old}}[i]$ *for* $i = 0,\ldots,n_{\text{proc}}-1$.
2. *Compute the total workload from all processors:*
   $\text{total\_workload} = \sum_{p=0}^{n_{\text{proc}}-1} \#\text{cells}(p)$

3. *Compute the positions of the new separators on the SFC:*

   a) $\text{new\_positions}[0] = 0$
   b) For $i = 1, \ldots, n_{\text{proc}} - 1$ *do*
   $\text{new\_positions}[i] = \text{new\_positions}[i-1] + i \cdot (\text{total\_workload}/n_{\text{proc}})$

4. *For* $pos = 1, \ldots, n_{\text{proc}} - 1$ *do*

   If $\text{new\_positions}[pos]$ *belongs to the local processor, then determine the new separator*
   $\text{sep}_{\text{new}}[pos]$ *at position* $\text{new\_positions}[pos]$.

5. *Distribute new separators to all processors.*
6. *Redistribute data according to the new separators.*

4.2 Parallel Grid Adaptation and Data Transfer

Once load-balancing is achieved, each processor should perform the grid adaptation, see Section 3.1, on the local data. Special attention must be paid to the cells located at the processor's boundary, i.e., the cells that have at least one neighbour belonging to another processor. Since these are the only ones that make the difference between serial and parallel algorithms and because they are similarly handled in all the steps of the grid adaptation, we shall address below only the parallelisation of the encoding step in some detail. More specifically, we shall mainly discuss the special treatment of the boundary cells. As described in Section 3.2, the encoding step consists of computing the cell averages on level l starting from data on level $l+1$ and the computation of the details on level l. Since the approach to parallelising the coarsening is different from the one for computation of details, they will be discussed separately.

Parallel coarsening. To compute the parent's cell average $\hat{u}_{l,\mathbf{k}}$ on the processor indicated by the parent's position on the SFC and the separators between the elements of the partition, all its children $\hat{u}_{l+1,\mathbf{r}}$, $\mathbf{r} \in \mathscr{M}^0_{l,\mathbf{k}}$, should already be available on the same processor. Due to the locality properties of the SFC and the compactness of each element of the partition, i.e., the ratio of an element's volume and its surface is large, ensured by its construction, the children are nearest neighbours in the one-dimensional space. This entails that only a few cells at the boundaries between partition elements have to be transferred between processors before computing the cell averages on level l, see Figure 14. When running through the active data on level $l+1$ for constructing the set of parent cells that need to be computed, the processor the parent belongs is also determined and so a buffer is set up, containing the children cells that need to be transferred to a neighbour processor. The buffer is transferred to the corresponding processor before the computation of the averages on level l actually begins. Once the cell averages of all cells' parents have been computed, the ghost cells transferred from other processors can be deleted from the local hash map.

Algorithm 4 *(Parallel Coarsening) Proceed levelwise from $l = L-1$ downto 0: (cf. Algorithm 1, Step I)*

I. **Computation of cell averages on level** *l:*

1. *For each active cell $(l+1,\mathbf{r})$, $\mathbf{r} \in I_{l+1,\varepsilon}$, determine*
   a) the parent cell on level l;
   b) the processor p to which the parent belongs to.
   If p is the current processor then $U_l^0 = U_l^0 \cup \mathscr{M}_{l,\mathbf{r}}^{*,0}$;
   else transfer cell $(l+1,\mathbf{r})$ to processor p and there add it to the local hash map.
2. *Compute cell averages for parents on level l:*
   $\hat{u}_{l,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{M}_{l,\mathbf{k}}^0} m_{\mathbf{r},\mathbf{k}}^{l,0}\,\hat{u}_{l+1,\mathbf{r}},\ \mathbf{k} \in U_l^0$
3. *Delete the cells received from other processors.*

Parallel details computation. In the case of the computation of details, more data from the neighbour processors have to be transferred, see Figure 15. For the parallel coarsening, a single step data transfer is made, since each processor can determine the cells on level $l+1$ that are necessary for the neighbours to compute the averages on level l by itself. In case of the multiscale decomposition for computing the details a two step transfer has to be performed. In a first step, each processor has to send a request to the others for the cells that influence the details computation of the local cells, so we obtain a first pair of MPI_Isend and MPI_Recv calls. A new pair of such calls is needed to fulfil the requests, when actually all the data located at the geometrical boundary of the partitions has to be transferred to the neighbouring processors.

Algorithm 5 *(Parallel Details Computation) Proceed levelwise from $l = L-1$ downto 0: (cf. Algorithm 1, Step II)*

I. **Computation of details on level** *l:*

0. *On each processor initialise the index sets $U_{l,p}^{\mathbf{e}} = \emptyset$, $p = 0,\ldots,n_{proc}-1$*
1. *For each active cells on level $(l+1,\mathbf{r})$, $\mathbf{r} \in I_{l+1,\varepsilon}$ do*
   a) determine all cells on level l influencing their corresponding details:
   $(l,\mathbf{k}) \in \mathscr{M}_{l,\mathbf{r}}^{*,\mathbf{e}}$, $\mathbf{e} \in E^*$;
   b) for each $(l,\mathbf{k}) \in \mathscr{M}_{l,\mathbf{r}}^{*,\mathbf{e}}$ determine the processor p where the details of cell $(l,\mathbf{k})$ should be computed:
   if $p = p_{loc}$ (current processor) then $U_{l,p}^{\mathbf{e}} := U_{l,p}^{\mathbf{e}} \cup \{\mathbf{k}\}$;
   else transfer index $(l,\mathbf{k})$ to processor p and there add it to the index set $U_{l,p}^{\mathbf{e}} := U_{l,p}^{\mathbf{e}} \cup \{\mathbf{k}\}$ and transfer cell $(l+1,\mathbf{r})$ to processor p and there add it to the local hash map.
2. *For each detail on level l determine the cell averages on level $l+1$ that are needed to compute the detail:*
   $P_{l+1} := \bigcup_{\mathbf{e}\in E^*} \bigcup_{\mathbf{k}\in U_l^{\mathbf{e}}} \mathscr{M}_{l,\mathbf{k}}^{\mathbf{e}} \setminus I_{l+1,\varepsilon}$

3. *For all indices* $(l+1,\mathbf{r}) \in P_{l+1}$ *that belong to other processors* $p \neq p_{loc}$ *(current processor),* $p = 0,\ldots,n_{proc}-1$ *do*
   a) send requests to processor p to transfer the unavailable data;
   b) receive needed data from processor p.
4. *Accept requests from other processors and send back the data to the other processors requested from the current processor.*
5. *Compute a prediction value for the cell averages on level* $l+1$ *not available in the adaptive grid:*
   $\hat{u}_{l+1,\mathbf{k}} = \sum_{\mathbf{r}\in\mathscr{G}^0_{l,\mathbf{k}}} g^{l,0}_{\mathbf{r},\mathbf{k}}\,\hat{u}_{l,\mathbf{r}},\ \mathbf{k} \in P_{l+1}$
6. *Compute the details on level l:*
   $d_{l,\mathbf{k},\mathbf{e}} := \sum_{\mathbf{r}\in\mathscr{M}^{\mathbf{e}}_{l,k}} m^{l,\mathbf{e}}_{\mathbf{r},\mathbf{k}}\hat{u}_{l+1,\mathbf{r}},\ \mathbf{k} \in U^{\mathbf{e}}_l,\ \mathbf{e} \in E^*$
7. *Delete data received from other processors from the local hash map.*

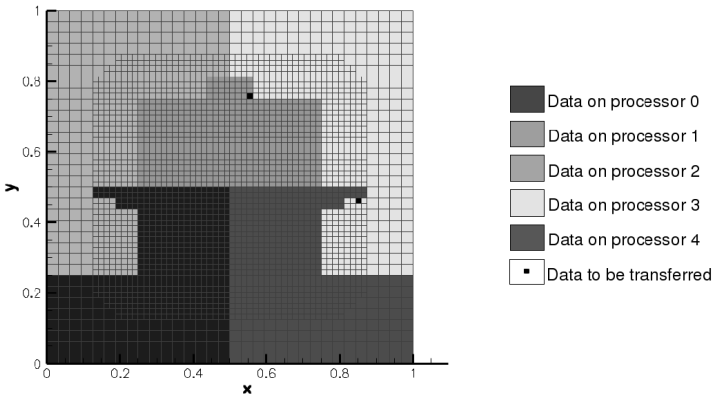

Fig. 14 Cells to be transferred for parallel coarsening.

5 Embedding of Parallel Multiscale Library into the Quadflow Solver

The finite volume solver Quadflow [6, 7] has been designed to handle (i) unstructured grids composed of polygonal(2D)/polyhedral(3D) elements [5] and (ii) block-structured grids where in each block the grid is determined by local evaluation of B-Spline mappings [17]. Therefore the solver can handle grids provided by an external grid generator. However, grid adaptation is only available for block-structured grids, where in each block the grid is locally refined using the concept of multiscale-based grid adaptation [19].

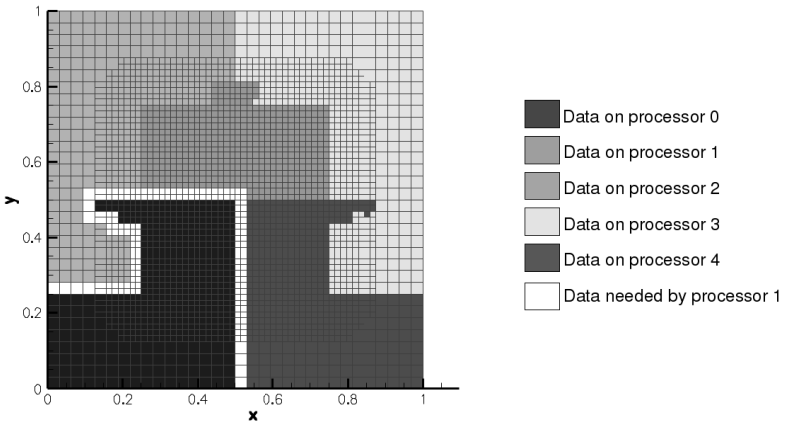

Fig. 15 Cells to be transferred for parallel details computations.

Note that the flow solver and the multiscale-based grid adaptation pose totally different algorithmic requirements: on one hand, there is a finite volume scheme working on arbitrary, unstructured discretisations. On the other hand, there is the multiscale algorithm assuming the existence of hierarchies of structured meshes. The flow solver module is face-centred, since the central issue is the computation of the fluxes at the cell faces, while the adaptation module is cell-centred, analysing and manipulating cell averages. Moreover, the data structures used in the two parts are also different: while for the adaptation part a special implementation of hash maps is used, see Section 3.3, for the flow solver module the FORTRAN style data structures remained optimal. The link between these two modules is done by a data conversion algorithm, which organises all the data communication — the transfer of the conservative variables, volumes, cell centres, the registration of the knots, the construction of the faces and determination of their neighbouring cells and nodes — between the two modules in a connectivity list.

In order to embed the parallelised version of the multiscale library into Quadflow, the transfer had to be adjusted: besides the special treatment required by some cells located at the physical boundary or at the far field boundary of the domain even in the serial case, cf. [17], in parallel special attention is needed also by the cells at the partition's boundary. Since the adaptive mesh is determined at runtime, as well as the partition, and since the adaptive mesh and implicitly the shape of the elements of the partition could change at any time step, there is no way of knowing in advance which cells are located on a processor's boundary. This entails that the partition boundary on each processor should be reconstructed each time the connectivity list is built, in order to transfer the cells on the boundary to the neighbour processors, see Figure 16. In this way, the boundary faces of the partition and the flux at these faces can be properly computed on each processor.

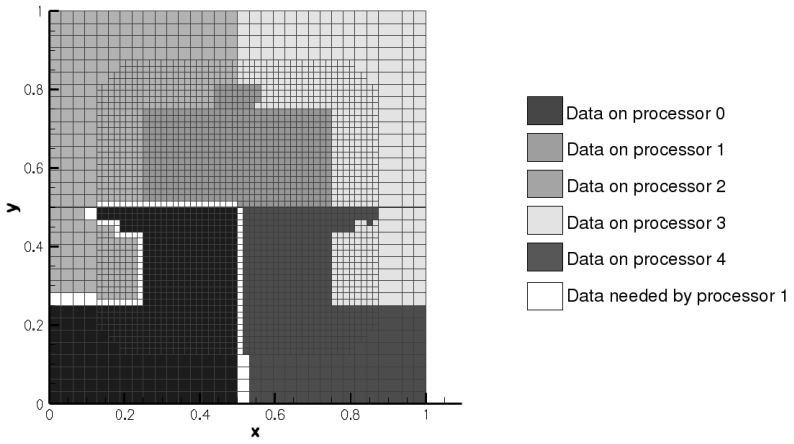

Fig. 16 Parallel Transfer and Conversion.

6 Numerical Results

First of all, we investigate in Section 6.1 the performance of the parallelised multiscale library. For this purpose, we focus on the multiscale transformation (MST), i.e., the encoding of the data. Note that a complete cycle of grid adaptation also includes thresholding, prediction, grading and decoding. In Section 6.2 we then present numerical simulations of the behaviour of a Lamb-Oseen vortex using the parallelised Quadflow solver.

6.1 Performance Study for Multiscale Transformation

The performance of the multiscale based grid adaptation has been investigated by means of a data set corresponding to a locally refined grid that consists of 437236 cells. The underlying grid hierarchy is determined by $L = 10$ refinement levels and a coarse grid discretisation of 8×8 cells. For this configuration, we performed the adaptation process using an increasing number of processors.

The experiments were performed on a Sun Fire X4600 system, with 2 AMD Opteron 885 nodes having 8 sockets per node (a total of 16 processors), 32 GB memory and a high speed low latency network (InfiniBand) for parallel MPI and hybrid parallelised programs.

Table 1 shows the results of these experiments with respect to the different number of processors mentioned in the first column. The second column contains the values for the initial workload, i.e. the number of cells on each processor. Columns 3 and 4 show the times measured when running one MST step and the time spent

No. of procs.	Initial workload	MST time [CPU s]	Transfer time [CPU s]	No. of cells sent	No. of cells received
1	437236	10.21580	0	0	0
2	218618	5.55433	0.066706	2020	2020
3	145746	3.45852	0.170083	5985	5975
4	109309	2.81636	0.125800	2048	2048
5	87448	2.13640	0.127011	10826	10706
6	72876	1.78998	0.126228	7240	7190
7	62464	1.55737	0.133888	9097	9078
8	54658	1.47147	0.103048	6899	6954
9	48588	1.33229	0.105228	7112	7068
10	43729	1.23401	0.135434	8856	8791

Table 1 Performance study for parallel multiscale transformation.

in the MPI_Isend and MPI_Recv routines, respectively. The number of cells sent by one processor during this MST step is shown in the fifth column, while the sixth column contains the number of cells received by the same processor.

After the partitioning is done, each processor is getting a number of cells equal to the total number of cells in the adaptive grid over the number of processors. Note that the processor with the highest rank also takes the few cells that remain if the total number of entries cannot be divided by the number of processors. Better performance and good scaling may be observed in column 3 as the number of processors increases (see also Figure 17). When it comes to the transfer of cells between neighbour processors, the number of entries sent by one processor is different than the one received on most processor configurations chosen due to the adaptive grid: one processor might have more finer cells sitting on the partition boundary than its neighbours have on the other side of the boundary, which gives the difference in the number of ghost cells needed to be transferred from one processor to the other. The number of processors chosen also influences the shape of the elements of the partition created at runtime. Thus, having a symmetric adaptive grid on a single processor and choosing to run a parallel computation on 2 or 4 processors might lead to a symmetric partitioning of the grid on all refinement levels. This implies the minimisation of the interprocessor communication. The consequences of this symmetry can be observed in Table 1, where the minimum number of cells to be transferred is achieved on 2 processors. At small difference, the computation on 4 processors also gives few entries to be transferred in comparison with the rest of the configurations tested. The fact that on 2 and 4 processors a symmetric partitioning is obtained is also emphasised when inspecting the number of cells sent and received across the partition boundary: here the number of cells sent to the neighbours is equal to the number of cells received. The opposite is observed on a configuration of 5 processors, where the number of cells on the partition's boundary reaches the maximum value and implicitly increases the interprocessor communication. A partitioning to 5 processors is shown in Figure 13, where the asymmetry of the partitions is obvious.

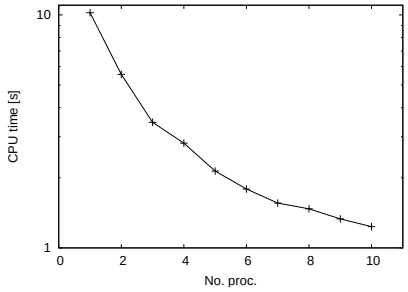

Fig. 17 CPU time for performing MST.

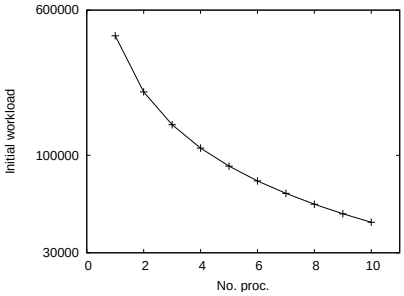

Fig. 18 Amount of data on each processor.

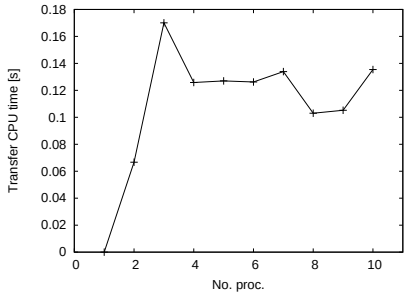

Fig. 19 CPU time for performing data transfer.

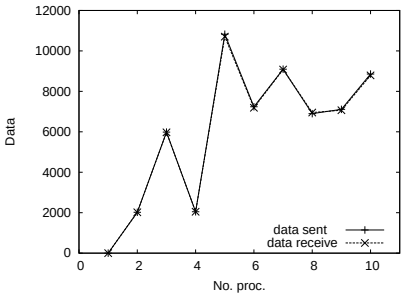

Fig. 20 Amount of data to be sent and received between processors.

Speedup. According to Amdahl's law [1] for a given problem the maximal speedup of a computation on p processors is bounded by $s_{\max} \leq \frac{p}{1+f(p-1)}$, where f is the fraction of the runtime of the code that is not parallelised.

We compared the scaling of our experiment with Amdahl's law and therefore assumed that the maximum speedup was measured. Using a nonlinear least squares fit, we were able to estimate the fraction of the code that is not parallelised and obtained $f = 0.0203 \pm 0.0017$, cf. Figure 21.

Having a fraction of 2% of the program not parallelised, according to Amdahl's law, the maximum speedup that could be reached for this fixed configuration would be 50. However, Amdahl's law doesn't take into account that the fraction of the serial parts can be reduced by scaling the problem to the number of processors. So, for a fixed configuration, infinite speedup cannot be achieved, but when the problem size grows, also speedup could be expected on an increasing number of processors.

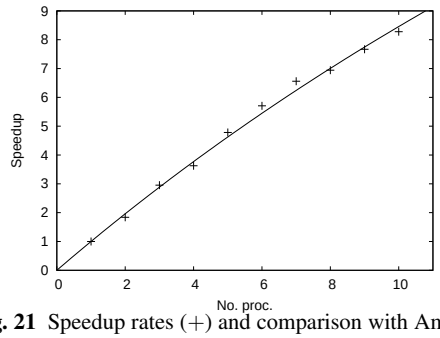

Fig. 21 Speedup rates (+) and comparison with Amdahl's law (solid line).

6.2 Application

The system of vortices in the wake of airplanes continues to exist for a long period of time in reality. It is possible to detect wake vortices as far as some 100 wing spans behind the airplane, which are a hazard to following airplanes. In the frame of the collaborative research centre SFB 401, the research aimed to induce instabilities into the system of vortices to accelerate their alleviation. The effects of different measures taken in order to to destabilise the vortices have been examined in a water tunnel. A model of a wing was mounted in a water tunnel and the velocity components in the area behind the wing were measured using particle image velocimetry. It was possible to conduct measurements over a length of 4 wing spans. The experimental analysis of a system of vortices far behind the wing poses great difficulties due to the size of the measuring system. Numerical simulations are not subject to such severe constraints and therefore Quadflow is used to examine the behaviour of vortices far behind the wing. To minimise the computational effort, the grid adaptation adjusts the refinement of the grid with the goal to resolve all important flow phenomena, while using as few cells as possible.

In the present study instationary, quasi-incompressible, inviscid fluid flow described by the Euler equations for compressible low Mach number flow is considered. A first assessment is presented to validate the ability of Quadflow to simulate the behaviour of the wake of an airplane. A velocity field based on the experimental data from the water tunnel measurements is prescribed as boundary condition in the inflow plane. The circumferential part of the velocity distribution $v_\Theta(r)$ is described by a Lamb-Oseen vortex according to

$$v_\Theta(r) = \frac{\Gamma}{2\pi r}\left(1 - e^{-\left(r/d_0\right)^2}\right). \tag{10}$$

The axial velocity component in the inflow direction is set to the constant inflow velocity of the water tunnel. The two parameters of the Lamb-Oseen vortex, circulation Γ and core radius d_0 are chosen in such a way that the model fits the measured velocity field of the wing tip vortex as close as possible. The radius r is the distance from the centre of a boundary face in the inflow plane to the vortex core.

Instead of water, which is used as fluid in the experiment, the computation relies on air as fluid. The inflow velocity in x-direction u_∞ is computed to fulfil the condition that the Reynolds number in the computational test case is the same as in the experiment. The experimental conditions are a flow velocity $u_w = 1.1\,\frac{\mathrm{m}}{\mathrm{s}}$ and a Reynolds number $Re_w = 1.9 \cdot 10^5$. From the condition $Re_{air} = Re_w$ the inflow velocity in x-direction can be determined as $u_\infty = 16.21\,\frac{\mathrm{m}}{\mathrm{s}}$. For purpose of consistency the circumferential velocity v_θ has also been multiplied by the factor $\frac{u_\infty}{u_w}$. The velocity of the initial solution is set to parallel, uniform flow $u_0 = u_\infty$, $v_0 = w_0 = 0.0$.

The computation has been performed on 16 Intel Xeon E5450 processors running at 3 GHz clock speed. The computational domain matches the experimental setup which extents $l = 6\,\mathrm{m}$ in x-direction, $b = 1.5\,\mathrm{m}$ in y-direction and $h = 1.1\,\mathrm{m}$ in z-direction. The boundaries parallel to the x-direction have been modelled as symmetry walls. This domain is discretised by a coarse grid with 40 cells in flow-direction, 14 cells in y-direction and 10 cells in z-direction, respectively. The number of refinement levels has been set to $L = 6$. With this setting the vortex can be resolved on the finest level by about 80 cells in the y-z-plane.

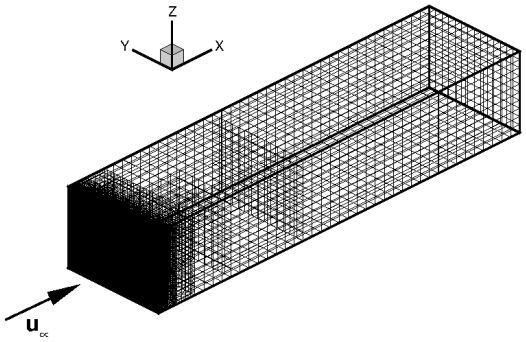

Fig. 22 Initial computational grid.

Since Quadflow solves the compressible Euler equations a preconditioning for low Mach numbers has to be applied. The preconditioning is used in a dual-time framework wherein only the dual time-derivatives are preconditioned and used for the purposes of numerical discretisation and iterative solution, cf. [21]. The spatial discretisation of the convective fluxes is based on the AUSMDV(P) flux vector splitting method [10]. For time integration the implicit midpoint rule is applied. In each timestep the unsteady residual of the Newton iterations is reduced by four orders of

magnitude. The physical timestep is set to $\Delta t = 5 \cdot 10^{-5}\,\mathrm{s}$ which corresponds to a maximum CFL-number of about $CFL_{max} = 28.0$ in the domain. The grid is adapted after each timestep. After every 100th timestep the load-balancing is repeated.

To guarantee a sufficiently fine grid to resolve the vortex properly at the start of the computation, the grid on the inflow plane is refined to the maximum level, see Figure 22. Due to this procedure the first grid contains 384000 cells. When the information at the inlet has travelled through the first cell layer, the forced adaptation of the cells at the inlet is not necessary anymore. From there on the grid is only adapted due to the adaptation criterion based on the multiscale analysis.

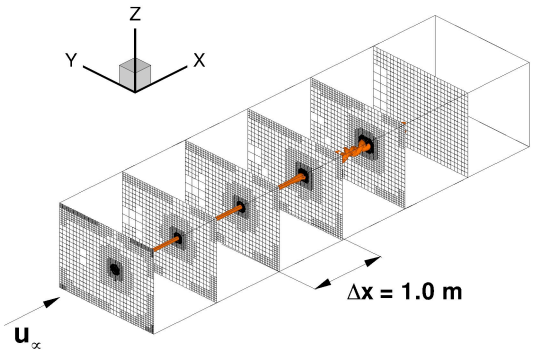

Fig. 23 Slices of the computational grid after 5466 timesteps at six different positions and the distribution of $\lambda_2 = -3$.

After 5466 timesteps, which corresponds to a computed real time of $t = 0.27\,\mathrm{s}$, the grid contains 787000 cells. Figure 23 shows six cross sections of the mesh, which are equally spaced in x-direction with distances $\Delta x = 1.0\,\mathrm{m}$. In addition, the isosurface of the λ_2-criterion with the value $\lambda_2 = -3$ is also presented. The λ_2 criterion has been proposed by Jeong et al. [14] to detect vortices. A negative value of λ_2 identifies a vortex, whereas the smallest of these negative values marks the core of the vortex. As can be seen from Figure 23, the vortex is transported through the computational domain. The locally adapted grid exhibits high levels of refinement only in the vicinity of the vortex. A more detailed view of the grid for the cross sections at $x = 0.0\,\mathrm{m}$ and $x = 2.0\,\mathrm{m}$ is presented in Figure 24.

It can be seen that only near the vortex core the grid is refined up to the maximum level. We conclude that Quadflow is well suited to examine the instationary behaviour of a vortex. In particular the complexity reduction due to the grid adaptation makes it possible to perform the computations in reasonable time.

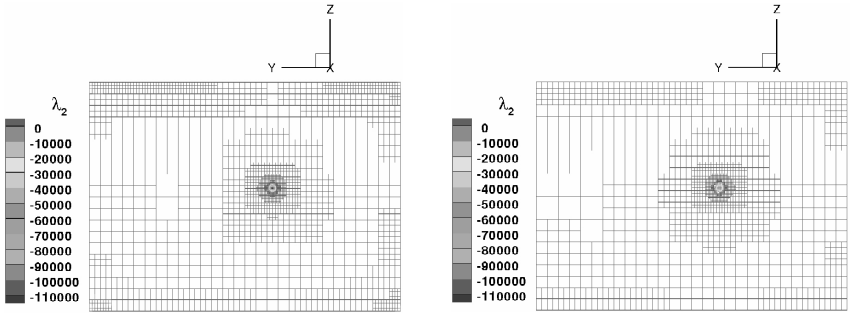

Fig. 24 Slices of the computational grid at two different positions in x-direction, the grid colour is consistent with the value of λ_2. Slice of the computational grid at $x = 0.0\,\mathrm{m}$ (left) and $x = 2.0\,\mathrm{m}$ (right).

References

1. G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.
2. S. Balay, K. Buschelmann, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang. PETSc web page, http://www.mcs.anl.gov/petsc. Technical report, 2001.
3. S. Balay, K. Buschelmann, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang. PETSc users manual. Technical report anl-95/11 - revision 2.1.5, Argonne National Laboratory, 2004.
4. S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A.M. Bruaset, and H.P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
5. F. Bramkamp. *Unstructured h-Adaptive Finite-Volume Schemes for Compressible Viscous Fluid Flow*. PhD thesis, RWTH Aachen, 2003. http://sylvester.bth.rwth-aachen.de/dissertationen/2003/255/03_255.pdf.
6. F. Bramkamp, B. Gottschlich-Müller, M. Hesse, Ph. Lamby, S. Müller, J. Ballmann, K.-H. Brakhage, and W. Dahmen. *H*-adaptive Multiscale Schemes for the Compressible Navier–Stokes Equations — Polyhedral Discretization, Data Compression and Mesh Generation. In J. Ballmann, editor, *Flow Modulation and Fluid-Structure-Interaction at Airplane Wings*, volume 84 of *Numerical Notes on Fluid Mechanics*, pages 125–204. Springer, 2003.
7. F. Bramkamp, Ph. Lamby, and S. Müller. An adaptive multiscale finite volume solver for unsteady an steady state flow computations. *J. Comp. Phys.*, 197(2):460–490, 2004.
8. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.
9. E.H. Dowell. *Aeroelasticity of Plates and Shells*. Noordhoff International Publishing, 1975.
10. J. Edwards and M.S. Liou. Low-diffusion flux-splitting methods for flows at all speeds. *AIAA Journal*, 36:1610–1617, 1998.
11. E. N. Gilbert. Gray codes and the paths on the n-cube. *Bell System Tech. J.*, 37:815–826, 1958. http://www.math.uwaterloo.ca/ wgilbert/Research/HilbertCurve/HilbertCurve.html.
12. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI - Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 2nd edition, 1999.
13. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI-2 - Advanced Features of the Message-Passing Interface*. MIT Press, 2nd edition, 1999.

14. J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995.
15. G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. *Supercomputing*, 1998.
16. G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48:71 – 85, 1998.
17. Ph. Lamby. *Parametric Multi-Block Grid Generation and Application to Adaptive Flow Simulations*. PhD thesis, RWTH Aachen, 2007. http://darwin.bth.rwth-aachen.de/opus3/volltexte/2007/1999/pdf/Lamby _ Philipp.pdf.
18. R. Massjung. Discrete conservation and coupling strategies in nonlinear aeroelasticity. *Comput. Methods Appl. Mech. Engrg.*, 196:91–102, 2006.
19. S. Müller. *Adaptive Multiscale Schemes for Conservation Laws*, volume 27 of *Lecture Notes on Computational Science and Engineering*. Springer, 2002.
20. S. Müller and A. Voss. A Manual for the Template Class Library `igpm_t_lib`. IGPM–Report 197, RWTH Aachen, 2000.
21. S.A. Pandya, S. Venkateswaran, and T.H. Pulliam. Implementation of preconditioned dual-time procedures in overflow. *AIAA Paper 2003-0072*, 2003.
22. H. Sagan. *Space-Filling Curves*. Springer, New York Berlin, 1st edition, 1994.
23. B. Stroustrup. *The C++ Programming Language*. Addison–Wesley, Reading, 1997.
24. A. Voss. Notes on adaptive grids in 2d and 3d part i: Navigating through cell hierarchies using cell identifiers. IGPM–Report 268, RWTH Aachen, 2006.
25. G. Zumbusch. *Parallel multilevel methods. Adaptive mesh refinement and loadbalancing*. Advances in Numerical Mathematics. Teubner, Wiesbaden, 2003.