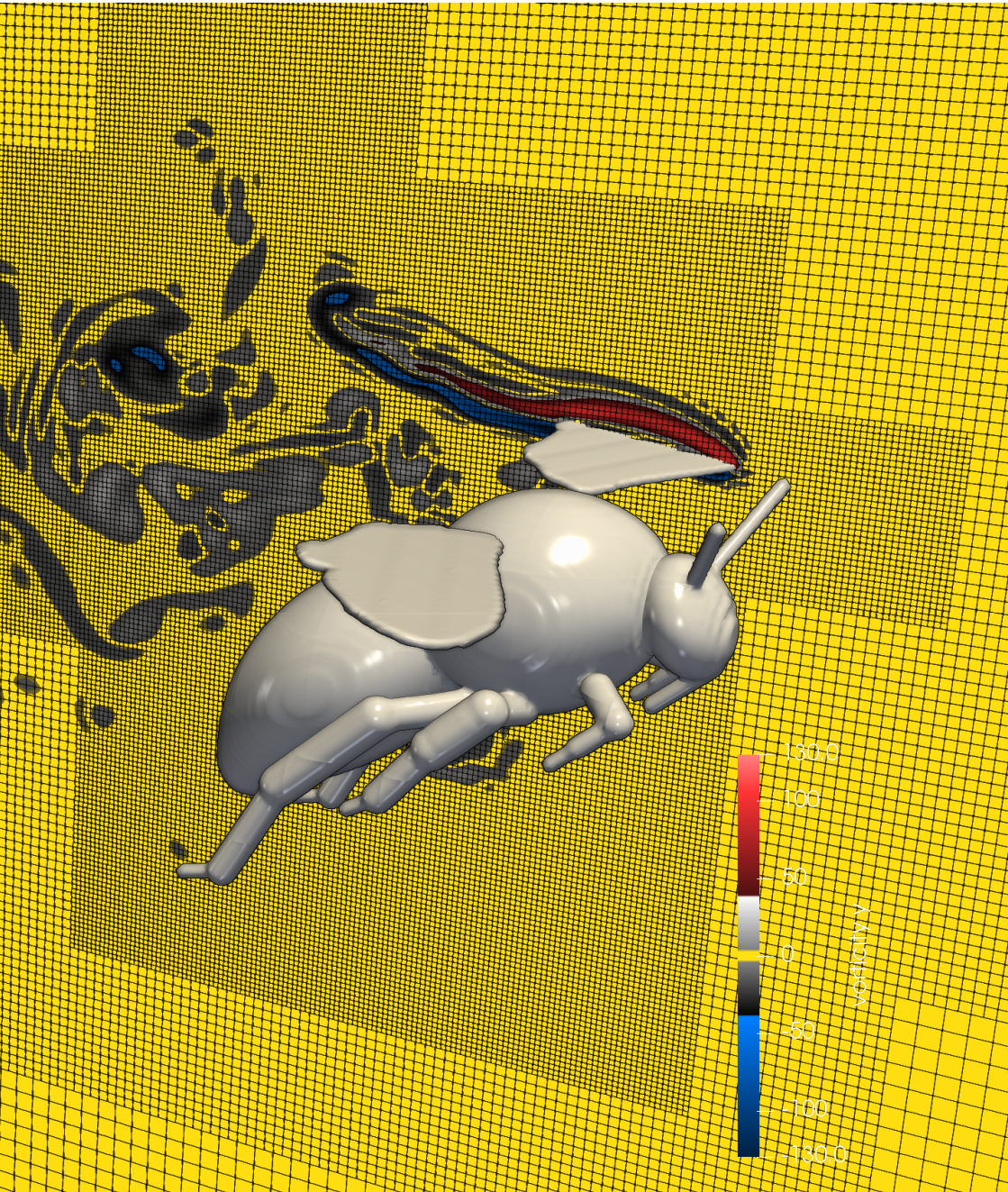
Three-dimensional simulations of flapping fliers using a multiresolution approach

Thomas Engels



Three-dimensional simulations of flapping fliers using a multiresolution approach

Thomas Engels

École Normale Supérieure
Technische Universität Berlin

Report

2018

Contents

1 Introduction

The tremendous computational complexity of insect flight, especially in turbulent flow conditions, motivates the use of adaptive numerical methods. In the turbulent flow regime a multitude of dynamically active spatial and temporal scales are involved. Introducing adaptivity can be understood in the sense that the computational effort is concentrated at locations and time instants where it is necessary to insure a given numerical accuracy, while efforts may be significantly reduced elsewhere. Adaptive discretization methods for solving nonlinear PDEs have a long tradition and can be traced back to the late seventies [3]. Adaptive methods are in many cases more competitive than schemes on regular fine grids, in particular for solutions of nonlinear PDEs exhibiting a non-uniformly distributed regularity. Essential ingredients of fully adaptive schemes are first reliable error estimators for the solution. For evolutionary problems, a major task is the time evolution of the grid and a reliable prediction of the grid for the next time step. The drawback of adaptivity is that a significant effort has to be made on programming data structures, which are usually based on graded trees, hash-tables or multi-domains. Moreover, the computational cost per cell is significantly increased. Hence, an adaptive method is only efficient when the data compression is large enough to compensate the additional computational cost per cell. Fortunately, for problems exhibiting local discontinuities, coherent vortices or steep gradients, adaptive computations are faster than fine grid computations.

Wavelets and related multiresolution analysis techniques yield reliable error indicators coupled with high computational efficiency and are thus well suited for developing adaptive solvers for computational fluid dynamics. The idea of wavelets is to decompose functions, or flow fields into space and scale contributions, and possibly directions. The wavelet transform has been discovered by Alex Grossmann and Jean Morlet in 1984 [23]. The fast wavelet transform algorithm has been designed by Stéphane Mallat [27] and yields an algorithm which is even more efficient than the fast Fourier transform. Nonlinear approximation (see e.g. [10]) provides the conceptual support for adaptivity; indeed, it introduces a systematic way to classify functions ac-

cording to their "sparsity", i.e. the possibility of describing a function by a small number of "degrees of freedom". A good practical adaptive algorithm should yield an approximation to a given accuracy of the solution of a PDE by activating a number of degrees of freedom comparable to the minimal theoretical number needed to achieve the same quality of approximation. Wavelet-based algorithms [8] for solving PDEs were the first ones to be rigorously recognized as guaranteeing an optimal complexity property. In the context of turbulence modeling and adaptive numerical simulations of fluid flows the Paris and Marseille team have been involved since the beginning of this field.

Adaptive multiresolution techniques for computational fluid dynamics are motivated by their huge potential in CPU time savings and reduction of memory requirements (for example the computations in [13] have shown that CPU and memory requirements can be reduced by at least one order of magnitude in terms of memory and CPU time for the case of compressible Euler equations), while controlling the quality of the solution. Computational efforts are automatically concentrated in regions where small scales in space and time are required. This error control allows for high fidelity and predictive computations, for example for lift and drag coefficients or pressure distributions which are of primordial importance in aerodynamics of insects. Adaptive multiresolution methods rely on classical solvers (e.g., finite volumes) and only the grid adaptation, prediction of the grid in the next time step and the interpolation require the multiresolution tools. Multiresolution techniques have become popular for hyperbolic conservation laws going back to the seminal work of Harten [24] in the context of finite volume schemes and cell-average multiresolution analysis. Starting point is a finite volume scheme for hyperbolic conservation laws on a regular grid. Subsequently a discrete multiresolution analysis is used to avoid expensive computations in smooth regions. A fully adaptive version, still in the context of hyperbolic conservation laws, has been developed to reduce also memory requirements [9, 22, 25]. This algorithm has been extended to the 3D case and to parabolic PDEs [33], and more recently to self-adaptive global and local time-steppings [11, 12, 29]. Therewith the solution is represented and computed on a dynamically evolving automatically adapted grid. Different

strategies have been proposed to evaluate the solution without requiring a full knowledge of fine grid cell- average values. The books of Cohen [7] and Müller [28] and the review article [13] give a detailled overview of the subject.

The ability of wavelets to identify and isolate localized structures such as coherent vortices, combined with the mathematical rigor of multiresolution analysis, make them attractive candidates for adaptive computational approaches and turbulence modeling [34]. Turbulence is characterized by its intrinsic multiscale behavior, its self-organization into coherent structures, and a generic randomness. The major computational challenge comes from the fact that turbulence is active over a large and continuous range of length scales, which increases with Reynolds number such as $Re^{3/4}$ for 3D turbulence. As a result, direct approaches that solve the governing Navier- Stokes equations at all scales without any model, i.e., by direct numerical simulation (DNS), are limited by the available computing power, which restricts the applicability of DNS to low- and moderate Reynolds numbers. In engineering applications, turbulence models may be necessary to reduce computational complexity while capturing the physics of turbulent flows. Wavelets offer a unique hierarchical framework for simulating and modeling turbulent flows based on the ability of wavelet multiresolution analysis to identify and isolate the energetic coherent structures that govern the dynamics of the flow. Sparse representations of turbulent flow fields coupled with wavelet-based numerical methods allow the tight integration of the computational and physics-based modeling. The wavelet-based numerical simulation, called coherent vortex simulation (CVS) was introduced by Farge et al. [20]. It adapts dynamically the local resolution of the computation to intermittent flow structures, using traditional discretization methods on these adaptive grids, e.g. finite differences. The underlying idea is the decomposition of the flow into coherent and incoherent contributions by means of wavelet filtering of the vorticity field. The evolution of the coherent flow is then computed deterministically, whereas the influence of the incoherent background flow is neglected. In the present application to insect flight, we will use CVS in the sense that we rely on adaptive meshes that are refined based on wavelets. CFD simulations, in particular insect flight in turbulent

flow conditions, produce large scientific data sets that often have localized regions of non-smooth data (or regions of very high gradients). The wavelet transform yields an optimal compression of these data sets and allows for efficient progressive refinement to reveal more detail in areas of interest, when non-smooth features are present, the visualization process that results from decoding compressed representations the specific features of the non-smooth regions can be visualized avoiding unwanted oscillations. Multiresolution transforms represent an improvement in feature detection and feature-based visualization.

2 Governing equations

We intent to approximate the incompressible Navier–Stokes equations (see, e.g., [4]),

$$\partial_t \underline{u} = -\underline{u} \cdot \nabla \underline{u} - \nabla p + \nu \nabla^2 \underline{u} \tag{2.1}$$

$$0 = \nabla \cdot \underline{u} \tag{2.2}$$

where ν is the (normalized) viscosity and the density ϱ_∞ has been normalized to unity. The approximation will be done using the method of artificial compressibility (ACM). The divergence-free constraint is approximated by an evolution equation for the pressure,

$$\begin{aligned} \partial_t \underline{u} &= -\left(\underline{u} \cdot \nabla\right) \underline{u} - \nabla p + \nu \nabla^2 \underline{u} \\ \partial_t p &= -c_0^2 \nabla \cdot \underline{u} \end{aligned}$$

where the new parameter c_0 can be interpreted as some sort of *speed of sound* which is large[1] compared to $|\underline{u}|$. This method goes back to Chorin [6] and has recently been revisited by Ohwada [30], who claims it to be quite efficient. It has the appealing property of not requiring us to solve a linear system in every time step, as classical projection methods do. As a remark we note that we cannot rewrite the non-linear term in rotational formulation ($\underline{\omega} \times \underline{u}$) anymore if the fluid is not divergence-free. The rotational formulation is

[1] One may think of the speed of sound in incompressible flow as the limit $c_0 \to \infty$

customary when working with Fourier spectral codes, as it is done in the flusi code, which we use extensively in the present work for obtaining reference simulations.

The no-slip boundary conditions on a solid body immersed in the fluid are approximated with the volume penalization technique, which has been applied successfully to a large span of problems in the past [15–19]. The penalized version of the governing ACM equations read

$$\partial_t \underline{u} = -\underline{u}\cdot\nabla\underline{u} - \nabla p + \nu\nabla^2\underline{u} - \frac{\chi}{C_\eta}(\underline{u}-\underline{u}_s) - \frac{\chi_{\rm sp}}{C_{\rm sp}}(\underline{u}-\underline{u}_\infty) \quad (2.3)$$

$$\partial_t p = -c_0^2\nabla\cdot\underline{u} - \gamma p - \frac{\chi_{\rm sp}}{C_{\rm sp}}(p-p_\infty). \quad (2.4)$$

Where we added two penalization terms, the red one for the obstacle itself (e.g., a cylinder or an insect) and the blue one to model outflow conditions. Note the latter also acts on the pressure in eqn. (2.4), whereas the former does not. By $\underline{u}_\infty$ we denote the free-stream velocity in the far field of the body; in an insect flight simulation, this would be the forward flight speed. The ambient pressure, p_∞, is set to zero, since the pressure in incompressible fluids is generally only defined up to a constant, which we may arbitrarily set to zero.

We also note the exponential damping term with the constant γ, which Ohwada [30] introduced as a way to overcome problems arising from incompatible initial conditions, for example when considering an impulsively started cylinder. In this case, $\underline{u}(\underline{x}, t=0) = \underline{u}_\infty$, thus the initial field does not satisfy the no-slip boundary condition on the cylinder. For early times t, the penalization will rapidly[2] force the velocity field to be equal to $\underline{u}_s$, say zero for a cylinder at rest. Thus, because of the penalization, a strong divergence $\nabla\cdot\underline{u}$ will be created near the cylinder, and this will be transported as a pressure wave with the velocity c_0. If the domain is periodic, as was the case in [30], this 'initial shock wave'[3] remains in the domain and deteriorates results. The constant $\gamma = 1$ was then used in [30] to gradually destroy the wave. We will

[2] as $C_\eta \ll 1$ by design of the method

[3] We use this term only to refer to the wavefront resulting from the startup singularity and do not mean discontinuities as they appear when solving the compressible Navier–Stokes equations.

see later that the sponge term is a much more effective way of doing do, but we keep the presssure damping term for completeness.

2.1 A brief look at characteristics

In the following section, we shall briefly discuss the characteristics of the artificial compressibility equation. This can be used to implement non-reflecting outlet boundary conditions in the future, but it is also useful to understand the limitation of the explicit time marching methods we use. We start with the 2D ACM equation in convective form for the inviscid ($\nu = 0$) case:

$$\begin{aligned} \partial_t u_x &= -u_x \partial_x u_x - u_y \partial_y u_x - \partial_x p \\ \partial_t u_y &= -u_x \partial_x u_y - u_y \partial_y u_y - \partial_y p \\ \partial_t p &= -c_0^2 \left(\partial_x u_x + \partial_y u_y \right) \end{aligned}$$

and rewrite it in divergence form (see, e.g., [31]: $\nabla \cdot (\underline{u} \otimes \underline{u}) = (u \cdot \nabla)\, \underline{u} + (\nabla \cdot \underline{u})\, \underline{u}$). Here, we assume $\nabla \cdot \underline{u} = 0$, which is only approximately true for c_0 finite. Then we rewrite the equation in flux-form, which yields

$$\partial_t \begin{pmatrix} u_x \\ u_y \\ p \end{pmatrix} + \partial_x \begin{pmatrix} u_x^2 + p \\ u_x u_y \\ c_0^2 u_x \end{pmatrix} + \partial_y \begin{pmatrix} u_x u_y \\ u_y^2 + p \\ c_0^2 u_y \end{pmatrix} = \underline{0}$$

and can be shortened to

$$\partial_t \underline{q} + \partial_x \underline{F}^x + \partial_y \underline{F}^y = \underline{0}$$

which we linearize to obtain

$$\partial_t \underline{q} + \underline{\underline{X}} \partial_x \underline{q} + \underline{\underline{Y}} \partial_y \underline{q} = \underline{0}$$

where

$$X_{ij} = \partial F_i^x / \partial q_j = \begin{pmatrix} 2u_x & 0 & 1 \\ u_y & u_x & 0 \\ c_0^2 & 0 & 0 \end{pmatrix}$$

and

$$Y_{ij} = \partial F_i^y / \partial q_j = \begin{pmatrix} u_y & u_x & 0 \\ 0 & 2u_y & 1 \\ 0 & c_0^2 & 0 \end{pmatrix}.$$

We seek waves in the direction $\underline{k}$ with $|\underline{k}| = 1$, which yields

$$\underline{\underline{K}} = k_x \underline{\underline{X}} + k_y \underline{\underline{Y}} = \begin{pmatrix} 2u_x k_x + u_y k_y & u_x k_y & k_x \\ u_y k_x & u_x k_x + 2u_y k_y & k_y \\ c_0^2 k_x & c_0^2 k_y & 0 \end{pmatrix}$$

we now seek to diagonalize the matrix $\underline{\underline{K}}$. The eigenvalues are

$$\begin{aligned} \lambda_1 &= \underline{k} \cdot \underline{u} \\ \lambda_{2,3} &= \underline{k} \cdot \underline{u} \pm \sqrt{c_0^2 + (\underline{k} \cdot \underline{u})^2} \end{aligned}$$

The direct consequence of the above is the selection of our time step Δt based on the CFL condition:

$$\Delta t = \mathrm{CFL} \cdot \frac{\Delta x}{|\underline{u}| + \sqrt{c_0^2 + |\underline{u}|^2}}$$

In our code, we employ a Runge-Kutta scheme with fourth order accuracy in time. This scheme remains stable for $CFL < 1.5$, but in practice we always used $CFL = 1.0$ in the present work. The sponge- and penalization terms impose additional time step restrictions if integrated explicitly, namely

$$\Delta t < \min(C_{sp}, C_\eta)$$

as does the Laplacian term, namely

$$\Delta t < 0.5 \frac{\Delta x^2}{\nu}.$$

The smallest of the above restrictions finally sets the time step in our solver.

3 Numerical solution on adaptive grids

3.1 Fundamental ideas of the implementation

The first new idea of the WABBIT code has been described above. It consists in relaxing the incompressibility constraint $\nabla \cdot u$ and employing the ACM equations instead of the classical incompressible Navier–Stokes equation. In future research, this approach can likewise be used to design an *active penalization* method[4] which is of higher order than the classical method (see, e.g., [1, 2, 5]).

The concept of adaptivity is to concentrate computational efforts where they are required by the solution at time t. The grid shall thus be refined where fine scale structures exist, while it can be coarsened in regions where the flow is smooth and regular. Multiresolution indicators will be used to detect those regions. Theoretically, those indicators can decide for each single grid point whether it is significant or not. However, this kind of sparse grid is very difficult to implement in a high-performance framework. Hence, many results in the literature on such grids have been obtained using single-core machines (e.g. [32, 33]). As first suggested by Domingues[5] [14], we employ a block-based formulation instead of a point-based one. This kind of hybrid datastructure introduces a free parameter, the block size B_s, which can be used to balance the interest of sparsity and CPU efficiency. The computational efficiency of blocks is related to their layout in memory: a block is a contiguous chunk of data. Internally, the entire chunk is transferred from the memory to the CPU cache, were the actual calculations are performed. The bandwidth of this datatransfer is an important bottleneck for modern CPU. Hence, if the complete chunk is transferred, the computation time is drastically decreased, even though more points have been computed. Our grid is graded, that means we allow at most a factor two difference in resolution between two neighboring blocks. Each block is identical in size and contains B_s^D (where $D = 2$ or 3) points. We do not split blocks among mpiranks. If a

[4] Previous attempts to improve the penalization method failed due to the elliptical constraint $\nabla \cdot u = 0$, hence we have some hope that combined with the ACM, those methods can work

[5] The idea of using blocks is older than the work of Domingues et al., but they were the first to propose it in a multiresolution context

block is marked for refinement, it is split into 2^D blocks of the same size. The reverse process, coarsening, combines 2^D blocks into one mother block.

3.2 Multiresolution algorithm

The general solution process to advance the numerical solution $\varphi\left(t^n, x\right)$ on the grid $\mathcal{G}^n$ to the new time level $n+1$ can be outlined as follows.

1. Refinement. We assume the grid $\mathcal{G}^n$ is sufficient to adequately represent the solution $\varphi\left(t^n, x\right)$, but we cannot suppose this will be true at the new time level. Non-linearities and other sources may create scales that cannot be resolved on $\mathcal{G}^n$. Therefore, we have to extend $\mathcal{G}^n$ to $\widetilde{\mathcal{G}}^n$ by adding a "safety zone" [34] to ensure that the new solution $\varphi\left(t^{n+1}, x\right)$ can be represented on $\widetilde{\mathcal{G}}^n$.

2. Evolution. On the new grid $\widetilde{\mathcal{G}}^n$, we solve the partial differential equations using finite differences and explicit time-marching methods.

3. Coarsening. We now have the new solution $\varphi(t^{n+1}, x)$ on the grid $\widetilde{\mathcal{G}}^n$. The grid $\widetilde{\mathcal{G}}^n$ is a worst-case scenario and guaranteed to resolve $\varphi(t^{n+1}, x)$ using *à priori* knowledge on the non-linearity. It can now be coarsened to the final grid $\mathcal{G}^{n+1}$, removing, in part, blocks created during the refinement stage.

4. Load balancing. The remaining blocks are, if necessary, redistributed among MPI processes using a space-filling curve [36], such that all processes compute approximately the same number of blocks. The space-filling curve allows preservation of locality and reduces interprocessor communication cost.

3.3 Discretization in space and time

We use a fourth order spatial finite difference discretization as proposed by Tam [35]. This scheme has been obtained as weighted combination of a sixth and fourth order scheme, where the weight has been chosen in order to optimize the modified wavenumber k'. Consider the 1D case, where u_i are the

function values on the grid. Then the first derivative is approximated as

$$u' = D_{TW}u = (\alpha D_{4th} + (1-\alpha)D_{6th})u$$

where

$$D_{4th} = \frac{1}{\Delta x}\begin{pmatrix} & & & \ddots & & & \\ \cdots & \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} & \cdots \\ & & & \ddots & & & \end{pmatrix}$$

and

$$D_{6th} = \frac{1}{\Delta x}\begin{pmatrix} & & & \ddots & & & \\ -\frac{1}{60} & \frac{3}{20} & -\frac{3}{4} & 0 & \frac{3}{4} & -\frac{3}{20} & \frac{1}{60} \\ & & & \ddots & & & \end{pmatrix}.$$

The weight $\alpha = -0.5912$ is the result of the optimization of k'. For the second derivatives, we employ standard 4th order stencils. The time integration is performed with a standard Runge–Kutta 4 scheme.

3.4 Choice of numerical parameters

The block size B_s The block size is the central computational parameter for our adaptive framework. It has no influence on the physics, but is rather the parameter that defines our hybrid data structure. Blocks of size B_s^D are contiguous in the machine memory and can thus be computed efficiently. From the internal point of view of the CPU, a block is completely read into the fast cache memory. Then, all operations can be performed, without additional access to the slow RAM of the machine. This sort of cache-aware loop structure is also used to speed up non-block based implementations (in which case the chunks become purely virtual, while our blocks may feature jumps in resolution to their neighbors). It can furthermore be highly efficient for vector machines, like the Earth Simulator in Japan.

Theoretically, setting $B_s = 1$ recovers the point value multiresolution grids with maximum sparsity. In those, each point is associated to a wavelet coefficient (detail). Those grids feature the lowest number of points, but the overhead of the data structure is prohibitive. Note that it is furthermore

technically not possible to set the block size lower then the number of ghost nodes, $B_s < g$, in our code.

The load balancing frequency Computational parameter, no influence on the physics. During the refinement and coarsening steps of the multiresolution algorithm, the number of blocks on each mpirank can become unbalanced, i.e., some mpirank hold more blocks than others. The balance is restored using the balance_load procedure, which did however in some simulations turn out to be expensive in terms of execution time. Hence, a switch has been added to perform the balancing only every couple of time steps. In this work, however, we use dealiasing of the finest blocks, and hence can use load-balancing in every time step without impact on the performance.

The artificial speed of sound c_0 Compared to the world of incompressible fluid dynamics, the new constant c_0 appears and has to be set accordingly. We have second order convergence, which we will demonstrate later. In order to appropriately model incompressible fluid, the relation $|\underline{u}|_{max} << c_0$ should be satisfied. Furthermore, tracking the divergence $\nabla \cdot \underline{u}$ over time allows a simple *a posteriori* control of the error. If the divergence is found to be too large, c_0 should be decreased.

The penalization parameter C_η The penalization parameter can be interpreted as permeability of the solid material, hence it is intuitively clear that it has to be *small*. The convergence towards the solution of the Navier–Stokes equations in the limit $C_\eta \to 0$ have been proven [1], but in a continuous setting (ignoring any discretization). The convergence rate, which is $C_\eta^{1/2}$, combined with the physical intuition do not yet yield a guideline on how to choose C_η. This can be obtained by considering the penalization as a sink for momentum, which has to compensate the momentum flux across the physical boundary $\partial\Omega_s$. This flux is mainly due to the viscosity ν, because the velocity on the wall is close to zero. Hence, a thin boundary layer forms on the *inside* of the obstacle Ω_s, and [18] suggests to keep the number of points

K_η within this layer constant. Hence,

$$K_\eta = \sqrt{\nu C_\eta}/\Delta x \tag{3.1}$$

or equivalently

$$C_\eta = (K_\eta \Delta x)^2/\nu. \tag{3.2}$$

In an adaptive simulation, Δx is not an *à priori* known constant, but rather dynamically adjusts to the solution. During this work, it did however turn out that the penalization term and its associated discontinuity is so strong that, in practice, the fluid–solid interface is always on the highest admissible level $J_{\max}$. Hence, we might well set

$$\Delta x = L_x 2^{-J_{\max}}/(B_s - 1)$$

by analogy with published results.

3.5 Ghost nodes synchronization

During this project, a considerable amount of time had to be spend on improving `wabbit`'s ghost node synchronization, i.e. the synchronization of a layer on each block which overlaps with the neighboring blocks. This procedure is critical for a good parallel performance. Together with Prof. Reiss, this routine has been completely rewritten. Since this part is technical, we limit its presentation to a brief summary of improvements.

1. In previous versions of the code, numerous problems appeared ironically at the simplest neighboring relation, where two blocks on the same level share the common boundary. Those difficulties are related to our grid definition: each block has a line of points which also belongs to neighboring blocks. This layer of points is called 'redundant nodes'. The old code assumed that those points were indeed identical, and if they weren't, errors could not be corrected. This entailed numerous difficulties: if the nodes' values differ by machine ϵ, this difference would grow and destroy the solution. The new code corrects this problem by arbitrarily assigning the 'redundant nodes' to either block and

overwriting them on the other. The choice of block is done by higher `lgt_ID`, but is arbitrary as long as it is unique.

2. The staging concept of previous codes has been reduced to at most two stages. Previous versions used up to six stages, each concerned with significant MPI communication. The new code can also work with just one stage, at the price of including one-sided interpolation stencils during the synchronization process.

3. The memory footprint of the code has been reduced by several orders of magnitude. This change rendered big computations, such as the present insect simulations, possible. The old code was limited to very small examples due to enormous memory consumption.

4 Validation and testing

4.1 Wavelet compression test

The first test we consider is to verify that the multiresolution part of `wabbit` does work properly. To this end, we use a scalar field $\phi(x,y,z)$, which we initialize as a Gaussian blob,

$$\phi(x,y,z) = \exp(-|\underline{x} - \underline{x}_0|^2 / \beta)$$

where $\beta = 10^{-2}$. The field is set on a domain of unit size, and we use a block size of $B_s = 17$ and $N_g = 4$ ghost nodes for the fourth order multiresolution operator. The blob is set on an equidistant grid with level $J_{max} = 4$. Subsequently, the grid adaptation is performed using a threshold ε, which we vary between 10^{-1} and 10^{-6}. The resulting sparse representation is then interpolated back to the full equidistant grid with $J_{max} = 4$ using the same predictor, and the error with respect to the initial field is evaluated. This error, the compression error, is expected to be proportional to ε.

Fig. 4.1 shows the error as a function of ε for this test case. The dashed line is the identity, and the error decays as expected with ε. For the largest values of ε, the block based data structure results in a grid which is too fine – the error hence is identical for $\varepsilon = 10^{-1}$ and 10^{-2}. This is the desired behavior – imposing ε results in an error *smaller* than $C\varepsilon$, where the constant C is problem-dependent.

4.2 2D: Three vortices problem

As a quantitative validation test for the flow module in 2D, we consider the three vortices problem proposed in [26]. The setup is periodic and involves no walls, making it a good starting point for the validation of our new numerical method. At time $t = 0$, three Gaussian vortices are set in the domain, two with positive and one with negative sign. Their circulation is $\Gamma = +1$

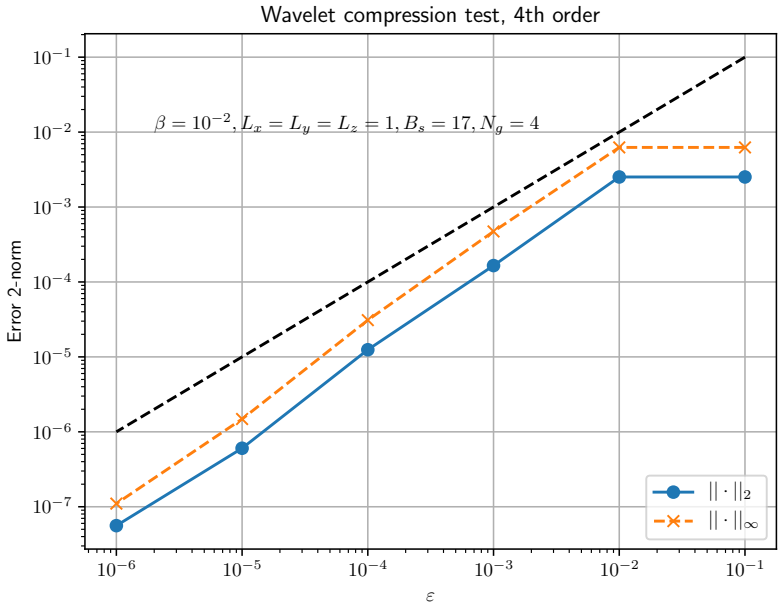


Figure 4.1: Wavelet compression test for a 3D Gaussian blob. Here, the 4th order predictor is used. Dashed line is the identity. Shown is the error decay as a function of ε for the 2-norm and the ∞-norm. Errors are normalized.

and $\Gamma = -1/2$, respectively. The initial vorticity is set as

$$\omega(x, t = 0) = \sum_{i=1}^{3} \frac{\Gamma_i}{\pi \cdot a^2)} \exp\left(-((x - x_{0,i})^2 + (y - y_{0,i})^2)/a^2\right) \tag{4.1}$$

where $a = 1/\pi$ and the vortex centers are $(0.75, 1)\pi$, $(1.25, 1)\pi$ and $(1.25, 1.25)\pi$. The viscosity is set to $\nu = 5 \cdot 10^{-5}$ and the domain size is 2π. Kevlahan and Farge [26] used a Fourier pseudo-spectral code for their simulations, but the original data is no longer available. We therefore re-compute the simulation using flusi, setting the same initial condition. We used a slightly higher resolution than what was used in [26], namely $N_x = N_y = 1536$. A Runge–Kutta 4 scheme with a CFL number of $CFL = 1.5$ is used.

Fig. 4.2 shows the time evolution of the vorticity, using the same isovalues

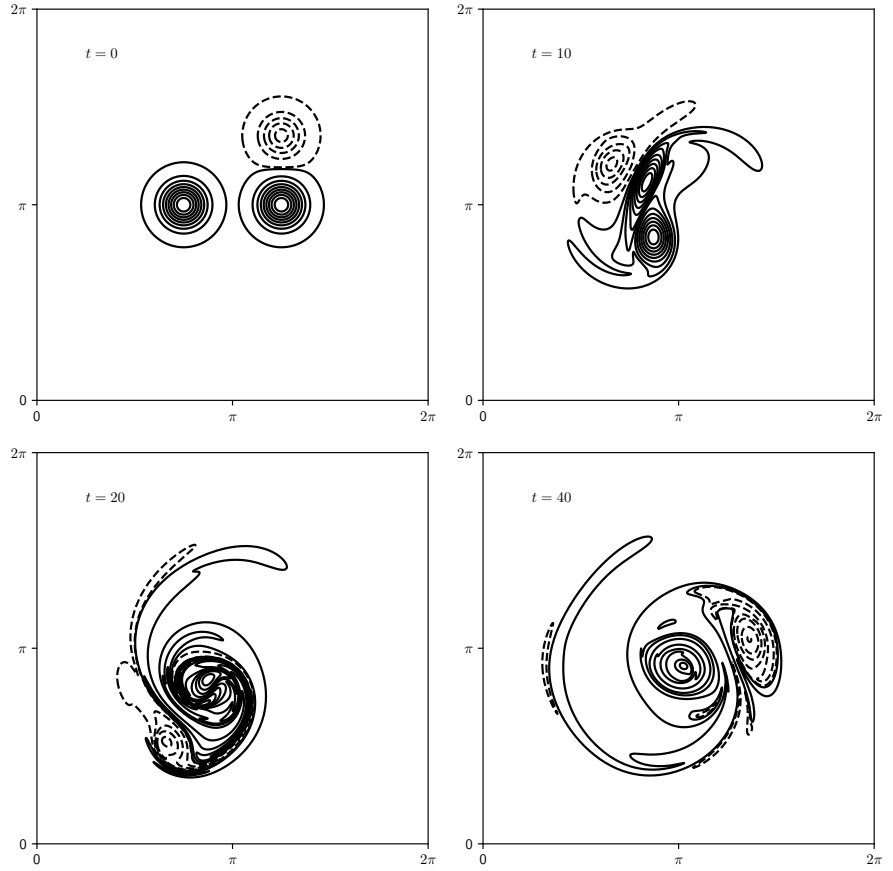


Figure 4.2: Time evolution of the vorticity ω in the three vortices problem, computed with the spectral code flusi. The initial configuration consists of three Gaussian vortices which undergo a merging process. Negative isovalues of vorticity represented by dashed lines.

of $\omega = [-\pi/100, +\pi/100]$, positive and negative values with 10 isovalues each. The same contour plot is presented in [26]. The vortices begin to rotate and eventually merge into a dipole-like structure, creating the characteristic vortex filaments. The problem is particularly suited for validation purposes as it features strong non-linear interaction between the vortices, but remains deterministic. More vortices would form a chaotic system and solvers could then only be compared in a statistical sense. Here, instantaneous comparison

of the entire field is possible.

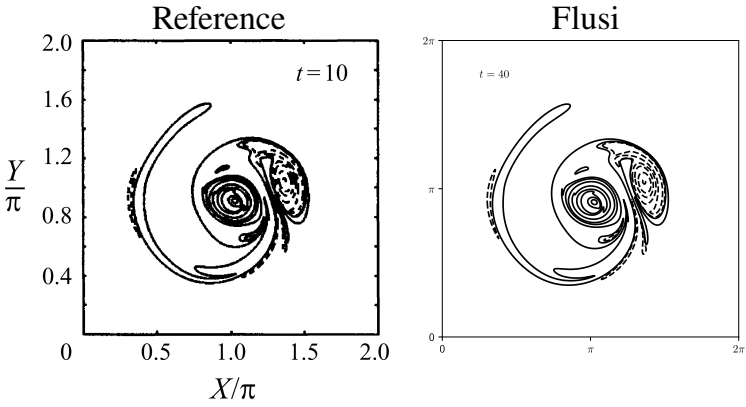


Figure 4.3: Comparison of vorticity field from [26] and flusi solution. The same isovalues for the contour are used. The fields are visually indistinguishable, justifying the flusi solution as reference data.

Fig. 4.3 compares `flusi`'s solution with the published data from [26]. Both fields agree well and are indistinguishable from each other. Note the time variable is scaled differently. This comparison confirms the validity of the reference solution.

Using this test case, we validate the fluid part of `wabbit`. To this end, we perform the following types of simulations.

1. The reference solution, obtained with the `flusi` code, where the incompressible Navier–Stokes equations are solved. The flow field is strictly divergence-free to machine precision. We use a grid of 1536×1536 points and a Runge-Kutta 4 time integrator.

2. The spectral solution of the artificial compressibility equations (2.3-2.4) with a finite c_0. This case, only the effect of c_0 enters the error contributions: the spectral discretization is sufficiently precise such that all errors from spatial and temporal discretization are below machine precision. We use the same grid as in the first type of simulations.

3. Equidistant simulations using `wabbit`. Here, the artificial compressibility eqns. (2.3-2.4) are solved using fourth order finite differences, and

hence the discretization error is no longer negligible. Note that as the initial condition (4.1) is defined in terms of vorticity, but `wabbit` cannot simply invert the curl operator. Hence, this inversion is done in `flusi`. All `wabbit` computations hence start with an equidistant initial condition for u and p, which can be directly adapted in the first time step. Note that the pressure p is computed with spectral accuracy, which can be different from an inversion of the curl using the same discretization as in `wabbit`. For the equidistant simulations, J_{min} is fixed and $J_{max} = J_{min} + 1$. Hence, we technically allow the refinement to push the grid one level up, then perform time evolution, before coarsening the entire grid down again. This ensures that the non-linear term is dealiased properly. The same could be achieved with explicit filtering, but is was technically easier to allow two levels.

4. Adaptive simulations using `wabbit`. Here, the parameter ε joins the set of numerical parameters. We now have ε, c_0 and J_{min}.

4.2.1 Equidistant computations

For all computations on the three vortices problem, we set $B_s = 33$ and $g = 4$. The choice of ghost nodes is made to use the fourth order method. We start by comparing the spectral solution of the artificial compressibility quations (2.3-2.4) with a finite c_0 to the quasi-exact solution of the incompressible Navier–Stokes equation. As the discretization can in both cases assumed to be exact (compact spectrum, small Δt), this allows us to evaluate the ACM model error alone. The error is evaluated at $t = 20$. Fig. 4.4 shows the decay of the error in that case, which is second order in c_0.

When solving the ACM equations using `wabbit`, we use a fourth order discretization in space, which we test next. The solution process is described above (type-3 simulation). Fig. 4.5 shows the error decay in this case. Note the reference solution is the spectral ACM computation with the same c_0 as the `wabbit` run, and not the incompressible solution. We have tested several values for c_0 (solid lines) and observe convergence orders between two and three. This is lower than the expected fourth order, but not low enough to point to serious errors in the code. In such cases, the convergence typically

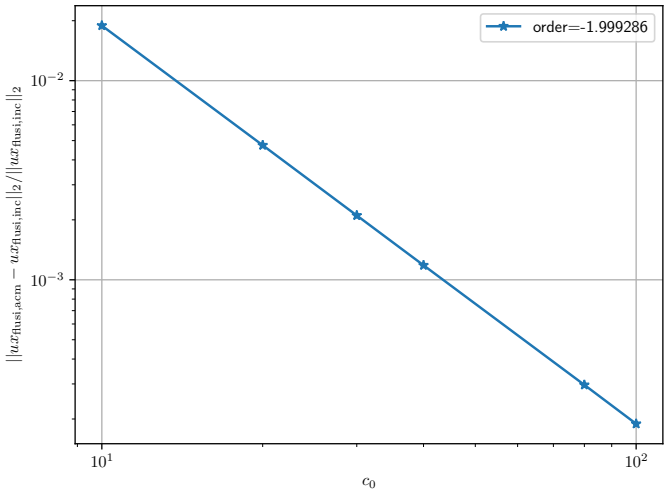


Figure 4.4: Equidistant simulations of the three vortices problem. Error of the spectral solution of the ACM equations with respect to the incompressible spectral solution. Here, c_0 is the sole parameter, since the discretization in the spectral ACM case can assumed to be quasi-exact. The model error of the ACM decays second order in c_0.

drops to first or zeroth order, which is not the case here. Our simulations are performed with dealiasing as described above, and for comparison we also present one series of simulations without it (dashed line). The convergence rate is very similar to the dealiased simulations, but the offset is a lot lower. This points to a principal difficulty with dealiasing in this code: if the viscosity is sufficiently high to remove high-wavenumber energy, which is produced by the non-linear term, then dealiasing is not necessary. This appears to be the case in this setup. Hence, dealiasing increases the error, because the right hand side is filtered effectively to be evaluated on $J - 1$. In other words, we could shift the dashed curve one datapoint to the left, to compare it with the dealiased simulation. The effect on computational performance can be quite substantial: the cost increases 16-fold (2^D times as many grid points, time step $\Delta t/2$). However, we must bear in mind that if the viscosity is too weak, the simulation accumulates energy in the highest

wavenumber and eventually diverges. This can, however, also be prevented with filters different from the prediction / restriction round trip we use in the present work. This direction is left for future work.

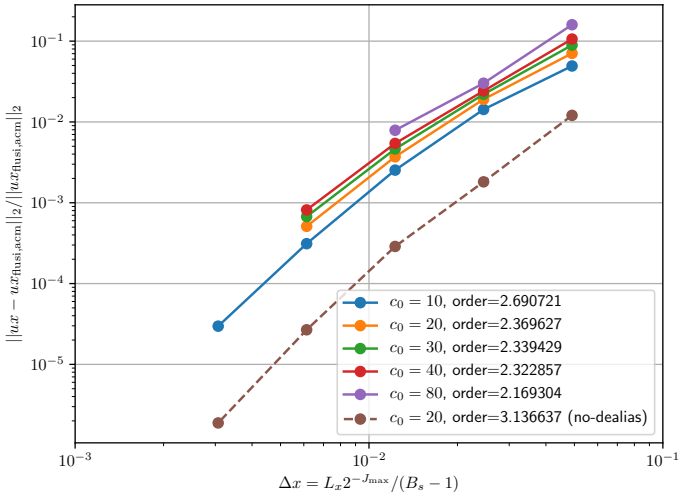


Figure 4.5: Equidistant simulations of the three vortices problem. Error of the `wabbit` *equidistant solution of the ACM equations with respect to the spectral ACM solution.*

In the next step, we compare the `wabbit` ACM solutions with the quasi-exact solution of the incompressible Navier–Stokes equations. We hence have two errors, the model error of the ACM and the discretization error. Fig. 4.6 shows the error decay. The requirement of error balancing now becomes apparent. For instance, consider the blue curve with $c_0 = 10$. While the error first decays if the resolution is increased, it later saturates due to the error of the ACM and increasing the resolution beyond a certain threshold makes no sense for fixed c_0. This threshold should be determined in future work, such that the relation $c_0(\Delta x)$ is pre-determined. It is questionable if that can succeed for the generic case, since it may be problem dependent.

Fig. 4.7 finally shows the same information represented differently and studies the error decays with c_0 at fixed resolution. Again, the initial convergence

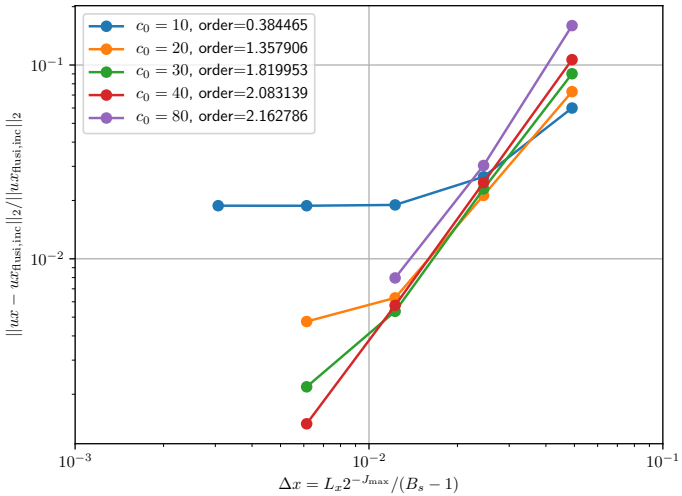


Figure 4.6: Equidistant simulations of the three vortices problem. Error of the `wabbit` *equidistant solution of the ACM equations with respect to the spectral incompressible solution.*

is the same as in Fig. 4.4, until the spatial discretization error saturates.

4.2.2 Adaptive computations

For the adaptive computations of the three-vortices problem, we are left with one additional error, which is the compression error. It is controlled via the parameter ε. Other parameters, notably the block size $B_s = 33$ are held constant. Recall that the initial condition is computed using the spectral solver, hence it is read from file and initialized on a dense grid, which is identical for all simulations (for a given J_{max}). Immediately after reading, the grid is adapted until it satisfied the prescribed precision criterion. The maximum level allowed in the simulations has the same notation as in the previous section. The right hand side is evaluated at most on J_{max}, and after time evolution coarsened to $J_{max} - 1$. This ensures proper dealiasing.

The artificial compressibility equations are supposed to approximate the in-

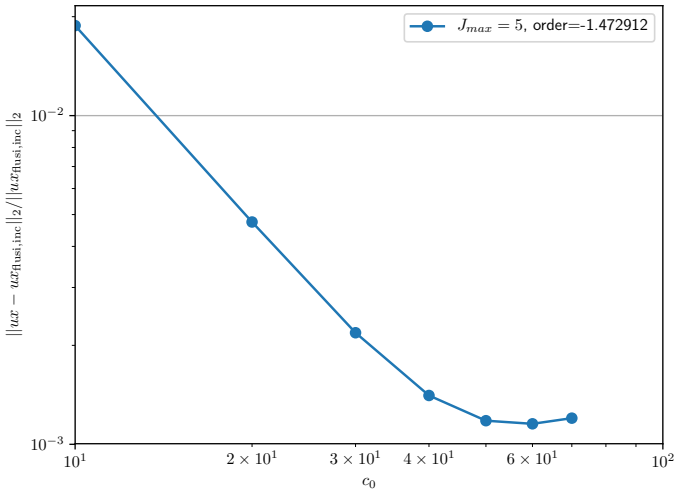


Figure 4.7: Equidistant simulations of the three vortices problem. Convergence in c_0 for fixed resolution $J_{max} = 5$, reference solution is spectral incompressible.

compressible Navier–Stokes equations, in which the pressure p is not a state variable but rather a Lagrangian multiplier that ensures $\nabla \cdot u = 0$. As a consequence, the pressure is smoother than the velocity in the incompressible solution. In the ACM this is only approximately true, $\nabla \cdot u \approx 0$, and pressure waves exist which travel in the domain. This wave-like character is often associated with relatively high frequencies, as we will see later. Hence, the pressure field in the ACM is less smooth than it is in incompressible Navier–Stokes. This leads us to the idea that the multiresolution thresholding can be applied either to the entire state vector (u, p) or to the velocity u alone. In the latter case, fine details in the pressure may be ignored, provided the velocity is sufficiently regular at this position. Here, we test both approaches.

Fig. 4.8 shows the error of the adaptive computations for a fixed $c_0 = 100$. Thresholding is applied to the complete state vector (solid lines) and to u only (dashed lines). Comparing the y-axis with the y-axis in Fig. 4.4, it is clear that the ACM error is negligible in front of discretization- and compres-

sion error in this example. The error behaves as expected. Smaller ε result in smaller errors, provided the maximum refinement level is high enough. For fixed J_{max}, the error saturates below a certain ε, when the discretization error dominates. In general, the difference in error between the two thresholding rules is small. The curves follow the same overall trend. In particular, thresholding the velocity only does not increase the total error.

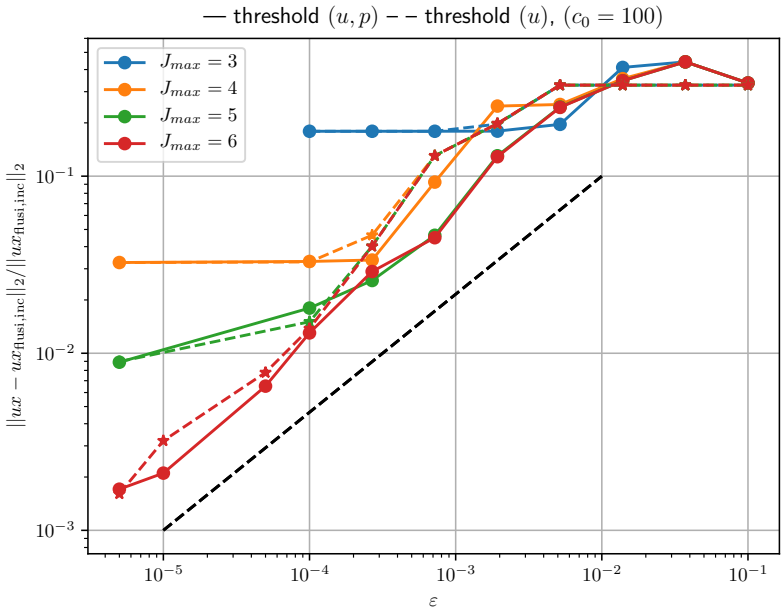


Figure 4.8: Adaptive simulations of the three vortices problem. Convergence in ε for fixed $c_0 = 100$ and different J_{max}. The error is given with respect to the incompressible solution. Note the ACM error is sufficiently small not to appear in this figure (cf. Fig. 4.4). Solid colored lines: thresholding is applied to the complete state vector (u, p). Dashed colored lines: thresholding applied only to u. Black dashed line is a slope ε^1.

While the error remains virtually unchanged by the thresholding rule, the computational cost can vary appreciably. To this end, Fig. 4.9 shows the number of blocks N_b saved to disk at the time of error comparison, which is $t = 20$ for all simulations. Note N_b is evaluated after grid adaptation

(coarsening), hence the right hand side is evaluated on $4N_b$ blocks. One can see a significant reduction in N_b especially for intermediate value of ε. Once ε is decreased sufficiently, the difference becomes small. This is because the grid becomes full in both cases. We conclude that applying the thresholding operator to the velocity only appears to be a valuable strategy.

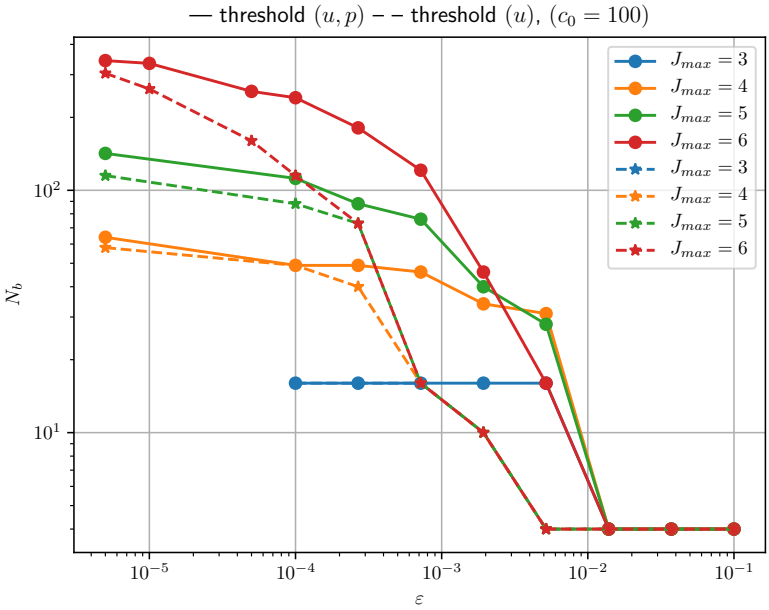


Figure 4.9: Adaptive simulations of the three vortices problem. Number of blocks at the time of comparison, $t = 20$. Note N_b is evaluated after grid adaptation (coarsening), hence the right hand side is evaluated on $4N_b$ blocks.

Fig. 4.10 shows the vorticity and divergence of the flow field for both thresholding rules. For $\varepsilon = 10^{-4}$ the number of blocks differs by a factor of two. We do however note that the fluid divergence $\nabla \cdot u$ is larger if thresholding is applied to u only.

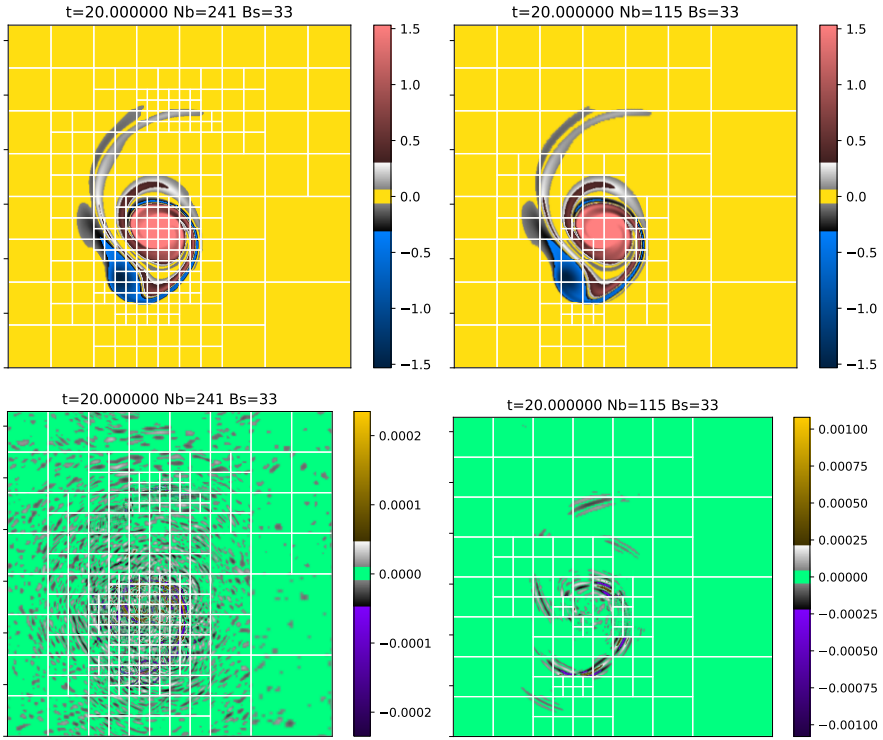


Figure 4.10: Adaptive simulations of the three vortices problem. Vorticity (top) and divergence (bottom), $\varepsilon = 10^{-4}$, $J_{max} = 6$, thresholding applied to all components (left) and velocity only (right).

4.3 2D: flow past a cylinder

We now consider a first test case involving wall, still in 2D to keep the setup simple and computational cost low. This aim of this section is to first establish the choice of parameters for the outflow sponge, then to proceed to a quantitative validation. The setup is illustrated in Fig. 4.11. It consists of a cylinder of unit diameter in a large domain which is either 16×16 or 32×32 diameters. We use a block size of $B_s = 17$ throughout the present section. In the first subsection, the Reynolds number is $Re = Du_\infty/\nu = 200$, where the mean flow $u_\infty = 1$. The initial condition is

$$\begin{aligned} u_x(x, t = 0) &= u_\infty \\ u_y(x, t = 0) &= 0.1u_\infty \\ p(x, t = 0) &= 0 \end{aligned}$$

and these conditions are enforced in the sponge layer at the domain border. The cylinder is modeled with the volume penalization method, and we set $C_\eta = 10^{-4}$ in the first part of this section.

4.3.1 Choice of parameters for the sponge

The sponge layer has two primary functions: it (a) damps incoming waves and should minimize their reflection back in to the physical domain, and (b) enforces the far field boundary conditions at the domain borders. From equation (2.3-2.4) we note that the sponge term acts also on the pressure, i.e., it imposes a Dirichlet-type boundary condition on p. The (spatial) mean of the pressure $\langle p \rangle$ does not have any physical meaning in the governing equations: we hence arbitrarily set $\langle p \rangle = 0$ for all times t. We furthermore fix $c_0 = 35$ and perform adaptive simulations ($\varepsilon = 10^{-3}$, $J_{max} = 8$, dealiased).

The shape of the sponge layer is imposed by the desired minimization of reflections. Fig. 4.12 shows the sponge mask. It has unit values directly on the border of the domain and becomes zero at a distance L_{sp}, which we set to $L_{sp} = 0.75$ in the present experiments. The shape in between is a polynomial with zero gradients at the extremities.

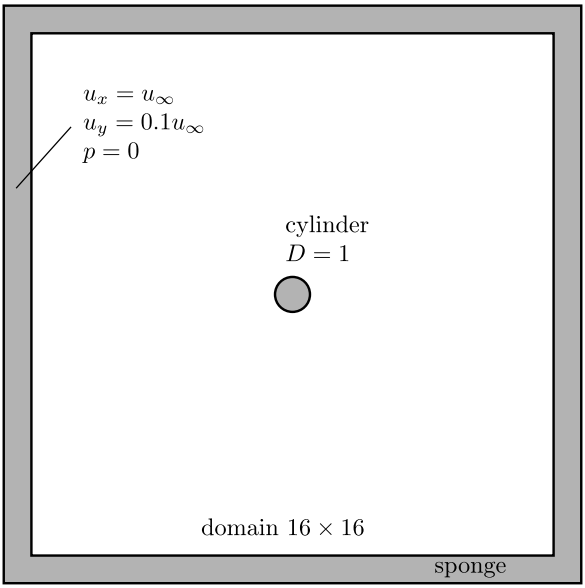


Figure 4.11: 2D flow past a cylinder, setup description.

In the spectral code `flusi` [15–18], sponges have likewise been used to remove the inherent periodicity of the Fourier discretization on which this code relies. Those sponges have however different properties than the ones we use here: namely, they act on the vorticity. The sponge constant was in those simulations set to a rather large value, say $C_{sp} = 10^{-2}$ or even larger. This can allow vortex pairs to enter the sponge collectively. If the sponge is too strong, it can happen that one out of two vortices in a traveling dipole is destroyed quickly, and then leaves the other one orphaned in the domain.

The physics in the present work is however different, which becomes particularly apparent for impulsively started flow, as we treat them in this case. The penalization quickly forces the flow inside the cylinder to zero, which results in the creation of divergence $\nabla.u$. This is then transported as a wave which velocity c_0. Fig. 4.13 illustrates this behavior for three distinct times. The impact of the wave on the sponge takes place at

$$t \approx \frac{L}{2}\frac{1}{c_0} = 0.229$$

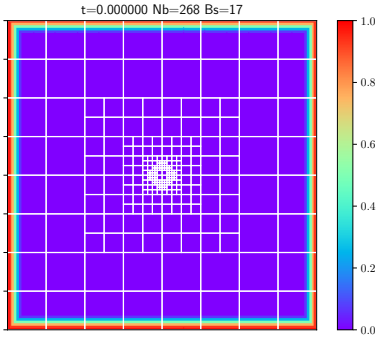


Figure 4.12: Sponge mask as used in the simulations. The sponge is unity on the border and zero at a distance L_{sp}, in between a polynomial expression is used.

where some uncertainty arises from the finite size of the cylinder and sponge layer (hence the traveled distance is not exactly $L/2$).

We test three different values for the sponge parameter, $C_{sp} = 10^{-2}, 10^{-3}$ and 10^{-4}. The pressure field at $t = 0.34$ is shown in Fig. 4.14 for those three cases. One can observe that $C_{sp} = 10^{-2}$ is not sufficient: a significant portion of the wave is transmitted to the other side of the sponge, i.e. it passed through the sponge layer. The best result is obtained with $C_{sp} = C_\eta = 10^{-4}$. We hence recommend for future use to set the penalization and sponge parameter to the same value as the penalization parameter.

The reflected or transmitted waves unfortunately have an impact on the forces excerted on the cylinder. Fig. 4.15 shows the drag component, i.e., f_x, as a function of time. Around $t = 0.4$, the shock waves hit the cylinder and result in a sudden and unphysical change in the drag force. Smaller sponge constants result in smaller perturbation at this point, but even in the $C_{sp} = 10^{-4}$ case a difference to the reference computation with twice the domain size ($L = 32$, $\varepsilon = 10^{-3}$, $J_{max} = 9$) is visible. This is due to the remaining reflection from the sponge.

The domain size has a significant influence on the problem as well, as larger domains allow the pressure waves to decrease their intensity by simply 'spreading' their content over a larger area. This decay in peak pressure decays as

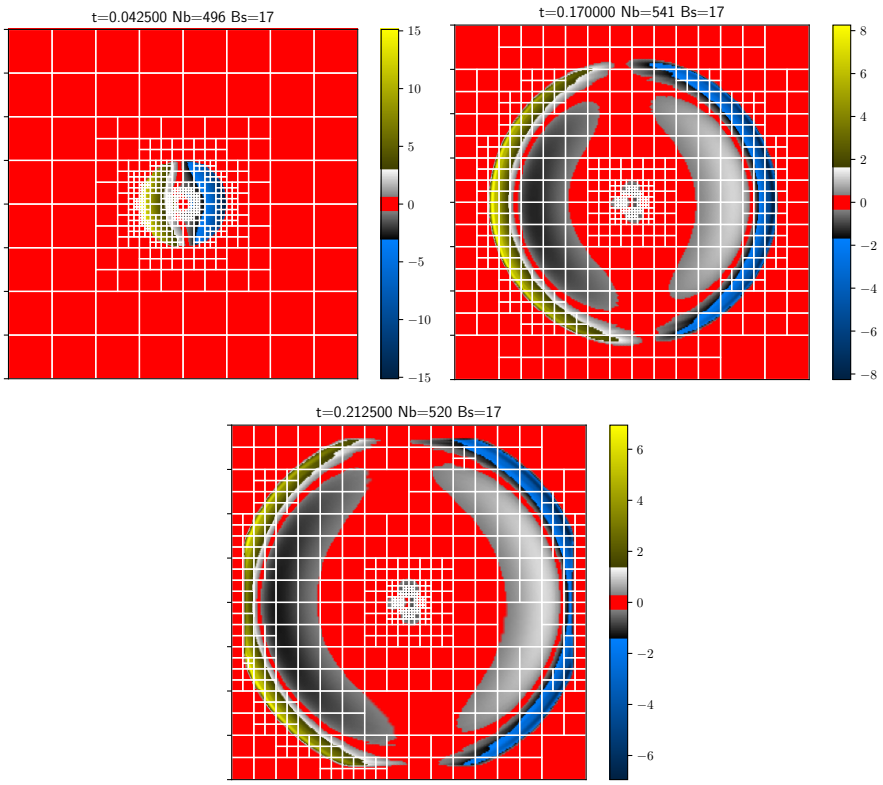


Figure 4.13: Propagation of the initial 'shock wave' created by the impulsively started cylinder. Shown is the pressure p at three different times during the early stage of the simulation.

R^{-1} in 2D and R^{-2} in 3D configurations. Hence, the entire problem will be less significant in 3D simulations. For illustration of this fact, Fig. 4.16 shows the pressure field at $t = 1$ in the reference simulation with the larger domain, 32×32. We conclude that the simulations can be sensitive to the design of the sponge, and that constants as they have been used in spectral computations cannot be used. The design of sponges is also a direction for future improvements of the code.

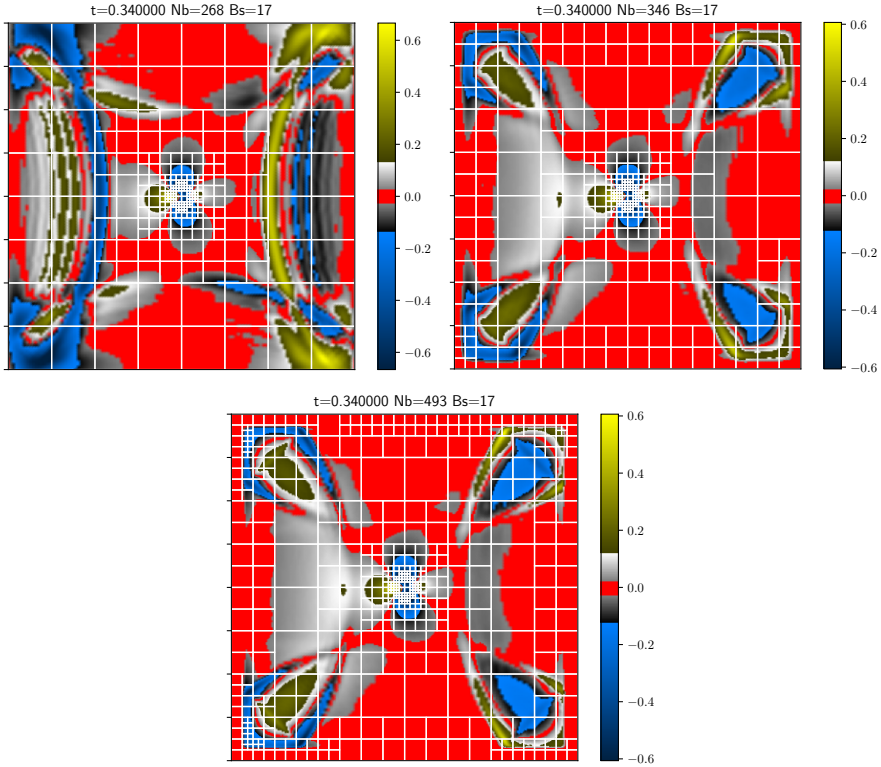


Figure 4.14: 2D flow past a cylinder. Pressure field at $t = 0.34$ for the three sponge constants $C_{sp} = 10^{-2}, 10^{-3}$ (top left to right) and 10^{-4} (bottom). Except the sponge constant all other parameters are identical. In the 10^{-2} case, a significant portion of the shock wave is transmitted.

4.3.2 Validation for $Re = 40$

For a quantitative validation of `wabbit`'s numerical method, which is the artificial compressibility coupled with the volume penalization, we now consider again the flow past a cylinder. The choice of physical parameters is dictated by the work [21], where a convenient reference simulation is presented, together with a detailed literature overview on both experimental and numerical results on this flow configuration. The Reynolds number is $Re = 40$. This Re is in the deterministic regime, and a steady state solution

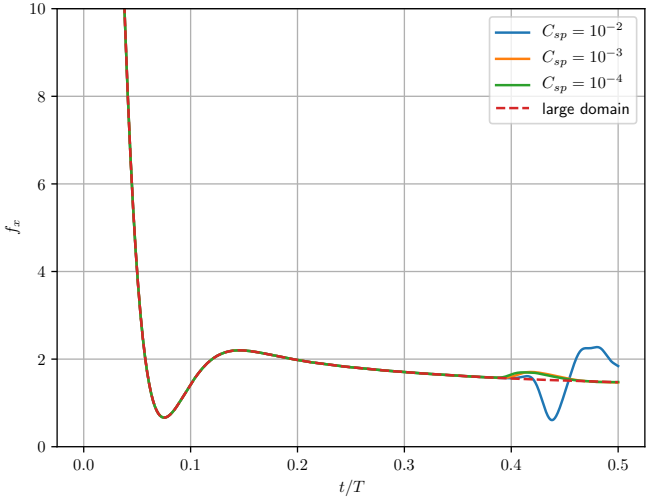


Figure 4.15: 2D flow past a cylinder. Drag force as a function of time for different sponge intensities. The red dashed line is a reference computation with twice the domain size; in this case, the pressure waves did not yet hit the cylinder.

develops. No periodic vortex shedding occurs, which simplifies the comparison with the literature.

Note this test case allows a quantitative validation of the code, but its low *Re* limits the usefulness of an adaptive computation. The grid is effectively adapted only in the startup phase of the simulation, until the pressure shock waves discussed in the previous section have disappeared. Afterwards, the grid remains practically constant. We should however point out, that the adaptive code is still highly useful for the present computation. This is because of the considerable domain size required by the low Reynolds number. Even though the case is 2D only, the computation using `flusi` is quite challenging: for an appropriate domain size, the resolution easily ranges in the 8192^2 class.

Here, we set the domain size to $L = 32$ diameters, and compute 15 time units (based on unit free stream velocity u_∞ and unit diameter D). The initial condition is again an impulsively started flow, as in the previous section.

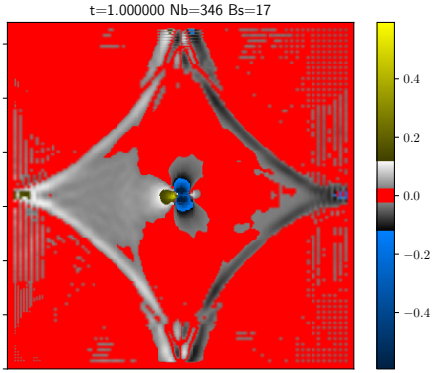


Figure 4.16: Flow past a cylinder in 2D. Shock reflection in the case of the larger domain (32×32). Due to the larger area, the intensity of the shock decays, which also reduces reflections.

Fig. 4.17 shows the quantities which are used for comparison. The drag coefficient $c_D = 2F_x$ is directly obtained from `wabbit`'s output. The force is computed by integration of the penalization term. The geometrical properties of the streamlines are somewhat more difficult to obtain. The streamlines are computed using a numerical integration algorithm in `paraview`; a high-resolution line seed is used. The geometrical properties were measured manually from this data, as the process is rather difficult to automatize and useful only for this simulation.

Table 1 summarizes the different simulations we have performed. The first series, A1–A5, studies the influence of the penalization parameter C_η, which turns out to be by far the most important numerical parameter in the present setup. From the literature we gather the idea of keeping the number of points in the penalization boundary layer constant, see eqn. (3.1). In a spectral simulation this was a successful approach. In the setup we use in the present work, the choice of K_η appears to be much more crucial. Simulation A1 uses $K_\eta = .15$, which is a typical value used in spectral computations. The drag coefficient c_D is however completely off and much too large by a factor of three. As the time step is $\Delta t < C_\eta$, we perform very small time steps in run A1. Hence, we could allow a large speed of sound c_0 at no extra cost. Note besides the very large drag coefficient, the wake properties are actually quite

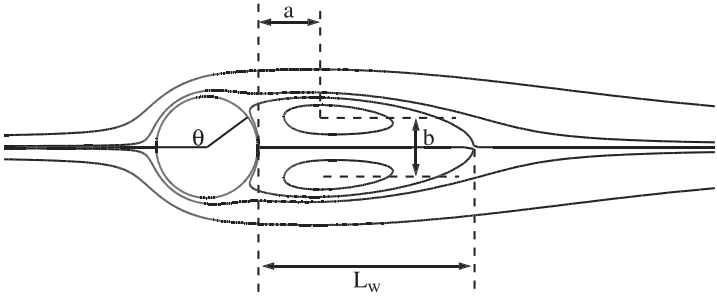


Figure 4.17: Definitions of quantities for comparison, figure adopted from [21]. Besides the drag coefficient, the wake length L_w and the center of recirculation a, b are used for comparison. Because of time constraints and technical difficulties, the angle θ is not used in the present work.

reasonable and not off by the same factor. We subsequently reduced K_η in runs A2–A5 and ultimately found a good agreement with the literature. For this example, the constant K_η should be in the range 1–3, which is considerably larger than in the spectral code. Note we did not decrease c_0 below 25.6 in the A series.

The speed of sound c_0 is the parameter investigated in series B. We keep $K_\eta = 1.5$ and vary c_0 between 12 and 50. The results are not very sensitive to c_0: barely any change can be observed. Note that the inflow velocity is $u_\infty = 1$ and the maximum velocity is about $u_{max} = 2$, hence the ratio c_0/u_{max} is at least 6. This value appears rather low, but still yields good results. We conjecture that this is a particular property of the steady state solution, where c_0 may eventually drop out of the equations. Specifically, we do not expect the same behavior in more complex and unsteady flow configurations.

The last series, C, investigates the influence of ε and J_{max} on the results. Both are found rather insignificant: the solution for $\varepsilon = 10^{-4}$ is virtually identical to the one with $\varepsilon = 10^{-3}$. Sole J_{max} slightly modifies the drag coefficients, while the wake parameters remain unchanged (to within the measurement precision). The trend with J_{max} is a decrease in drag, which is consistent with the literature, as our drag values are slightly too high.

Fig. 4.18 finally shows the flow field for run A5 (top) and A1 (bottom). The vorticity ω is shown as background pseudocolor plot, superimposed with

Name	Parameters				Results			
	c_0	K_η	J_{max}	ε	c_D	L_w/D	a/D	b/D
A1	280	0.15	9	10^{-3}	4.78	2.20	0.73	0.59
A2	25.6	0.50	9	10^{-3}	2.25	-	-	-
A3	25.6	0.75	9	10^{-3}	1.95	-	-	-
A4	25.6	1.50	9	10^{-3}	1.71	2.12	0.69	0.56
A5	25.6	2.50	9	10^{-3}	1.64	2.06	0.67	0.58
B1	12.0	1.5	9	10^{-3}	1.70	2.11	0.69	0.58
B2=A4	25.6	1.5	9	10^{-3}	1.71	2.12	0.69	0.56
B3	50.0	1.5	9	10^{-3}	1.73	2.10	0.68	0.58
C1	12.0	1.5	9	10^{-4}	1.71	2.13	0.70	0.58
C2	12.0	1.5	10	10^{-3}	1.63	2.13	0.70	0.58
Ref computation [21]					1.49	2.24	0.71	0.59
Range in literature [21]				min	1.48	2.13	0.71	0.59
				max	1.62	2.35	0.76	0.60

Table 1: Flow past a cylinder at $Re = 40$. Key parameters for the simulations are c_0, K_η, J_{max} and ε. Since no automated procedure exists for extracting L_w, a and b from the simulation data, it has been measured manually. Hence, for simulations where the drag coefficient is too different from the reference value, this extraction has not been done. For comparison, the values from the reference computation [21] are also shown, together with the min and max values from all numerical and experimental papers cited therein.

grid lines (white) and stream lines (black). The qualitative pattern is visually very similar to published results. Note the striking similarity of both flow fields, despite a large difference in the drag coefficient. We can from this data conjecture that the drag coefficient may not be appropriately computed using the volume integration; the solution might not be very sensitive to C_η (or K_η), if a different method to obtain the drag force is used, c.f. for example [2].

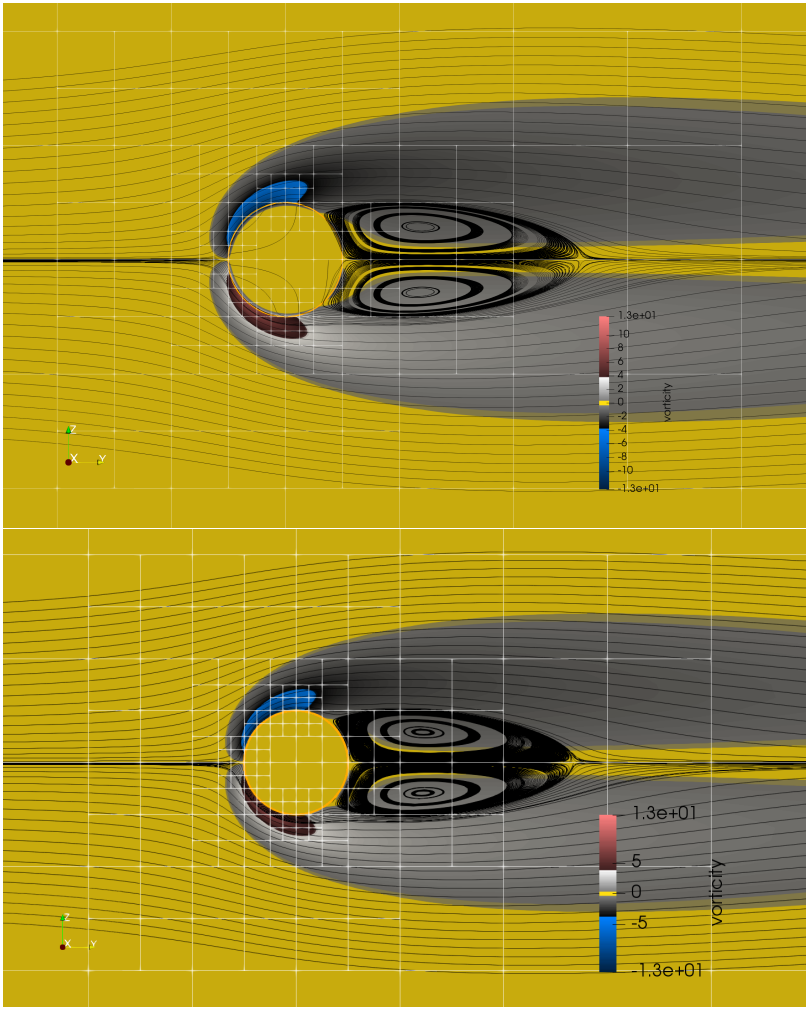


Figure 4.18: Flow past a cylinder at $Re = 40$, `wabbit` computation. Key parameters of the top simulation are: $K_\eta = 2.5$, $T = 15$, $J_{max} = 9$, $c_0 = 25.6$. Key parameters of the bottom simulation are: $K_\eta = 0.15$, $T = 15$, $J_{max} = 9$, $c_0 = 280$. Shown is the vorticity as background pseudocolor-plot, the grid lines (white) and streamlines (black) which have been used to extract the geometrical quantities defined in Fig. 4.17.

4.4 3D: Propagation of a pressure blob

In this section we compare the solution obtained with `wabbit` to that obtained with `flusi`. The configuration is intended as a numerical convergence test, between two codes which solve the same equation differently, and the solution is not physically relevant. We therefore do not test the convergence with respect to c_0. The initial condition is quiescent flow, $\underline{u}(t=0)=0$, combined with a Gaussian blob for the pressure,

$$p(\underline{x}, t=0) = \exp(-|\underline{x}-\underline{x}_0|^2/\beta)$$

where $\beta = 0.01$. We fix the parameter c_0 arbitrarily to 15, the domain size is $1\times1\times1$, the block size $B_s = 17$ and $\nu = 10^{-5}$. For this configuration, the initial pressure distribution forms waves that propagate through the periodic domain. No sponges are used to remove the spatial periodicity. Note this solution cannot exist in perfectly incompressible fluids, where $\nabla.\underline{u} = 0$ and hence $p(t,\underline{x}) = 0$ for quiescent fluid.

The reference solution is obtained using the spectral code `flusi` and likewise solves the artificial compressibility equations 2.3-2.4, but uses a spectral discretization combined with a Runge–Kutta 4 time discretization. We compare the solution at $t = 0.1$. The pressure waves have at this time traveled $t/c_0 = 1.5$ length units, which is larger than the domain size. While the time for comparison is somewhat arbitrary, it should be sufficiently long to allow errors to build up, which is hence ensured with our choice. For the reference solution, we used a 512^3 grid and a 768^3 grid to check if the resolution is sufficient and the reference solution can be trusted. Fig. 4.19 shows the spectra of the x-component of the velocity and the pressure at $t = 0.1$. The magnitude of the Fourier coefficients is virtually identical for both resolutions, and only differs in the highest wavenumbers. The magnitude of those coefficients is however so small ($< 10^{-20}$) that both spectra can be called 'compact in Fourier space'. Hence, the code converges spectrally, and the solution is extremely precise. We conclude that the reference solution is sufficiently precise.

We perform two series of simulations with `wabbit`: equidistant and adaptive.

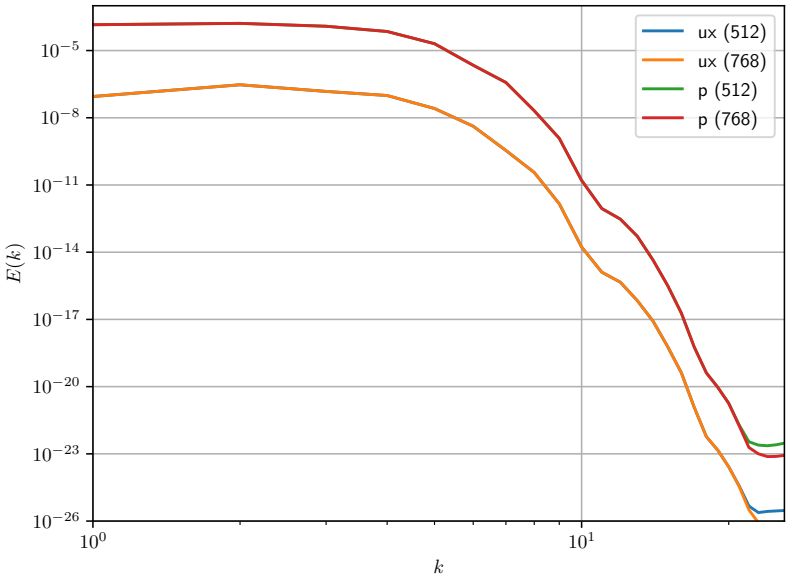


Figure 4.19: Pressure blob test in 3D. Spectra of flusi solutions for the pressure blob at time $t = 0.1$. Shown are the magnitudes of the Fourier coefficients of the x-velocity and pressure, both for two resolutions (512^3 and 768^3). The k-axis is show only until $k = 26$, because magnitudes of higher wavenumbers are zero to machine precision.

For the equidistant simulations, $J_{min} = 2, 3, 4$ is fixed and $J_{max} = J_{min} + 1$. Hence, we technically allow the refinement to push the grid one level up, then perform time evolution, before coarsening the entire grid down again. This ensures that the non-linear term is dealiased properly. The same could be achieved with explicit filtering, but is was technically easier to allow two levels. The computational overhead of the approach was very small. Fig. 4.20 shows the error as a function of the resolution $\Delta x = 2^{-J_{\min}}/(B_s - 1)$, for both u_x and p. The expected 4th order convergence is observed for both quantities. This test confirms that `wabbit`'s right hand side is properly implemented in the 3D case.

In the adaptive simulations, we set $\varepsilon = (10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6})$

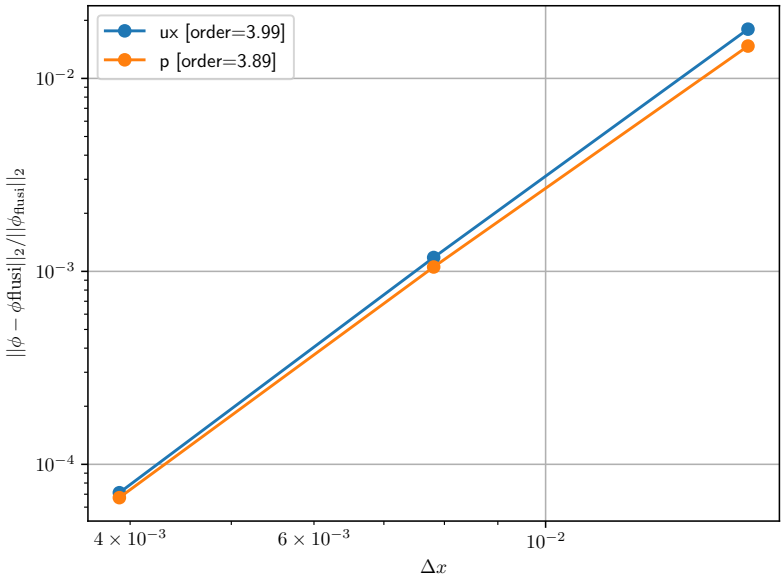


Figure 4.20: Pressure blob test in 3D. Error decay of `wabbit`'s solution on an equidistant grid as a function of the resolution. The reference solution is obtained using the spectral code. For both the x-component of velocity and the pressure, the expected fourth order is obtained.

and $J_{max} = (3, 4, 5, 6, 7)$. All other parameters are the same as in the previous tests. Fig. 4.21 shows the error as a function of ε and J_{max}. For large values

4.5 3D: performance considerations

The choice of computational parameters can depend on the test case and its physical parameters. As a first test, we consider here the method of artificial compressibility in three space dimensions. Hence, the state vector has four entries. The initial condition is quiescent flow, $\underline{u}(t = 0) = 0$, combined with a Gaussian blob for the pressure,

$$p(\underline{x}, t = 0) = \exp(-|\underline{x} - \underline{x}_0|^2 / \beta)$$

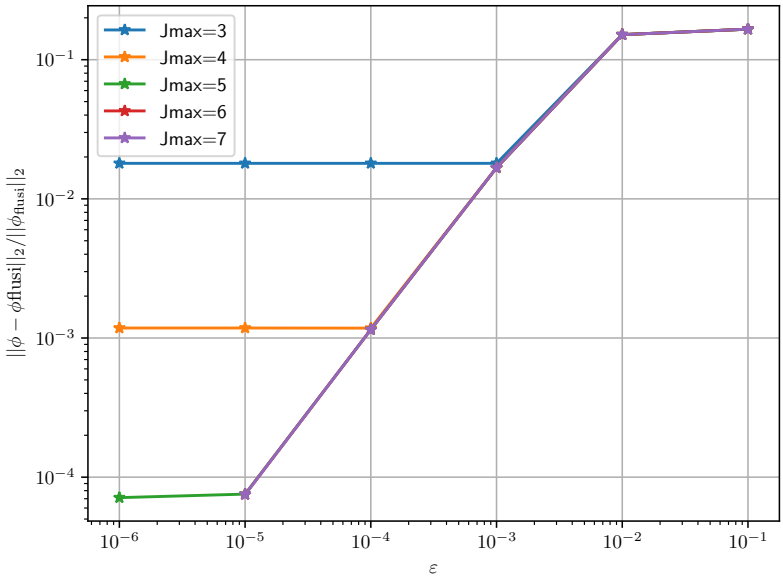


Figure 4.21: Pressure blob test in 3D. Error as a function of the multiresolution threshold ε. The maximum level J is a parameter of the curves. If ε is decreased below 10^{-2}, the expected linear behavior is found, i.e. the error is directly proportional to ε. The error saturates below a threshold $\varepsilon_0(J_{max})$, which is when the discretization error dominates the compression error.

where $\beta = 0.01$. For the performance tests, we vary only the block size B_s, and keep all other parameters fixed. This implies that the resolution Δx depends on the block size. At this point, however, we are interested in the computational performance rather than the solution itself, which justifies that choice.

The tests are performed on the production machine we use, the IBM BlueGene/Q of IDRIS, situated in Orsay, France[6]. We set `bg_size=64` and use 16 ranks per node, thus 1024 threads are executed. The memory of 1024 GB is allocated in the beginning of each simulation[7]. If the number of blocks is

[6]http://www.idris.fr/turing/

[7]Due to various small buffers and some approximations in the memory computations,

too small, notably smaller than the number of mpiranks, then `wabbit` cannot perform efficiently. Hence, for the performance testing, we design the test case such that enough blocks exist, by setting $\varepsilon = 10^{-5}$ and $J_{max} = 6$. It is sufficient to compute $n = 30$ time steps for the purpose of performance testing.

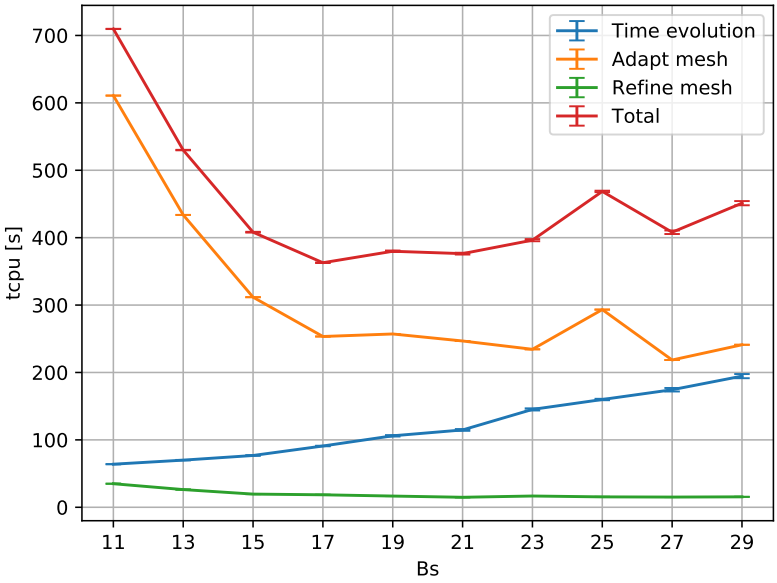


Figure 4.22: Influence of B_s on the performance of an adaptive simulation of a pressure blob. CPU times are shown as average (over all mpiranks) and standard deviations (errorbars, small in magnitude). Computed on 1024 CPUs.

Fig. 4.22 shows the contributions to the total cpu time, split into refinement, time evolution and adaptation (coarsening). These are the three major parts in the code. CPU timing is averaged among all mpiranks. All contributions are shown as a function of the block size B_s. The standard deviation of the CPU time is always small, indicating a good load balancing in this case. All mpiranks perform approximately the same number of operations. The general trend is that the grid adaptation becomes cheaper with increasing

`wabbit` is told to allocate 900.0GB of memory

B_s, while the time evolution (the actual right hand side evaluations) become more expensive. The grid refinement is only weakly dependent on B_s and furthermore negligible in this example.

It should be noted that Fig. 4.22 is valid only for this particular right hand side and physics module. For simulations where more expensive right hand sides are evaluated, the graph can look different (cf Fig. 5.2 for the case of an insect).

The trend is however a U-shape: for small blocks, the sparsity is high, which renders the right hand side cheaper. At this point, however, the overhead of adaptivity is high, such that the total cost is higher. With increasing B_s, we sacrifice the sparsity more and more, which renders adaptivity cheaper but the right hand side more expensive. Fig. 4.23 illustrates the number of blocks and the total number of grid points. For this example of a pressure blob, a block size in the range 17–23 appears to be a good choice.

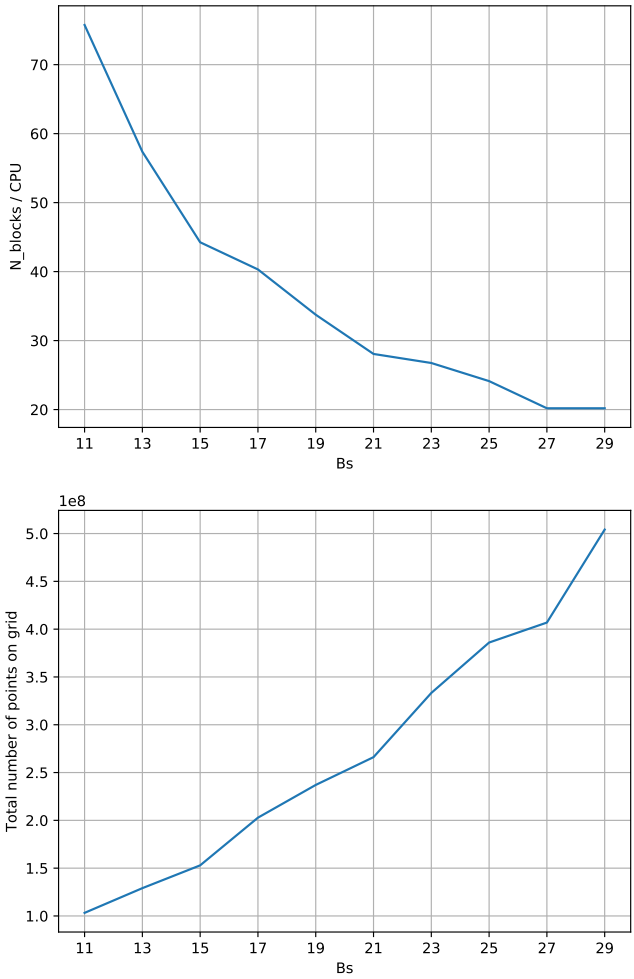


Figure 4.23: Top: Influence of B_s on the number of blocks per cpu N_b used in the simulation of the pressure blob. With increasing B_s, the number of blocks decreases. Bottom: total number of points on the grid, $N_{tot} = N_b \cdot B_s^3$. The sparsity reduces with increasing B_s, hence the total number of points increases. The largest run contained 500 million grid points.

5 Application to insects

In the final section of this work, we consider the main application, which is flapping insect flight. Fig. 5.1 visualizes the wingbeat of the model insect.

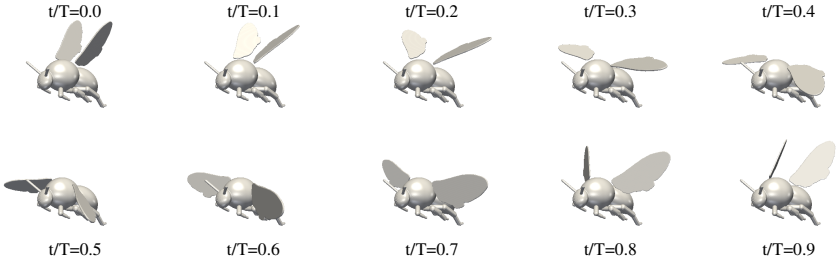


Figure 5.1: Visualization of the wingbeat.

5.1 Performance considerations

Prior to performing full scale computations, we check the performance of the code. We consider the insect test case (bumblebee *bombus terrestris*) studied in [15, 18]. For the performance tests, we first vary only the block size B_s, and keep all other parameters fixed. This implies that the resolution Δx depends on the block size. At this point, however, we are interested in the computational performance rather than the solution itself, which justifies that choice.

The parameters are chosen such that the 1024 CPU we use have a sufficient number of blocks on which they evaluate the right had side (ranging from $N_b = 21960$ for $B_s = 11$ to $N_b = 17256$ for $B_s = 35$). The number of blocks N_b weakly depends on B_s, hence the parallel efficiency is approximately constant. In particular, no CPU idles in any of the simulations. Fig. 5.2 shows the CPU time fractions for time evolution, refinement and coarsening as a function of the block size B_s. The overall picture is very different for the one discussed in Fig. 4.22, as the CPU time monotonically increases with B_s. The reason for this behavior is the expensive mask function χ which encodes the insect. It has to be constructed at every evaluation of the right hand side, hence four times per time step. Unfortunately, it is not possible to construct the mask for the insects body only once, as is done in flusi (for simulations

where the insect is tethered). Rather, even if the body is not time-dependent, our grid is, and hence there is no easy way around re-constructing the mask at every substep. As a direction for future improvement, one can try to construct the mask for the body at least only once per time step, since the grid is not altered *during* the time step.

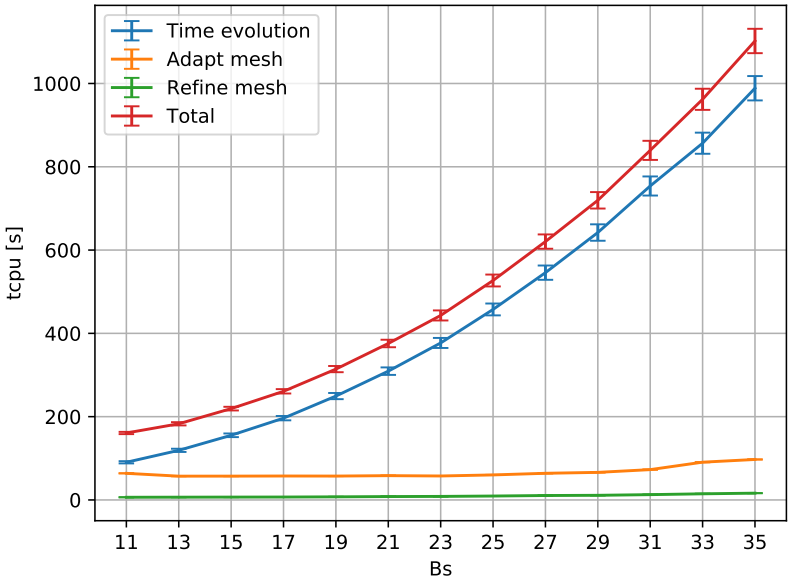


Figure 5.2: Insect performance test. We performed $n = 25$ time steps on an adaptive grid ($L = 8$, $J_{max} = 7$, $\varepsilon = 10^{-3}$) with the artificial compressibility method. Shown is the CPU time in seconds for the three major tasks of the simulation as a function of the block size B_s. Note that the mask generation for the insect considerably contributes to the overall cost.

As shown in Fig. 5.3, the CPU could be reduced by simply omitting the body entirely, which reduces the CPU time requirement by a factor of about 1.7. This, however, is not what we want to do.

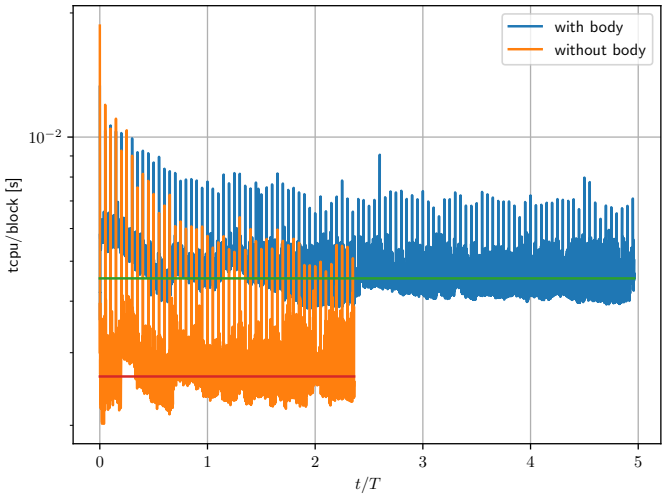


Figure 5.3: CPU time per block for a complete bumblebee model including the body and for a model that contains only the wings. Shown is each time step. Peaks correspond to HDD I/O, solid lines indicate mean over the data. The computational cost is about 1.73 fold higher in the case with body.

5.2 Choice of parameters

Our choice of parameters is determined by the parameters from the reference computation [15]. The key parameters are summarized in Table 2. In `wabbit`, we can set a much larger domain size then in flusi at little additional cost. We study only one value of ε due to time limitations, and use two values for J_{max}, a coarse one (7) and a finer one (8). The latter simulation is not finished by the time of writing. All runs presented in the section are obtained using 1024 CPU cores on TURING, the IBM BlueGene/Q machine in Paris, with a memory allocation of 900 GB per run.

5.3 Results

The present results are the first adaptive simulations of flapping insects. Therefore, we should point out that those are preliminary and are rather to

Parameter	`Flusi`	`Wabbit`
Domain size L	$4 \times 4 \times 6$	$16 \times 16 \times 16$
Resolution	$768 \times 768 \times 1152$	max. 4096^3
Block size B_s	–	17
Max. level J_{max}	–	7 and 8
threshold ε	–	10^{-2}
viscosity ν	$5.92 \cdot 10^{-4}$	$5.92 \cdot 10^{-4}$
speed of sound c_0	incompressible	30.0
inflow velocity	1.246	1.246
penalization C_η	$2.5 \cdot 10^{-4}$	variable

Table 2: Parameters of `flusi` and `wabbit` simulations. All quantities are dimensionless.

be seen as intermediate steps. As those simulations are completely new, we did not know what to expect from the data. Fig. 5.4 shows how the number of blocks evolves over time. For the coarser simulations with $J_{max} = 7$, the complete data for five strokes is available for the four values of C_η we studied. Their behavior is qualitatively similar: for early times, $t < 1$, no wake yet exists, hence the number of blocks is primarily determined by the insect mask itself. For later times, the wake is constituted, and more blocks are required. Towards the end of the computation, $t > 3$, the 'steady state' is reached, i.e. the wake is fully formed and viscosity inhibits further growth. In other words, the production/dissipation balance of turbulence is reached. The higher resolution computation with $J_{max} = 8$ results in a more than 2-fold increase in the number of blocks. As the time step is, due to the CFL condition, divided by two, this simulation is more expensive than the low resolution ones.

The computational time is mostly consumed in the right hand side (57%), followed by the coarsening (39%) and the refine everywhere (3%). The overhead of adaptivity is hence still significant, but the ratio of right hand side and grid routines is already quite good.

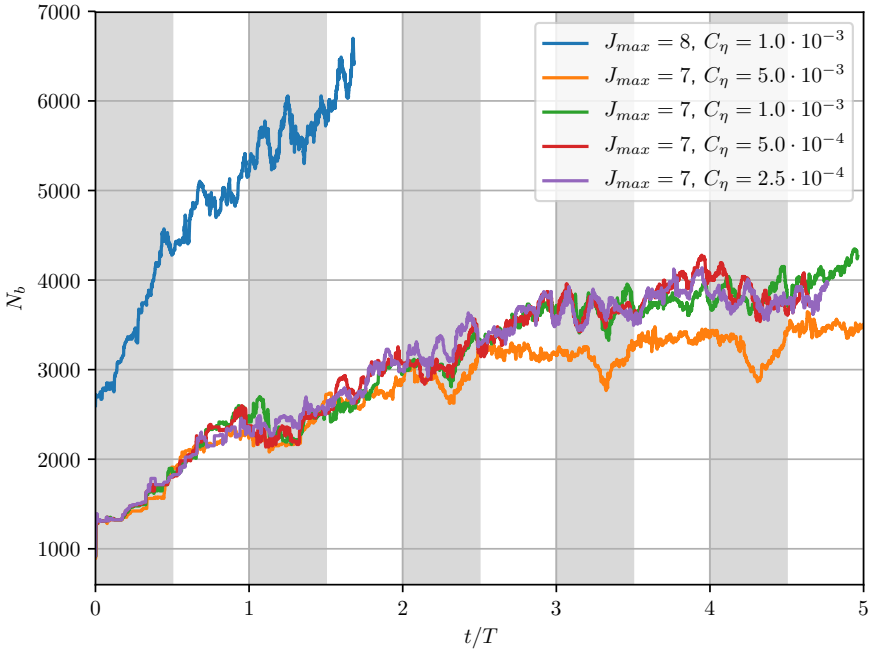


Figure 5.4: Adaptive insect computations, number of active blocks N_b as a function of time. The value N_b is the number of blocks after grid adaptation, hence the right hand side is evaluated on $8N_b$ blocks. Stroke duration $T = 1$, gray shaded times are downstrokes.

Fig. 5.5 shows the forces acting on the entire insect. They are obtained using the conventional volume integration of the penalization term, $F = \int \chi(u - u_s)/C_\eta$. Unsteady corrections are neglected [15, 18]. The forces do not agree well with the results obtained with the spectral code (black dashed line), which, while not representing the 'true' solution neither, can be supposed to be much more trustworthy than present preliminary results.

We first discuss the coarser simulations with $J_{max} = 7$. While the qualitative behavior seems to agree (location of peaks etc), especially the influence of the penalization parameter is alarming. For the thrust force computed with $J_{max} = 7$, for instance, the peak force for the downstroke (areas) monotonically increases with decreasing C_η. Hence, no convergence to the flusi solution can be observed, rather, the smaller C_η, the worse the solution at

those times. At the reversal (gray/white interface) the peak is best hit by the $C_\eta = 1.0 \cdot 10^{-3}$ solution (green). During upstrokes (white), the curves agree well during the part when the force is negative, and for the subsequent peak the same behavior as for the peak during the downstroke is observed. Again, the half-stroke averaged value shows no convergence to the flusi solution.

For the lift force, the picture is again different, this time the $C_\eta = 5.0 \cdot 10^{-4}$ solution (red) is the best. Its stroke averaged values agree more or less with flusi, but again during the upstroke (white) the agreement is poor. Thus, from this data, we cannot choose the 'best' C_η. We can just say that $C_\eta = 5.0 \cdot 10^{-3}$ (orange) appears to be too large, as very little lift is produced.

Increasing the resolution to $J_{max} = 8$ (blue) generally improved the picture. Note we can, unfortunately, not perform the four runs for this resolution due to time limitations. Especially the lift during the upstroke is improved. In general, the lift force for $C_\eta = 1.0 \cdot 10^{-3}$ is similar for both $J_{max} = 8$ (blue) and 7 (green). Note this is not the case for the thrust, where the blue line is closest to the orange line.

Just from the force data it is hence not possible to decide which C_η is the best choice. By the time of writing, we do not have any alternative means to compute the forces (several ways are possible, see e.g. [2]) or another quantity suitable for quantitative validation. The fact that the forces are off does not necessarily nullify the results, as it may, like in the 2D cylinder at $Re = 40$ studied previously, ust be the force computation itself which is faulty.

Figs. 5.6 and 5.7 show isosurfaces of vorticity magnitude in side- and front view. Contrarily to the force computations, one visually establishes a convergence for the vorticity field. It appears that $C_\eta = 5 \cdot 10^{-3}$ is indeed significantly too high and that the wake in this case lacks many fine scaled structures present in the other cases. The remaining three cases are more similar to each other, but in the front view a structure above the insect can be identified which is intensified for decreasing C_η.

Fig. 5.8 shows the direct comparison with results from the literature for $J_{max} = 7$, $C_\eta = 5 \cdot 10^{-4}$ for the same isosurface of vorticity magnitude. The `wabbit` computation lacks many details of the vorticity field. From this fig-

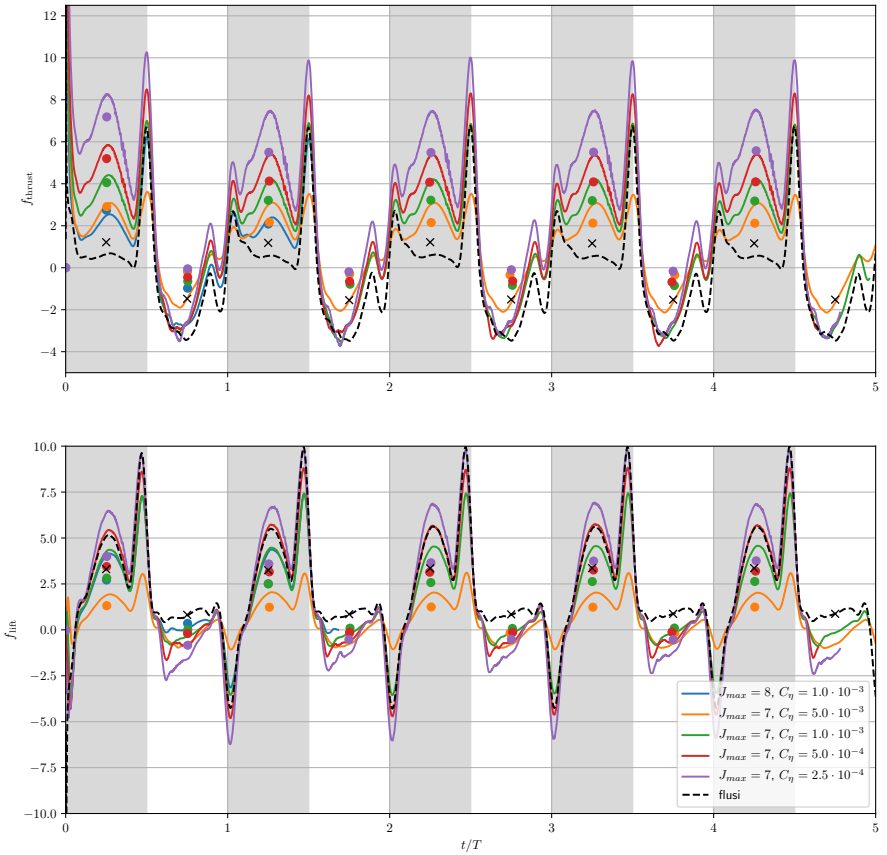


Figure 5.5: Adaptive insect computations, forces obtained from volume integration of the penalization term. Thrust (top) and lift force (bottom). The sideways force is zero due to lateral symmetry. Black dashed line is the spectral code reference. Colored dots represent averages over the half strokes (i.e. the white and gray shaded areas). The high resolution computation is not finished by the time of writing.

ure, it seems the resolution is not high enough, but at least some principal features are preserved. Increasing the resolution to $J_{max} = 8$ (same figure) recovery many details. We conclude that the $J_{max} = 7$ computations are not yet well enough resolved.

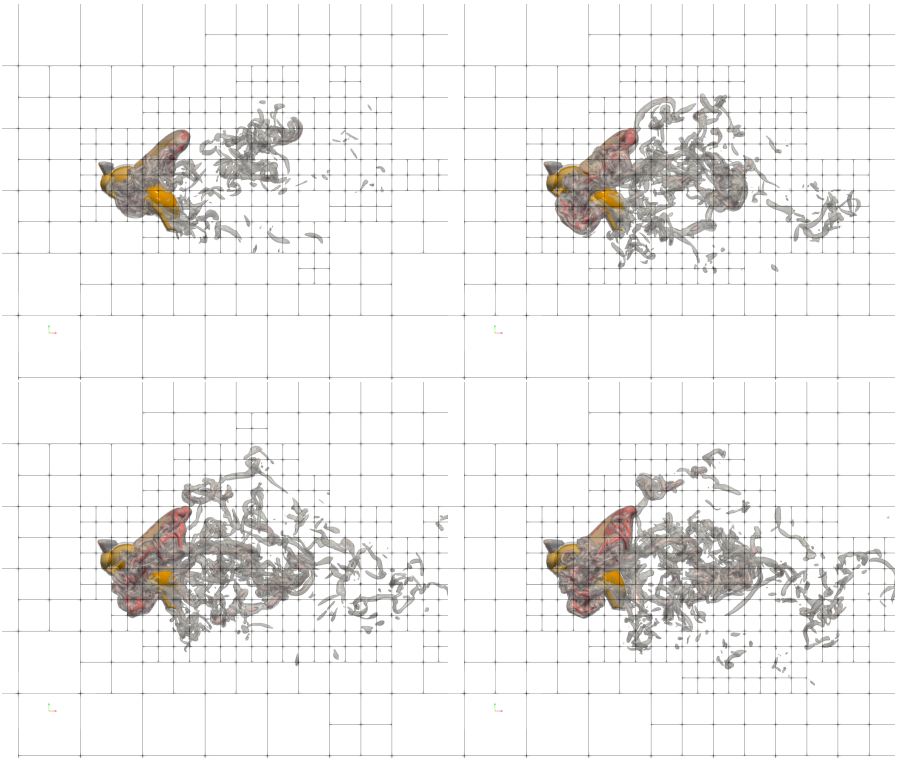


Figure 5.6: Adaptive insect computations, isosurfaces of vorticity magnitude $|\omega| = 25, 50$ *and* 100. *Shown is the side view at time* $t/T = 3.95$ *for* $C_\eta = 5 \cdot 10^{-3}, 10^{-3}, 5 \cdot 10^{-4}$ *and* $2.5 \cdot 10^{-4}$ *(from top left to bottom right).*

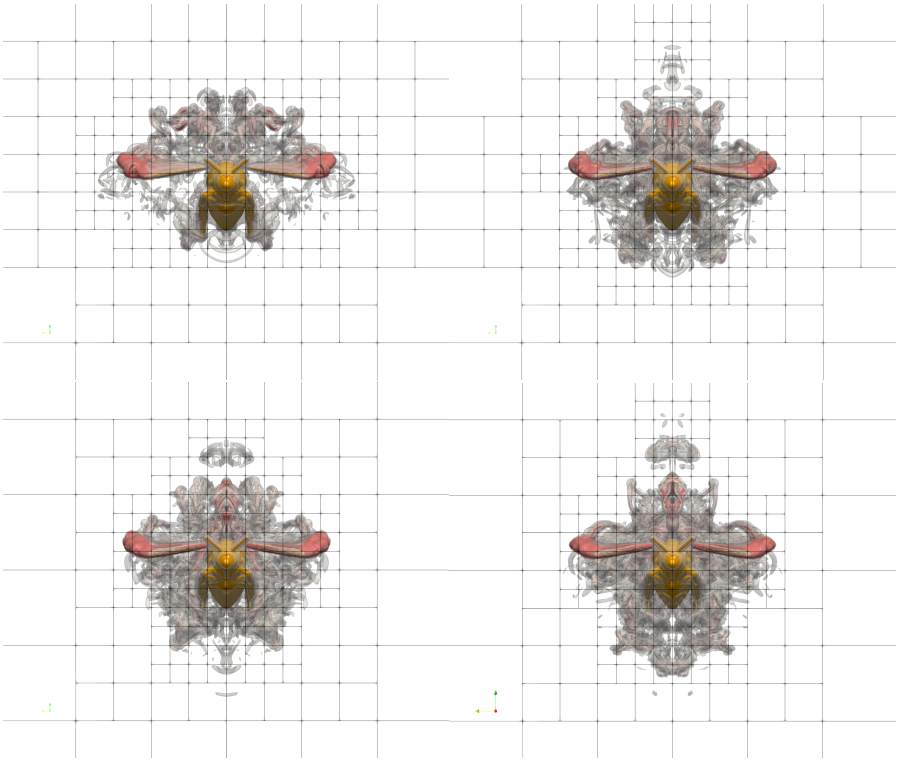


Figure 5.7: Adaptive insect computations, isosurfaces of vorticity magnitude $|\omega| = 25, 50$ *and* 100*. Shown is the frontal view at time* $t/T = 3.40$ *for* $C_\eta = 5 \cdot 10^{-3}$, 10^{-3}, $5 \cdot 10^{-4}$ *and* $2.5 \cdot 10^{-4}$ *(from top left to bottom right).*

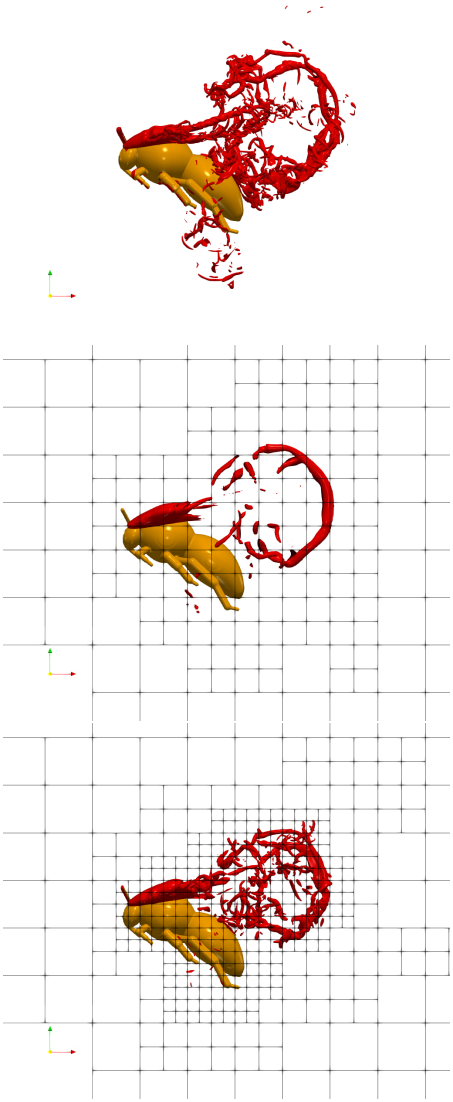


Figure 5.8: Adaptive insect computations, direct comparison with the flusi code (top). Shown is the same isosurface $|\omega| = 100$. The two `wabbit` *cases are $J_{max} = 7$, $C_\eta = 5 \cdot 10^{-4}$ (middle) and $J_{max} = 8$, $C_\eta = 1 \cdot 10^{-3}$ (bottom). The coarser simulation lacks many details present in the spectral solution, while the finer one recovers them to some extent. The finer run is not yet finalized by the time of writing, hence the comparison is at $t/T = 1.4$.*

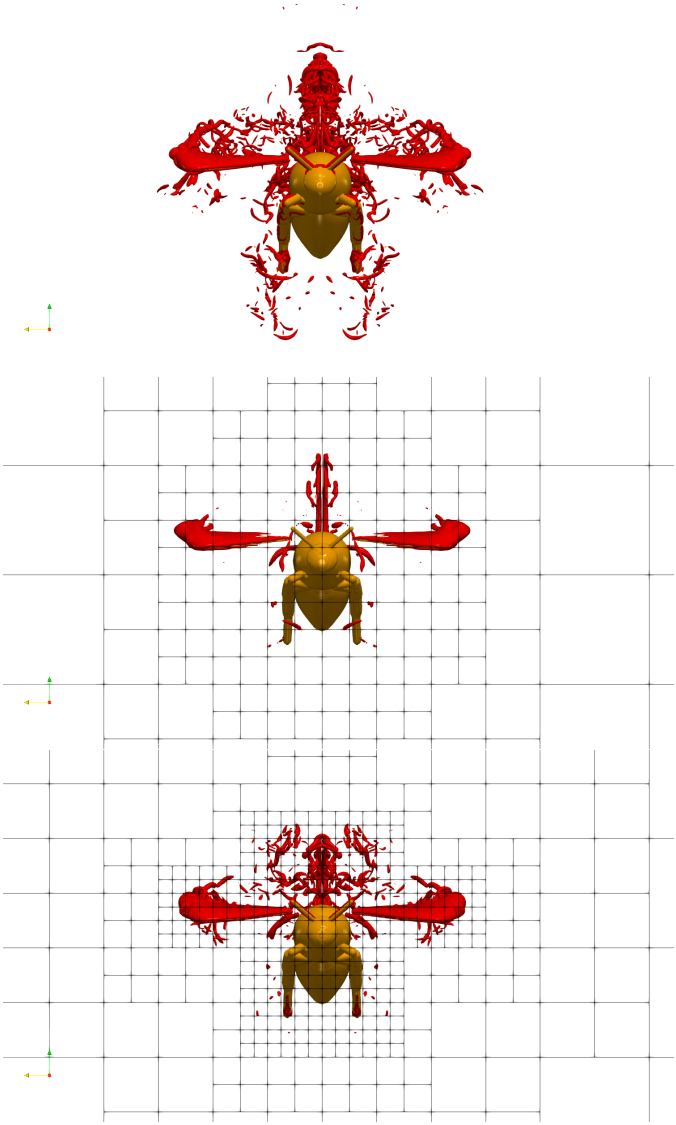


Figure 5.9: Adaptive insect computations, direct comparison with the flusi code (top). Shown is the same isosurface $|\omega| = 100$. The two `wabbit` *cases are $J_{max} = 7$, $C_\eta = 5 \cdot 10^{-4}$ (middle) and $J_{max} = 8$, $C_\eta = 1 \cdot 10^{-3}$ (bottom). The coarser simulation lacks many details present in the spectral solution, while the finer one recovers them to some extent. The finer run is not yet finalized by the time of writing, hence the comparison is at $t/T = 1.4$.*

6 Conclusions

During this work, the following milestones have been reached

1. The new ghost node synchronization module of the adaptive code `wabbit` is implemented and tested
2. The code to generate the mask function for an insect is imported from the `flusi` code and adapted to be suitable for `wabbit`
3. Extensive validation tests have been performed to verify the different parts of the code. They reveal some problems with the penalization method which must be resolved in future work. In particular the force computation must be carefully verified in future work using different methods.
4. The first adaptive simulations of a flapping insect have been performed. Those results are preliminary but promising and demonstrate the feasibility. For future work, the code's performance must be improved in order to allow higher J_{max} for the simulations in a reasonable time frame. Then, even more detailed simulations are to be performed. By the time of writing, we also perform simulations of a fruit fly at $Re = 150$, which will allow a larger variation of parameters, especially the penalization parameter.

References

[1] P. Angot, C. Bruneau, and P. Fabrie. A penalization method to take into account obstacles in incompressible viscous flows. *Numer. Math.*, 81:497–520, 1999.

[2] M. Bergmann and A. Iollo. Modeling and simulation of fish-like swimming. *J. Comput. Phys.*, 230:329–348, 2011.

[3] A. Brandt. Multilevel adaptive solutions to boundary value problems. *Math. Comp.*, 31:333–390, 1977.

[4] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods in Fluid Dynamics*. Springer Verlag, 1986.

[5] G. Carbou and P. Fabrie. Boundary layer for a penalization method for viscous incompressible flow. *Adv. Diff. Equ.*, 8:1453–2480, 2003.

[6] A.J. Chorin. A numerical method for solving incompressible viscous flow problems. *J. Comput. Phys.*, 2:12–26, 1967.

[7] A. Cohen. *Wavelet methods in numerical analysis*, volume VII of *Handbook of Numerical Analysis*. Elsevier, Amsterdam, 2000.

[8] A. Cohen, W. Dahmen, and R. DeVore. Adaptive wavelet methods for elliptic operator equations – convergence rates. *Math. Comp*, 70:27–75, 1998.

[9] A. Cohen, S. M. Kaber, S. Müller, and M. Postel. Fully adaptive multiresolution finite volume schemes for conservation laws. *Math. Comp.*, 72:183–225, 2003.

[10] R. A. DeVore. Nonlinear approximation. *Acta Numerica*, 7:51–150, 1998.

[11] M. Domingues, S. Gomes, O. Roussel, and K. Schneider. An adaptive multiresolution scheme with local time stepping for evolutionary PDEs. *J. Comput. Phys.*, 227:3758–3780, 2008.

[12] M. Domingues, S. Gomes, O. Roussel, and K. Schneider. Space-time adaptive multiresolution methods for hyperbolic conservation

laws: Applications to compressible euler equations. *Appl. Num. Math.*, 59:2303–2321, 2009.

[13] M. Domingues, S. Gomes, O. Roussel, and K. Schneider. Adaptive multiresolution methods. *ESAIM: Proceedings*, 34:1–96, 2011.

[14] M. Domingues, S. M. Gomes, and L.M.A. Díaz. Adaptive wavelet representation and differentiation on block-structured grids. *Appl. Num. Math.*, 47:421–437, 2003.

[15] T. Engels, D. Kolomenskiy, K. Schneider, F.-O. Lehmann, and J. Sesterhenn. Bumblebee flight in heavy turbulence. *Phys. Rev. Lett.*, 116:028103, 2016.

[16] T. Engels, D. Kolomenskiy, K. Schneider, and J. Sesterhenn. Two-dimensional simulation of the fluttering instability using a pseudospectral method with volume penalization. *Computers & Structures*, 122:101–112, 2012.

[17] T. Engels, D. Kolomenskiy, K. Schneider, and J. Sesterhenn. Numerical simulation of fluid-structure interaction with the volume penalization method. *J. Comput. Phys.*, 281:96–115, 2015.

[18] T. Engels, D. Kolomenskiy, K. Schneider, and J. Sesterhenn. FluSI: A novel parallel simulation tool for flapping insect flight using a Fourier method with volume penalization. *SIAM J. Sci. Comput.*, 38(5):S3–S24, 2016.

[19] T. Engels, D. Kolomenskiy, K. Schneider, and J.L. Sesterhenn. A numerical study of vortex-induced drag of elastic swimmer models. In *Proceedings 6th International Symposium on Aero-aqua Bio-Mechanisms, November 13.-16., Honolulu, Hawaii, USA.*, 2014.

[20] M. Farge, K. Schneider, and N. Kevlahan. Non-gaussianity and coherent vortex simulation for two-dimensional turbulence using an adaptive orthonormal wavelet basis. *Phys. Fluids*, 11:2187–2201, 1999.

[21] R. Gautier, D. Biau, and E. Lamballais. A reference solution of the flow over a circular cylinder at re = 40. *Computers & Fluids*, 75:103–111, 2013.

[22] B. Gottschlich-Müller and S. Müller. Adaptive finite volume schemes for conservation laws based on local multiresolution techniques. In M. Fey, editor, *Hyperbolic problems: Theory, numerics, applications*, volume 1 of *Int. Ser. Numer. Math*, pages 385–394. Birkhauser, Basel, 1999.

[23] A. Grossmann and J. Morlet. Decomposition of hardy functions into square integrable wavelets of constant shape. *SIAM J. Math. Anal.*, 15:723–736, 1984.

[24] A. Harten. Multiresolution algorithms for the numerical solution of hyperbolic conservation laws. *Comm. Pure Appl. Math.*, 48:1305–1342, 1995.

[25] M. Kaibara and S. M. Gomes. A fully adaptive multiresolution scheme for shock computations. In E.F. Toro, editor, *Godunov Methods: Theory and Applications*. Klumer Academic/Plenum Publishers., 2000.

[26] N. Kevlahan and M. Farge. Vorticity filaments in two-dimensional turbulence: Creation, stability and effect. *J. Fluid Mech.*, 346:49–76, 1997.

[27] S. Mallat. Multiresolution approximation and orthonormal wavelet basis of l2 (r). *Trans. Am. Math. Soc.*, 315:69–87, 1989.

[28] S. Müller. *Adaptive multiscale schemes for conservation laws*, volume 27 of *Lectures Notes in Computational Science and Engineering*. Springer, Heidelberg, 2003.

[29] S. Müller and Y. Stiriba. Fully adaptive multiscale schemes for conservation laws employing locally varying time stepping. *J. Sci. Comput.*, 30:493–531, 2007.

[30] T. Ohwada and P. Asinari. Artificial compressibility method revisited: Asymptotic numerical method for incompressible Navier–Stokes equations. *J. Comp. Phys.*, 229:1698–1723, 2010.

[31] R. Peyret. *Spectral Methods for Incompressible Viscous Flow*. Applied Mathematical Sciences 148. Springer, 2001.

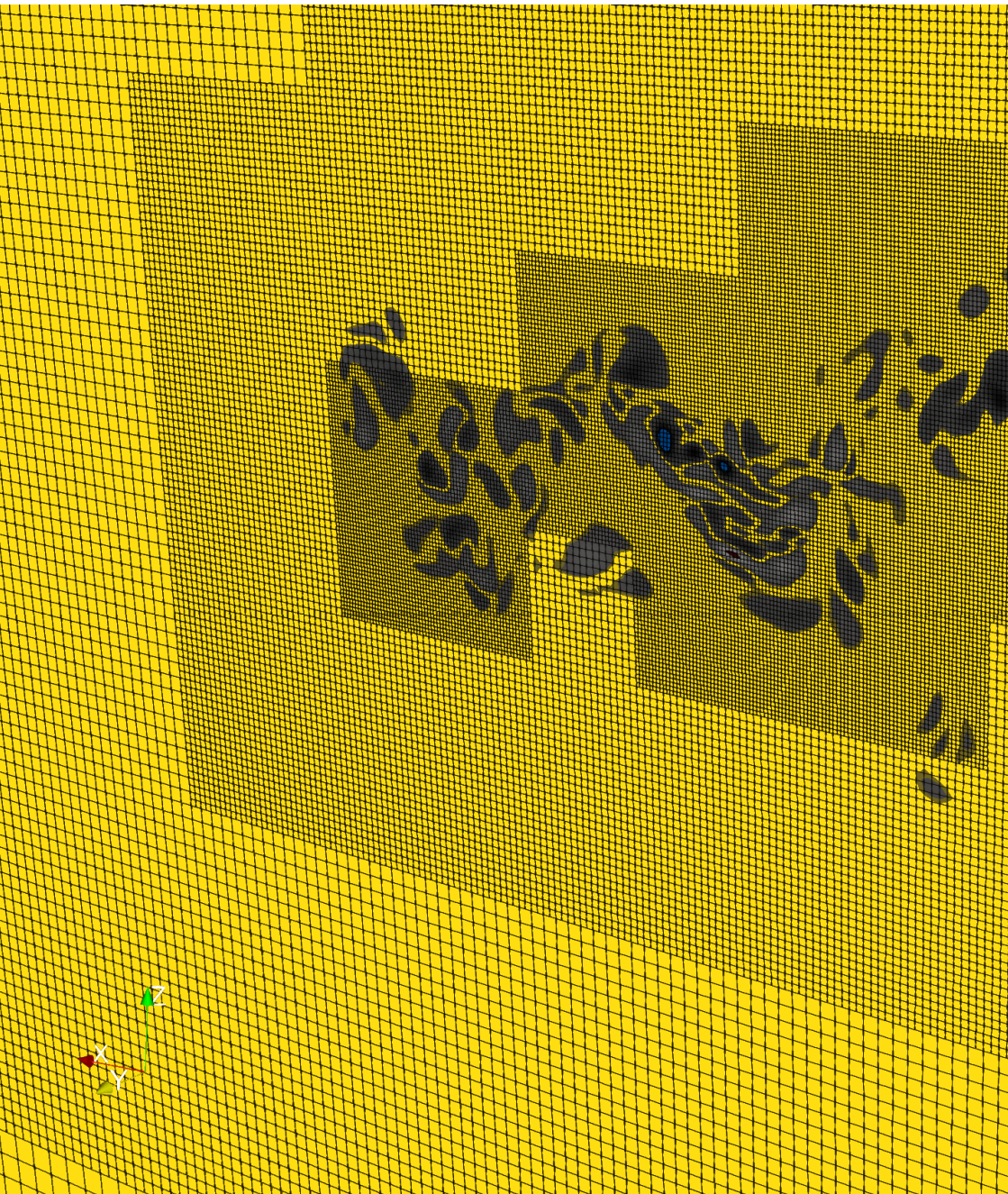
[32] O. Roussel and K. Schneider. Coherent vortex simulation of weakly compressible turbulent mixing layers using adaptive multiresolution methods. *J. Comput. Phys.*, 229:2267–2286, 2010.

[33] O. Roussel, K. Schneider, A. Tsigulin, and H. Bockhorn. A conservative fully adaptive multiresolution algorithm for parabolic pdes. *J. Comput. Phys.*, 188:493–523, 2003.

[34] K. Schneider and O. Vasilyev. Wavelet methods in computational fluid dynamics. *Annu. Rev. Fluid Mech.*, 42:473–503, 2010.

[35] Christopher K.W. Tam and Jay C. Webb. Dispersion-relation-preserving finite difference schemes for computational acoustics. *Journal of Computational Physics*, 107(2):262–281, 1993.

[36] G. Zumbusch. *Parallel multilevel methods: adaptive mesh refinement and loadbalancing.* Advances in Numerical Mathematics. Springer, 2003.


Z
X
Y