Computers and Fluids 186 (2019) 58–73

Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

An explicit gas kinetic scheme algorithm on non-uniform Cartesian meshes for GPGPU architectures

Stephan Lenz*, Martin Geier, Manfred Krafczyk

Institute for Computational Modeling in Civil Engineering, TU Braunschweig, Pockelsstr. 3, Braunschweig 38106, Germany

ARTICLE INFO

Article history:
Received 28 January 2019
Revised 8 April 2019
Accepted 17 April 2019
Available online 18 April 2019

Keywords:
Gas kinetic scheme
Non-uniform mesh
Nested time stepping
GPGPUs

ABSTRACT

We present an efficient algorithm of a two-dimensional Gas Kinetic Scheme (GKS) suitable for General Purpose Graphics Processing Units (GPGPUs). The algorithm features conservative second order quadtree-type refinement for Cartesian meshes and nested time stepping. The implementation is validated for several incompressible and thermal compressible test cases in the low Mach (Ma) number regime. We find that heat transfer for high Rayleigh (Ra) number and turbulent convection can be simulated correctly. Finally, we demonstrate a simulation of a Rayleigh-Bénard setup with $Ra = 10^{13}$ that qualitatively resembles the conditions of fire on a room scale. The implementation delivers a performance of more than one billion cell updates per second.

1. Introduction

The kinetic approach to fluid dynamics, based on the Boltzmann transport equation, employs a unified flux of the momentum distribution whereas the Navier–Stokes equation uses three different transport terms based on pressure gradient, diffusion and advection, respectively. While in the Navier–Stokes equation, only the diffusion and pressure related flux are linear, the momentum flux in the Boltzmann equation is completely linear. This, however, comes at the price of a higher dimensionality of the momentum distribution function exceeding the degrees of freedom related to the macroscopic quantities of the Navier–Stokes equation. It is not a priori clear which approach offers higher modeling accuracy and simulation efficiency (after discretization) for macroscopic flow regimes. Significant progress is evidently made on both sides. An efficient numerical method based on the Boltzmann transport equation but aiming at the same fluid dynamic regime as the Navier–Stokes equation is the Lattice Boltzmann Method (LBM). However, this method is based on a low order approximation of the Boltzmann equation and it is difficult to include energy conservation while sustaining its computational efficiency. Some applications, like the simulation of fire, require compressible fluid solvers which can deal with high temperature and density ratios even at low Mach numbers. Such problems are difficult to solve with the classical lattice Boltzmann method due to the restrictive discretization of the velocity space that allows only for constant temperatures or a small range of temperatures without inflating the number of discrete velocities. Large temperature differences cause turbulent natural convection, such that high efficiency is required, to obtain accurate results in the framework of Large Eddy Simulation (LES) or Direct Numerical Simulation (DNS). A method that offers more physical flexibility than the lattice Boltzmann method with regard to the aforementioned problem class is the Gas Kinetic Scheme (GKS). It inherits the linear advection and the unified momentum flux from the gas kinetic equation while it reduces the dimensionality of the variable space to the macroscopic quantities. The current paper proposes an efficient implementation of this method on General Graphics Processing Units (GPGPUs).

One of the open questions in computational fluid dynamics (CFD) is the choice of the optimal meshing strategy. Among the many possibilities, unstructured and Cartesian mesh lay on opposing ends of the spectrum concerning flexibility, geometrical conformity and simplicity. One of the main advantages of Cartesian grids is their trivial generation that requires little human intervention. In industrial applications of CFD, it is a general observation that the most time consuming and laborious task is the preparation of the mesh. The actual fluid dynamic computation usually requires only a small fraction of the wall-clock time needed for the numerical solution of the spatio-temporal problem. In contrast, academic publications usually mention only the time needed for the actual computation, ignoring the pre-processing time as outside of the scope of their analysis. Another advantage of Cartesian methods is that no data on cell shapes have to be stored. In unstructured meshes shape data usually requires an amount of memory

* Corresponding author.
E-mail address: lenz@irmb.tu-bs.de (S. Lenz).
URL: http://www.tu-bs.de/irmb.com (S. Lenz)

https://doi.org/10.1016/j.compfluid.2019.04.011
0045-7930/

comparable to the state vector whereas the memory consumption in Cartesian methods is clearly dominated by storage for the primary variables. This also implies an advantage concerning data traffic during parallel computations based on the Cartesian meshes. Such methods might also excel in terms of accuracy, in particular if the mesh is uniform (non-stretched) as symmetric finite difference operators are usually one asymptotic order more accurate than their non-symmetric counterparts.

On the other hand, Cartesian grid methods are often described as inflexible and inefficient. An obvious disadvantage of Cartesian grids is their non-conformity to arbitrary boundaries (i.e. inclined or curved). Cartesian grids can be extremely inefficient when stored in a matrix format of which most cells are void when the bounding box of the domain is much larger than the flow volume. (On the contrary, such a discretization is very efficient for domains of box shape.) Also, Cartesian meshes do not allow for a continuous change in resolution in arbitrary directions.

Both the advantages and disadvantages of Cartesian meshes over unstructured meshes are substantial and different scholars draw different conclusions. In order for Cartesian meshes to be competitive to unstructured meshes at least three extensions have to be taken into account:

- Non-conforming boundaries have to be treated by special methods such as immersed boundary approaches [1], boundary interpolation methods [2] or cut cell approaches with special treatment of the boundary cells [3].
- The mesh points or cells have to be stored using indirect addressing between cells or blocks of cells in order to support complex shaped domains.
- For local mesh refinement and adaptivity, meshes of different resolution have to be coupled leading to global or local time-stepping schemes.

Implementing these extensions for Cartesian grids eliminates most of their disadvantages while keeping most of their advantages. In this paper we focus on the last two options, leaving non-conforming boundary treatment to future work. In particular, we focus on a massively parallel implementation of a finite volume discretization of the GKS.

Today's increase in computational performance in scientific computing is mostly due to massive parallelization. Global operations are difficult to parallelize and cannot utilize the potential parallel power of thousands and tens of thousands of cores. Schemes with explicit time marching and local spatial support, on the other hand, are inherently parallel. Such schemes scale very well on massive parallel hardware. In fluid dynamics the Lattice Boltzmann Method (LBM) is an outstanding example for an explicit scheme with minimal spatial support which shows very high performance on different massive parallel architectures such as GPUs [4] and distributed CPU clusters with more than a trillion grid nodes [5,6]. Grid refinement in LBM is implemented by connecting grids of different resolution levels [7,8] and complex boundaries are implemented by a cut cell approach [9,10]. This approach allows an efficient and automatic grid generation with billions of cells and very complex geometries can be discretized without human intervention [6,11].

When it comes to compressible flows, LBM requires additional modeling for the consistent treatment of the energy equation (e.g. [12]). The gas kinetic scheme [13,14] which emerged in the last two decades, naturally includes a consistent approach for fully compressible flows. Kinetic theory is used to derive finite volume schemes, most often with very good shock capturing capabilities. Most applications of GKS are found in the areas of aerodynamics [15,16] and rarefied flows [17,18]. For low Mach number flows the shock capturing can be omitted [19–21]. The result is a straightforward derivation of an explicit local scheme for compressible fluid dynamics. Low Mach number compressibility is relevant for thermal flow problems including large temperature differences. Applications for such flows are the simulation of compressible natural convection [22] and especially fire induced flows.

As a finite volume method, GKS can be implemented on arbitrarily unstructured meshes [23,24]. Chen et al. presented a Cartesian grid implementation of GKS on uniform meshes [2]. Changes in resolution on Cartesian meshes can be implemented by quadtrees in 2D or octrees in 3D, such that each refinement level is a uniform Cartesian mesh. The meshes of different resolution can be coupled, to obtain a continuous fluid domain. This coupling strongly depends on the requirements of the solution scheme, for examples see [25–28]. Quadtree-type refinement for a GKS was already demonstrated by Yuan et al. [29]. They reconstruct the hydrodynamical variables per cell and interpolate the flow state from fine to coarse in order to obtain coarse level neighbor values. The computation of fine neighbor values on the coarse side of the interface is not described in detail.

The work of Lyu et al. [30] is noteworthy here, because it deals with the problem of gradient computation on cell faces which are located at the coarse to fine interface. They utilize a least square approach for the gradients, which requires a large non-uniform stencil for second order accuracy. Furthermore, the stencil is not universal, but depends strongly on the local mesh and shape of the coarse to fine interface.

In this paper we present a GKS implementation on two dimensional Cartesian meshes with a quadtree-type structure, inspired by non-uniform LBM grids. All non-local operations are performed per cell face with only next neighbor support. We reconstruct the cell face variables and gradients on the coarse to fine interface with the help of interpolated ghost cells, such that we can use uniform centered stencils, which preserve second order accuracy. Also the conservation property of the finite volumes is preserved. In addition, the quadtree-type structure implies a performance enhancement, namely nested time stepping. A local stability condition is satisfied, rather than a global one, such that the advective CFL number with respect to some reference velocity is essentially constant throughout the domain.

Furthermore, we exploit the explicit and local structure of the GKS by implementing it for General-Purpose-Graphics-Processing-Units (GPGPUs) within the CUDA framework [31]. To the best of our knowledge there are only two GPGPU implementations for a GKS variants on uniform meshes have been published so far: the analytical gas kinetic method (AGKM) [32] and a so-called unified GKS for rarefied flows [33]. We show that a very high performance of more than a billion cell updates per second is possible for locally refined Cartesian meshes in two dimensions on state-of-the-art hardware.

In Section 2 we briefly recall relevant gas kinetic theory. Based on that, we summarize the derivation of the GKS for low Mach number flows with gravitational forces on uniform meshes in Section 3. The coupling of different resolutions and the nested time stepping algorithm are presented in Section 4. Section 5 deals with the GPGPU implementation and performance considerations. The present scheme is then analyzed and validated for incompressible and thermal compressible flow problems at low Mach number in Section 6. Finally, the paper is concluded in Section 7.

2. GKS physics in a nut-shell

In classical fluid dynamics the flow state is characterized by macroscopic variables velocity $\vec{U} = (U, V)^T$, pressure p and temperature T. Gas kinetic theory, in contrast, characterizes the flow state by a mesoscopic scalar particle distribution function $f = f(\vec{x}, \vec{u}, \underline{\xi}, t)$. The distribution function is high dimensional as it depends not only on space $\vec{x}$ and time t, but also on the particle

velocity $\vec{u}$ and K internal degrees of freedom $\underline{\xi} = (\xi_1, \dots, \xi_K)^T$. The internal degrees of freedom originate in the molecular structure. In two dimensions mono-atomic gases have $K = 1$ and diatomic molecules (such as air to a large part) have $K = 3$. Two translational degrees of freedom are taken into account explicitly and the translational motion into the third direction appears in the internal degrees of freedom.

The mesoscopic particle distribution function f is connected to the macroscopic flow state via conserved moments by

$$\underline{W} = \int_{-\infty}^{\infty} \underline{\psi} f d\Xi, \tag{1}$$

where $\underline{\psi} = \left(1, u, v, \frac{1}{2}(u^2 + v^2 + {\xi_1}^2 + \dots + {\xi_K}^2)\right)^T$ and $d\Xi = dudvd\xi_1 \dots d\xi_K$. The conserved moments yield conserved hydrodynamic variables $\underline{W} = (\rho, \rho U, \rho V, \rho E)^T$, where ρ is the density and E contains kinetic and internal energy per unit mass.

On the mesoscopic level, the flow evolution is described by the Boltzmann equation. For the derivation of GKS the collision operator is approximated by the BGK relaxation model [34]:

$$\frac{\partial f}{\partial t} + \vec{u} \cdot \nabla_{\vec{x}} f + \vec{g} \cdot \nabla_{\vec{u}} f = -\frac{f - f^{eq}}{\tau} \tag{2}$$

The BGK approximation assumes that the distribution is relaxed towards a Maxwellian equilibrium f^{eq}. The gravitational acceleration is $\vec{g} = (g_x, g_y)^T$.

The macroscopic Navier–Stokes equations can be derived from the Boltzmann equation by Chapman-Enskog expansion [13]. A suitable choice for the relaxation time τ is then given by $\tau = 2\lambda\mu/\rho$, with $\lambda = 1/(2RT)$ and R being the specific gas constant.

3. The finite volume approach for the Gas Kinetic Scheme

In this section we summarize the derivation of the present GKS for two-dimensional low Mach number thermal compressible flows. In order to be self-contained we recall the derivation from Lenz et al. [22], which is based on the work of Xu [13] and Tian et al. [35]. The original GKS of Xu [13] neglects the gravitational contribution $\vec{g} \cdot \nabla_{\vec{u}} f$ in the Boltzmann equation. Tian et al. [35] take this contribution into account, but otherwise follow the same path for the flux computation as the original GKS. The second extension of Tian et al. [35] is the consistent treatment of the source term.

All computations considered in this work are two-dimensional, but the vectorial equations also hold for three dimensions. Hence, an extension to three dimensions is straightforward.

Finite volume methods utilize the conservation form of the Navier–Stokes equation on discrete cells i as control volumes

$$\underline{W}_i^{n+1} = \underline{W}_i^n - \frac{1}{V_i} \sum_j \underline{\mathcal{F}}_j A_j + \underline{\mathcal{Q}}_i. \tag{3}$$

This equation can be derived from the BGK equation (Eq. (2)) by taking the conserved moments (Eq. (1)), averaging over one cell with volume V_i, using the Gauss theorem and integrating over one time step. Then the flux densities $\underline{\mathcal{F}}_j$ over the jth cell face with area A_j and the source term $\underline{\mathcal{Q}}_i$ inherit kinetic definitions based on the time dependent distribution f.

The remaining task is to model f. In the basic GKS considered in this work, f is modeled based on the conserved variables and their spatial and temporal gradients. A formal solution of the BGK equation along particle trajectories is used [13]:

$$f\left(\vec{0}, \vec{u}, \underline{\xi}, t\right) = f_0\left(\delta\vec{x}(t), \vec{u} + \delta\vec{u}(t), \underline{\xi}, 0\right) e^{-\frac{t}{\tau}} + \frac{1}{\tau} \int_0^t f^{eq}\left(\delta\vec{x}(t - t'), \vec{u} + \delta\vec{u}(t - t'), \underline{\xi}, t'\right) e^{-\frac{t - t'}{\tau}} dt'. \tag{4}$$

The time dependent $f(\vec{0}, \vec{u}, \underline{\xi}, t)$ is composed of a decaying initial distribution f_0 and an equilibrium f^{eq} integrated over time. The particle motion is taken into account by tracking the particle state in space $\delta\vec{x}$ [13] and velocity space $\vec{u} + \delta\vec{u}$ [35] back to an initial time. The resulting spatio-temporal variance of f^{eq} is approximated by Taylor expansion. The initial distribution f_0 is constructed by the Taylor expanded f^{eq} at $t = 0$ and a non-equilibrium contribution. The latter is modeled to first order by the Chapman-Enskog expansion, which implies that the hydrodynamic moments of f obey the Navier–Stokes equations. Finally f can be written as

$$f\left(\vec{0}, \vec{u}, \underline{\xi}, t\right) = f_0^{eq}\left(1 - \tau\left(\vec{a} \cdot \vec{u} + \vec{b} \cdot \vec{g}\right) + (t - \tau)A\right), \tag{5}$$

where $\vec{a} = \underline{\vec{a}} \cdot \underline{\psi} = \nabla \ln f^{eq}$ and $A = \underline{A} \cdot \underline{\psi} = \partial_t \ln f^{eq}$ are the spatial and temporal expansion coefficients of the Taylor expansion, respectively. The coefficient $\vec{b} = \nabla_{\vec{u}} \ln f^{eq}$ relates to expansion in particle velocity space. For details, how to compute these coefficients see Xu [13] and Lenz et al. [22].

Flux densities are obtained by

$$\underline{\mathcal{F}}_j = \int_{t_n}^{t_{n+1}} \int_{-\infty}^{\infty} u \underline{\psi} f d\Xi dt. \tag{6}$$

Since f depends explicitly on time, the time integration can be solved analytically. In addition, f is formulated in terms of the known initial equilibrium f_0^{eq}, such that the phase space integration reduces to a sum of moments of the Maxwellian which are known (cf. [13]). Hence, the computation of the flux densities is straightforward.

A drawback of the BGK approximation is a unit ratio of viscosity and thermal diffusivity, i.e. a unit Prandl number $\mathrm{Pr} = \nu/k = 1.0$. In order to set non-unit Prandl numbers, we apply the Prandl number fix of Xu [13].

For consistent treatment of the energy source due to gravitational acceleration, we apply the approach presented by Tian et al. [35]. To that end the mass flux density vector $\vec{\mathcal{F}}_i^{\rho}$ per cell is computed as the average of the adjacent cell faces, i.e.

$$\vec{\mathcal{F}}_i^{\rho} = \frac{1}{4} \sum_{j=1}^{4} \mathcal{F}_j^{\rho} \vec{n}_j, \tag{7}$$

where $\vec{n}_j$ is the face normal. The energy source term is then computed as

$$\mathcal{Q}_i^{\rho E} = \vec{\mathcal{F}}_i^{\rho} \cdot \vec{g}. \tag{8}$$

For more details on the consistent source term treatment see the explanations in [35] and [22].

In Eq. (3) the area integral of the flux density over cell surfaces A_j is approximated by one point Gauss integration, i.e. $\underline{\mathcal{F}}_j$ is evaluated in the middle of the cell face and multiplied by the respective area. Modeling the time dependent distribution function f requires the values of the hydrodynamic variables and their gradients at the cell face, which are computed by linear interpolation and central finite differences, respectively. Since we use local uniform meshes, both approximations will be second order accurate. The stencil used is shown in Fig. 1, where the cells are named as $\underline{W}^{x,y}$ relative to the current cell face. The values at the cell face are computed as

$$\underline{W} = \frac{\underline{W}^{+1,0} + \underline{W}^{-1,0}}{2} \tag{9}$$

and

$$\nabla \underline{W} = \begin{pmatrix} \dfrac{\underline{W}^{+1,0} - \underline{W}^{-1,0}}{\Delta x} \\ \dfrac{\underline{W}^{+1,+1} + \underline{W}^{-1,+1} - \underline{W}^{+1,-1} - \underline{W}^{-1,-1}}{4\Delta x} \end{pmatrix}. \tag{10}$$

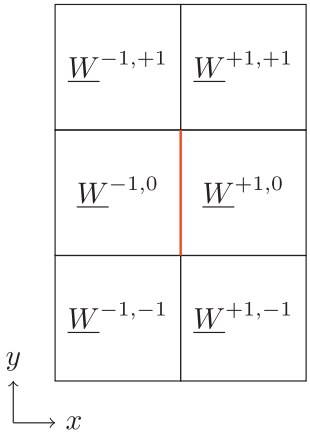

Fig. 1. Stencil for computation of the hydrodynamic variables and their gradients on the cell face. The current cell face is marked in red and the cells are named as $\underline{W}^{x,y}$ relative to the current cell face. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Non-uniform Cartesian meshes

In this section we present the algorithm for coupling uniform meshes of different resolutions in order to obtain non-uniform meshes, which are locally uniform and Cartesian (i.e. the cells are square shaped in two dimensions). We limit ourselves to rectangular domains, such that the mesh generation is straightforward. Complex geometries are not addressed in the scope of this work. Starting from a coarse background mesh, regions of interest are refined by splitting a coarse cell into four finer cells. The resulting mesh has similarities to a quadtree, but all parent-child connectivity is omitted after mesh generation. Multiple refinements are possible. The depth of a cell in the quadtree is called level, such that all cells on the same level have the same size and shape.

The choice of a refinement ratio of two (i.e. quadtree-type refinement) brings an algorithmic advantage. Methods with arbitrary cell sizes have to apply the same time step everywhere. This leads to very small CFL numbers in the coarse regions of the mesh. The CFL number is $\mathrm{CFL} = (U + c_s)\Delta t/\Delta x$, where c_s is the speed of sound. The quadtree refinement allows us to use nested time stepping. This implies, that fine cells perform two time steps of half the size during one corresponding time step on the coarse cells, hence the CFL number will be identical on coarse and fine mesh regions. For simulations where only small parts of the domain require high resolution, this substantially reduces the number of required cell updates.

In the following we will discuss the numerical adaptation of the present gas kinetic scheme on such a mesh and how to treat the interface between coarse and fine cells. We note that the algorithm is not specific for GKS, but is applicable for any local explicit finite volume scheme that utilizes face gradient information.

4.1. Second order accurate ghost cell interpolation

For faces that connect two cells of the same level, two point interpolation and finite differences are used to obtain the hydrodynamic variables and their gradients on the cell face, respectively (see Eq. (9)). This is not directly possible for the cell faces on the interface between coarse and fine levels. A naive approach is to reconstruct the cell face state from any cells around the face. A six point stencil is shown on the left side of Fig. 2. The drawback of this naive apprach is that, no matter how we choose the stencil (including 6 neighbors), it will never be centered, and hence only of first order accuracy.

The right part of Fig. 2 shows a possible remedy of this problem. One can use the same centered stencils from Eq. (9), where the points on the coarse side do not coincide with existing cell centers. These points are interpreted as cell centers of ghost cells. The ghost cell values have to be obtained by suitable second order accurate interpolations. Stencils for these interpolations will be derived below.

Fine ghost cells are introduced for all coarse face or edge neighbor cells of the interface. In order to avoid special cases, we always introduce all four fine ghost cells. By doing this, corners in the refinement region can be treated the same way as straight interfaces and no special cases have to be considered. It will be shown below that coarse ghost cells on the fine side of the interface are also required.

Each coarse ghost cell overlaps four fine cells. Hence, a second order accurate interpolation is obvious. The coarse ghost cell value is denoted by $\tilde{W}_{i,j}$, where the tilde distinguishes ghost cells from fluid cells. The child cell values $W^{k,l}_{i,j}$ are denoted by indices $k, l \in \{-1, 1\}$, corresponding to the location of the child cell in x and y direction, respectively. The fine to coarse interpolation in this notation is given by

$$\tilde{W}_{i,j} = \frac{1}{4} \sum_{k,l \in \{-1,1\}} W^{k,l}_{i,j}. \tag{11}$$

Inserting Taylor expansion for the cell centers yields

$$\tilde{W}_{i,j} = W_{i,j} + \frac{1}{32}\Delta x^2 \left(\frac{\partial^2}{\partial x^2} W_{i,j} + \frac{\partial^2}{\partial y^2} W_{i,j} \right) + \mathcal{O}(\Delta x^3). \tag{12}$$

It is evident that this interpolation is second order accurate.

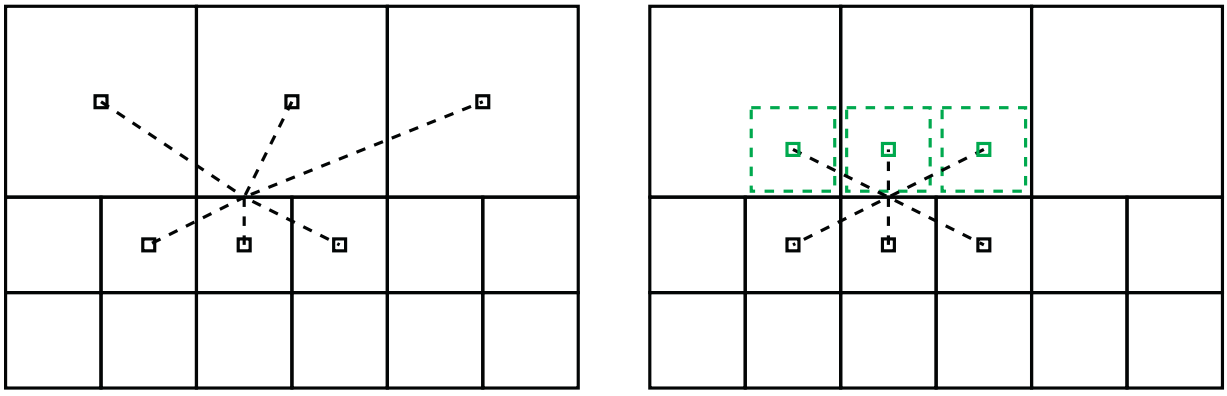

Fig. 2. Interface between a fine mesh on the bottom and a coarse mesh on the top. The dashed lines are the stencil for the reconstruction. The stencil on the left side uses only existing cells. This however results in a stencil that can only be first order accurate, since it uses a non-centered stencil. The stencil on the right side is centered. It requires auxiliary ghost cells (dashed green cells). which have to be constructed from the underlying mesh.

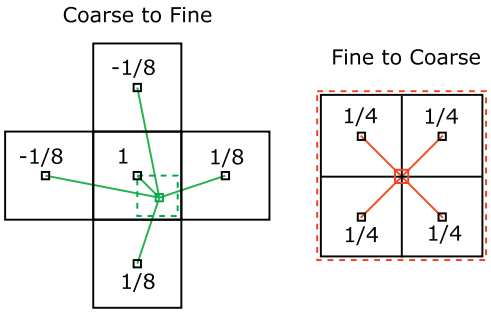

Fig. 3. Second order interpolation stencils for the computation of the ghost cell states. The dashed cells are the interpolated ghost cells and the black solid cells are the underlying fluid cells.

The coarse to fine stencil uses a set of five coarse cells to interpolate four fine ghost cells. The five coarse cells are the parent cell (index 0) of the four fine cells plus its four face neighbors (indices $m, n \in \{-1, 1\}$). The generic interpolation stencil is constructed by multiplying each coarse cell value with a specific weight $w_{m,\,n,\,k,\,l}$ and summing over all coarse cells, such that

$$\tilde{W}^{k,l}_{i,j} = w_0 W_{i,j} + \sum_{m,n\in\{-1,1\}} w_{m,n,k,l} W_{i+m,j+n}. \tag{13}$$

The weight of the parent cell w_0 is, due to symmetry, the same for all four child cells $(k,\, l)$. The stencil is further symmetric in a sense that the weights of the two coarse cells adjacent to the fine ghost cell (i.e. $m = k$ or $n = l$) must be equal, namely w_1. The same holds for the weights of the opposing two coarse cells (i.e. $m \neq k$ and $n \neq l$) with weights called w_2. After inserting Taylor expansion, using the symmetry and forcing the sum of the first order error terms to be zero, the resulting weights are

$$w_1 = \frac{3}{8} - \frac{w_0}{4} \text{ and } w_2 = \frac{1}{8} - \frac{w_0}{4} \tag{14}$$

The Taylor series leaves one open parameter (here w_0) for the coarse to fine interpolation. Every choice of w_0 generates a stencil of second order. A suitable weight w_0 can be obtained, by requiring that subsequent coarse to fine and fine to coarse interpolations of the same cell yield the original value. Inserting Eq. (13) in Eq. (11) yields $w_0 = 1$ after comparison of coefficients. With this choice the remaining weights can be computed from the Eq. (14), such that $w_1 = 1/8$ and $w_2 = -1/8$.

During subsequent coarse to fine and fine to coarse interpolation all four coarse neighbor cells appear twice with w_1 and twice with w_2, such that their contribution cancels out. Hence, the interpolation scheme is self consistent. The accuracy of the resulting interpolation is

$$\tilde{W}^{k,l}_{i,j} = W^{k,l}_{i,j} - \frac{1}{32}\Delta x^2 \left(\frac{\partial^2}{\partial x^2} W_{i,j} + \frac{\partial^2}{\partial y^2} W_{i,j} + 2\frac{\partial^2}{\partial x \partial y} W_{i,j} \right) + \mathcal{O}(\Delta x^3). \tag{15}$$

Both interpolation stencils are also shown in Fig. 3.

4.2. Flux treatment at the interface

The introduction of ghost cells enables the computation of second order accurate fluxes on all cell faces. This includes not only the fine cell faces on the interface between two levels, but also the coarse cell faces on the interface. Due to the overlap of coarse and fine cells it is not obvious how to handle the fluxes consistently, though. The finite volume method is flexible with regard to the shape of the cells in a sense that it does not matter how many faces contribute fluxes to the update of the cell. This opens different options how to utilize the fluxes on the interface.

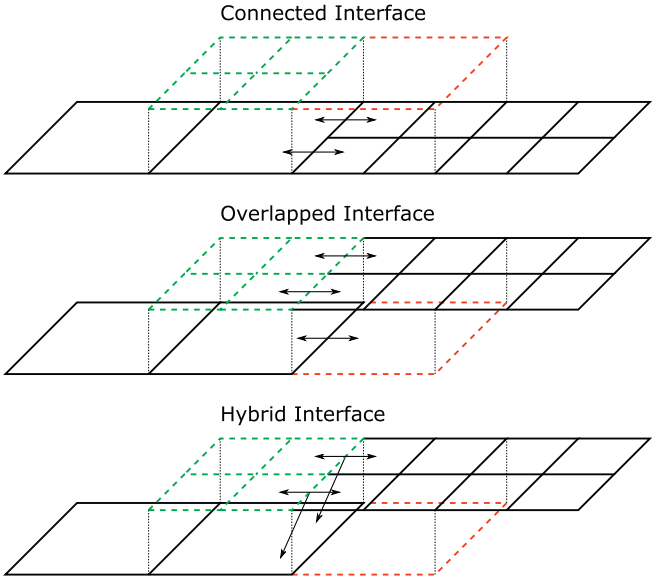

Fig. 4. Arrangement of the ghost cells at the interface for the *Overlapped* and *Connected Interfaces*. For the *Overlapped Interface* the fluxes are applied to the ghost cells. Hence the fine and coarse mesh interact only by the interpolation of the ghost cell states. The conservation property is violated. For the *Connected Interface* the ghost cells are only used for interpolation and gradient computation. The fluxes on the faces between coarse and fine cells are applied directly to the adjacent coarse and fine cells. Therefore, this scheme retains the conservation property. The *Hybrid Interface* combines the advantages of the former two. The ghost cells are included in the update and the fluxes are directly applied to the connected fluid cells.

First we investigate two basic options:

- *Connected Interface:* The fluxes are exchanged directly between the fluid cells, which requires only the fine cell faces. The ghost cells are only used to compute the hydrodynamic variables and gradients on the fine cell faces.
- *Overlapped Interface:* The ghost cells are treated as fluid cells. Fluxes are exchanged only between cells of the same level (including ghost cells). This requires the fluxes on the coarse cell faces. The sole connection between the levels is the interpolation of the ghost cells.

They are compared (together with a third option) in Fig. 4. Both options have different properties. The *Connected Interface* preserves the conservation property of the underlying finite volume method. In case of the *Overlapped Interface*, this property is lost at the interface, because the interpolation is not conservative, implying that mass, momentum and energy might be lost or generated at the interface.

Furthermore, the *Connected Interface* does not take the change of the ghost cells during the first nested time step into account, but keeps the ghost cell value constant. This will increase reflection of acoustic waves passing through the interface. The *Overlapped Interface* brings an advantage here. Since fluxes are applied to the ghost cells, they can also be included in the update of the conserved variables, such that the ghost cells adapt their state during the nested time steps.

Nested time stepping on the overlapped mesh requires an additional remark: It was mentioned above that for each coarse fluid cell at the interface all four child cells are introduced. This corresponds to having two layers of fine ghost cells around the refinement region independent of its shape. When included in the cell update, the outer layer would require suitable boundary conditions, which are difficult to define. Not defining such boundary conditions will invalidate the state of the outer layer during the first nested time step. The invalidation will propagate to the

inner layer during the second nested time step. This is no problem, though, as after the second nested time step the ghost cell values will be recomputed by interpolation and the invalid values are overwritten. Hence, the fluid cells never read from an invalid ghost cell.

Both introduced ways of applying the fluxes have different advantages and disadvantages. Fortunately, it is possible to combine both ways, such that the advantages of both are preserved:

- *Hybrid Interface:* The fluxes are exchanged directly between the fluid cells. On the coarse side of the interface a "copy" of the flux is added to the fine ghost cells and the ghost cells are included in the cell update.

4.3. Nested time steps

The algorithm for the nested time stepping is shown in Algorithm 1. Since the number of levels is usually small (between

Algorithm 1 Recursive nested time stepping.

```
1:  function NESTEDTIMESTEP(level, Δt)
2:
3:      if level ≠ finestLevel then
4:          call FINETOCOARSEINTERPOLATION(level), Eq. (11)
5:      end if
6:
7:      call SETBOUNDARYCONDITIONS(level)
8:
9:      if level ≠ finestLevel then
10:         call COARSETOFINEINTERPOLATION(level), Eq. (13)
11:
12:         call NESTEDTIMESTEP(level + 1, Δt/2)
13:         call NESTEDTIMESTEP(level + 1, Δt/2)
14:     end if
15:
16:     call COMPUTEFLUXES(level), Eq. (6)
17:
18:     call UPDATECELLS(level), Eq. (3)
19: end function
```

0 and 10), we use recursive time stepping. The function NESTEDTIMESTEP(*level*, Δt) performs one time step of length Δt for the given level. First, the coarse ghost cells of the current level are interpolated from the underlying fine cells. At this point all non-boundary ghost cells of the current level have the correct values. Hence, the boundary ghost cells of the current level can be set to fulfill the boundary conditions. Next the fine ghost cells on the next finer level are interpolated. This is followed by the evaluation of the next finer level by calling the same function twice with $level + 1$ and $\Delta t/2$. Only after the finer levels are complete, the fluxes of the current level are computed. Finally, the cells of the current level are updated.

A specific order of this algorithm is required. First, the order FINETOCOARSEINTERPOLATION, SETBOUNDARYCONDITIONS, COARSETOFINEINTERPOLATION is important. The FINETOCOARSEINTERPOLATION is independent of the other two calls, because it just reads information from fluid cells. The SETBOUNDARYCONDITIONS function must also set boundary ghost cells connected to coarse ghost cells (for tangential gradients) and, hence, requires these coarse ghost cells to be set beforehand. The COARSETOFINEINTERPOLATION employs a larger stencil which reads from boundary ghost cells and coarse ghost cells. Second, the UPDATECELLS must be done after the finer levels are evaluated. This is due to the *hybrid interface*. Since fine levels contribute fluxes to the update of the coarse cells, they have to be applied before the latter are being updated.

5. GPGPU implementation and performance

For scientific computing there are two major paradigms in parallelism, which are often combined. Distributed memory parallelism utilizes explicit communication, which requires a specific software design, but scales to tens of thousands of cores [5]. Shared memory parallelism does not require explicit communication, because all cores access the same address space and communicate implicitly. The scalability of this approach is limited by the amount of memory and number of cores that can access this memory concurrently as well as efficiently. In the last decade specialized hardware for shared memory emerged from graphics processing hardware, resulting in General-Purpose-Graphics-Processing-Units (GPGPUs) and other accelerators, such as Intels Xeon Phi series. This hardware uses up to thousands of cores that access the same memory space.

Nowadays the proprietary CUDA-framework [31] for the programming of NVIDIA GPGPUs is so advanced, that much of the tedious programming effort of early CUDA generations is drastically simplified. Hence, CUDA is a very potent framework to develop high performance scientific codes. Nonetheless, it also has limitations. The memory of GPGPUs is usually limited to a few GBs. Hence, very large scale simulations still require distributed memory paradigms for several GPGPUs to communicate explicitly. Further, it must be possible to divide the problem into thousands of independent smaller problems. For explicit, local algorithms this is relatively straightforward, such that they are well suited for massive parallel implementation. Pressure correction schemes on the other hand spend much time in global operations and are hence not that well suited.

As a fully explicit scheme the GKS thrives on massive parallel hardware. We present an efficient GPGPU implementation of the present GKS on quadtree-type meshes.

5.1. DataStructures

The present GKS utilizes two computational entities, namely the cells and the cell faces. The required data and connectivity are shown in Table 1. The cells hold the flow state data, namely the CONSERVEDVARIABLES plus a second field UPDATE. UPDATE collects the fluxes received by the cell and corresponds to the term $\sum_j \mathcal{F}_j A_j$. The cell faces hold the connectivity, i.e. which cells are located around the face, plus a flag used to fix the mass flux to zero in case of walls. The connectivity array FACETOCELL contains 8 elements, which are the six adjacent cells on the same level plus possibly two coarse cells, if the face is located on the interface. All geometrical information, such as cell sizes and face sizes are implicitly given by the level. Also the level does not have to be stored, because as shown in Algorithm 1 the computation kernels are called specifically for the entities of a specific level. The face orientation is also passed implicitly by processing the x-normal and y-normal faces separately.

We distinguish a *Push* scheme and a *Pull* scheme for the flux computation which is performed per face. A *Pull* scheme would

Table 1
Required data for the present GKS. The data type `Real` can represent either single or double precision floating point variables. The value in the square brackets denotes the number of variables required per quantity.

Per cell	Per cell face
`Real` CONSERVEDVARIABLES [4] `Real` UPDATE [4] `Real` MASSFLUX [2]	`int` FACETOCELL [8] `bool` ISWALL [1]
For Ghost Cell Interpolation	
`int` FINETOCOARSE [5]	`int` COARSETOFINE [9]

store the fluxes per cell face and the cell would read (pull) these fluxes for its update. This has a major disadvantage, namely that CellToFace[8] (Compare to Table 1) connectivity would be required. The eight required variables account for two fine neighbors on each side, which would be required at the coarse to fine interface. Also the number of cell faces is approximately twice the number of cells, hence storing the fluxes would require twice the memory of the Update field. Finally nested time stepping on the *Hybrid Interface* would not be possible with this approach. Therefore we utilize a *Push* scheme, where the cell faces directly write the fluxes to the two adjacent cells Update. For the Update variable it does not matter how many fluxes are written to it, as long as the fluxes are correctly scaled with time step Δt and face area Δx. For parallelization this introduces a race condition to the algorithm, because the collection of fluxes which might be processed by different cores, is not thread safe by itself. Hence, we utilize CUDAs built-in atomic operations which are hardware supported since Compute Capability 2.0 and 6.0 for single and double precision, respectively.

Furthermore, we distinguish two types of memory access patterns and their implementation in the present code. We apply these access patterns to the fields in Table 1, such that the they define the order of the elements.

- *Structure of arrays (SOA):* Data belonging to the same quantity is stored subsequently. In case of the ConservedVariables the order would be all densities, all x-momenta, all y-momenta and finally all energies.
- *Array of structures (AOS):* Data belonging to the same entity is stored subsequently. In case of the ConservedVariables the order would be density, x-momentum, y-momentum and energy of the first cell followed by the second cell and so on.

In the past decade a common claim in scientific computing on GPGPUs was that *SOA* brings much more performance than *AOS*. This is due to the concept of coalesced memory access. The *Single Instruction Multiple Data* paradigm of GPGPUs supports concurrent memory reads of multiple cores that perform the same instruction (which together are called a warp). A simultaneous memory read is only efficient though, if the memory of adjacent cores is also adjacent in memory. Hence, it is beneficial for the performance, if (e.g.) all densities are adjacent in memory, because the whole warp will read the densities simultaneously. We encapsulated both memory access patterns with pre-processor macros, such that we can easily swap between them.

Summing up the memory consumption in Table 1 yields 106 Bytes per cell for single and 146 Bytes per cell in double precision. The consumption per face has to be counted twice, since the number of faces is approximately twice the number of cells. The memory consumption of the coarse to fine interfaces, boundary conditions and others are usually negligible for large domains, as they are often of lower dimension. A common professional GPGPU, such as the Tesla P100 with 16GB of memory, can hold up to 150 million cells in single and 110 million cells in double precision with the present data structure, respectively.

It is possible to reduce the memory consumption for connectivity by utilizing pointer chasing, which was not implemented in the current work. Instead of storing all cells that are used for the flux computation per face, it is possible to store just the two cells, that are connected by the face. In addition to that, the CellToCell[4] connectivity and the ParentCell[1] are stored. Since these are stored per cell, they have to be accounted for only once. The ParentCell[1] refers to an overlaying cell on the coarse grid, if the cell is located at a coarse to fine interface. The cells for the computation of the tangential gradients are then addressed by following first FaceToCell and CellToCell subsequently. By this the number of connectivity information per cell can be reduced from 16 to 9 and the memory consumption per cell would be $78\,B$ in single and $118\,B$ in double precision. That would increase the number of cells per GPGPU to 205 million (single) and 136 million (double). In three dimensions this effect is more severe, partly because the number of faces is about three times the number of cells. In three dimensions the present approach (with FaceToCell[12] only) would require 36 connectivities per cell. The optimized version (FaceToCell[2], CellToCell[6], ParentCell[1]) would only require 13 connectivities per cell.

5.2. Numbering

The numbering of the computational entities is a major performance aspect in many computational solid and fluid dynamics codes. Mostly, this is more related to the structure of linear systems of equations and the performance to solve it and less to memory access patterns. Fortunately, our local explicit scheme is matrix free and we can use the numbering to optimize memory access patterns.

We number cells and faces first along the x-direction and then along the y-direction. This holds for both cell faces, oriented normal to x- and normal to the y-direction. An example memory access pattern for the computation of the hydrodynamic variables on the cell face is shown in Fig. 5. It is evident that this numbering supports coalesced memory access.

5.3. Performance measurements

The performance of a numerical code can be measured in many different ways and depends on a lot of factors. For parallel code,

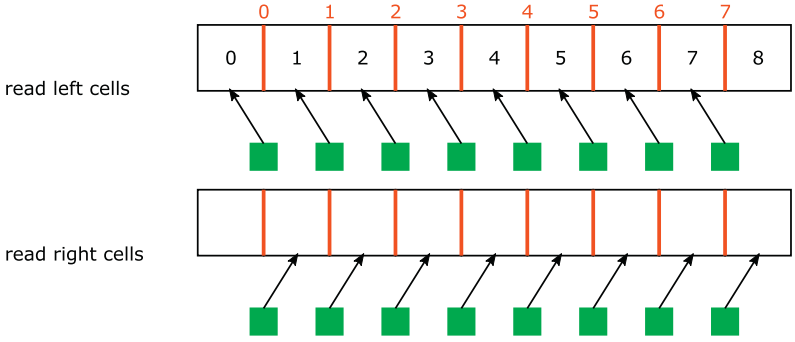

Fig. 5. Coalesced memory access pattern with consistent numbering of faces and cells. Every cell face first accesses its left and then its right neighbor cell. This would be the access pattern for the computation of the hydrodynamic variables on the cell face. The black squares are the cells, the red lines the cell faces, the green squares the CUDA cores and the black arrows represent the memory access.

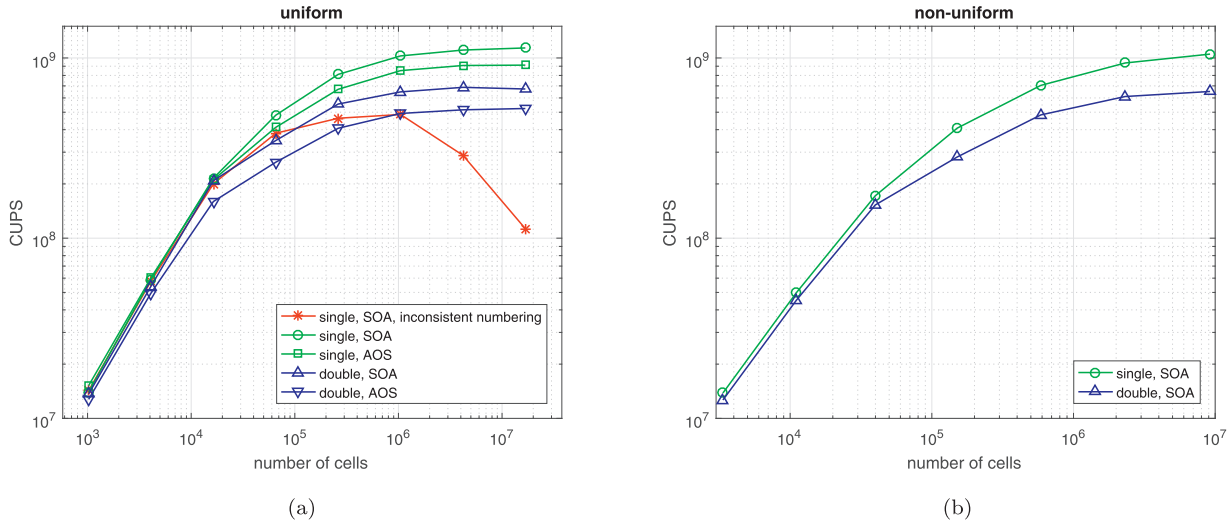

Fig. 6. GPGPU-Performance on a single NVIDIA Tesla P100 on uniform meshes (a) and non-uniform meshes (b). Maximal performance is reached for a system size of above a million cells. The general data structure (SOA vs. AOS) does not have a large impact on performance, as opposed to the cell and face numbering.

performance is mostly related to scaling, i.e. how does the computation time change when the number of cores or the number of computational entities is changed. For single GPGPU applications this scaling is not a good measure, because for moderately small systems all cores are utilized. Hence, we use an absolute measure for performance. We count the *Cell Updates Per Second* (CUPS). For nested time stepping the CUPS are computed as

$$CUPS = \frac{n_{\Delta t} \sum_{i=0}^{n_{levels}-1} n_{cells,i}\ 2^i}{\Delta t_{wall}}, \tag{16}$$

where $n_{\Delta t}$ is the number of time steps for the coarsest level, $n_{cells,i}$ is the number of cells for the ith refinement level and Δt_{wall} is the elapsed wall clock time.

The performance is measured for both uniform and non-uniform meshes. The test case for this performance benchmark is uniform advection with periodic boundary conditions. For each mesh one thousand time steps were performed. The performance results for a NVIDIA Tesla P100 [36] are shown in Fig. 6.

The single precision performance for of the present code with *Structure of Arrays (SOA)* and consistent numbering for both faces and cells gives a performance of 1.1 billion CUPS for large uniform and non-uniform meshes with about ten million cells. The performance increases with the number of cells. This is due to the latency hiding of the GPGPUs. Having more cells gives the GPGPU more options to swap tasks in and out, depending on memory availability. Nearly the same performance is achieved on a non-uniform mesh.

The performance for double precision computations is on the order of six hundred million CUPS, hence little more than half the update rate for single precision. This is due to the fact that the NVIDIA Tesla P100 has twice the number of single precision units than double precision units.

The access pattern *SOA* and *AOS* are also compared. The expectation is, that *SOA* should perform much better, because it allows coalesced memory access. Nevertheless, the performance measures show that *AOS* is only around 20% slower than *SOA*. This is due to efficient caching. Early generations of GPGPUs brought no caches at all, but in recent generations the caches increased to a substantial size in the Megabyte range. The access pattern in the flux computation obviously makes good use of the caches. The data of the adjacent cells is not only read by a single face, but by multiple faces. This can be seen in Fig. 5. Cell 3 for example is first read by face 3 and directly afterwards by face 2. Hence, the data of cell 3 is very likely to be still in the L1 cache, if faces 2 and 3 are evaluated in the same block. Hence, caching works well for both *SOA* or *AOS*.

The numbering, though, has a much larger impact on the performance, especially on the caching. We also show the performance of a simulation, where the mesh is numbered inconsistently, i.e. cells first in x-direction and the faces first in y-direction. The global L2 cache of the Tesla P100 has a size of 4096 *KB*. In single precision the memory consumption is 106 Bytes per cell, see Table 1. Hence the L2 cache can hold about 40.000 cells. In Fig. 6 that is close to the first point, where the CUPS rate with the inconsistent numbering deviates from the one with consistent numbering. Up to this point the whole simulation fits into the L2 cache, such that no performance difference is observed. For simulations with more than a million cells, performance drops drastically. For these resolutions a single block, that shares the L1 cache, is only evaluating a single line of faces, which drastically reduces the reusability of cell values. Hence, consistent numbering is important to achieve high performance on large simulations.

Finally, the same code (SOA memory pattern and no refinement) was run on a single CPU core (INTEL Xeon E5-2640v4) for comparison. There, the code shows a performance of about 3.5 million CUPS in single and 3 million CUPS in double precision. The performance is nearly constant over the problem size. In order to achieve a similar performance as on the GPGPUs, more than 300 CPU cores would be required. These cores would have to be distributed over several compute nodes with mutual network-based memory access, thus requiring communication, e.g. via MPI, which would significantly reduce performance.

Another performance measure is the bandwidth utilization of the GPGPU from the graphics memory to the compute cores. Most GPGPU applications are memory bound and, hence, the bandwidth determines the overall performance. The theoretical bandwidth of the NVIDIA Tesla P100 is 732GB/s. It is just the width of the memory bus times the memory clock rate and has, hence, no information about the maximal bandwidth that can actually be utilized during computation. The maximal sustainable bandwidth is the maximal bandwidth that can be utilized by a computation. It can be measured with a simple CUDA program, just executing $c_i = a_i + \alpha b_i$, where a_i and b_i are very long vectors and α is a scalar [37]. In single precision the NVIDIA Tesla P100 achieves a sustained bandwidth of 543GB/s, which is 74% of the theoretical value.

The utilized bandwidth of the present implementation is measured by the update rate times the memory consumption per update. The memory consumption is calculated by summing up the memory of all global variables accessed by either reading or writing. The two main kernels are the flux computation and the cell update. They require 82% (flux) and 18% (update) of the total computation time of large simulations. Their memory consumption is

$$\underbrace{2}_{\approx 2 \text{ faces per cell}} \times \Big(\underbrace{6 \times 4 \times 4B}_{\text{read cell data}} + \underbrace{8 \times 4B}_{\text{read connectivity}} + \underbrace{1B}_{\text{read isWall}} + \underbrace{2 \times 4 \times 4B}_{\text{write cell update}} + \underbrace{2 \times 1 \times 4B}_{\text{write mass flux}} \Big) = 338B \tag{17}$$

and

$$\underbrace{4 \times 4B}_{\text{read cell update}} + \underbrace{4 \times 4B}_{\text{read cell data}} + \underbrace{2 \times 4B}_{\text{read mass flux}} + \underbrace{4 \times 4B}_{\text{write new cell data}} = 56B \tag{18}$$

for flux computation and update, respectively. The boundary conditions which, require less than 1%, are neglected here. The measured bandwidth is

$$BW = 1.1 \times 10^9 \ \frac{\text{Cell Update}}{s} \times \frac{338B + 56B}{\text{Cell Update}} = 433 \ GB/s. \tag{19}$$

Hence, the present implementation utilizes 80% of the sustained bandwidth of 543GB/s. This shows, that the hardware is well utilized.

6. Validation

In this section we show simulation results for several two dimensional flow problems, utilizing the present scheme on non-uniform meshes and with nested time stepping.

6.1. Validation of wave propagation at grid interface

In Section 4.2 three different ways of utilizing the fluxes at the interface were introduced. Here these three approaches will be compared. As an initial test case we investigate the effective propagation of a pressure wave passing a grid interface. The domain mainly extends horizontally ($L = 64$), but only has a width of two coarse cells broad with periodic boundaries in the vertical direction. Mesh refinement is applied between $x = 47$ and $x = 49$. The wave is initialized as a Gaussian pulse in the middle of the domain at $x = 32$ and with an amplitude of 1% of the ambient pressure p_0. In addition to that, the background medium is advected with a velocity equal to 1% of the speed of sound. The left side of Fig. 7 shows the right going wave after passing through the interface. When observing the wave on the scale of the amplitude, no reflections are visible. The reflections at the interface are more than six orders of magnitude smaller than the amplitude of the wave. Observing the wave at this scale shows the reflections. The *Connected Interface* shows the largest reflection, which is due to not adapting the ghost cells during the nested time step. The *Overlapped Interface* shows some improvement, such that the maximum magnitude of the reflection is less than 50%. Moreover the sign of the wave maximum is opposite to the *Connected Interface*.

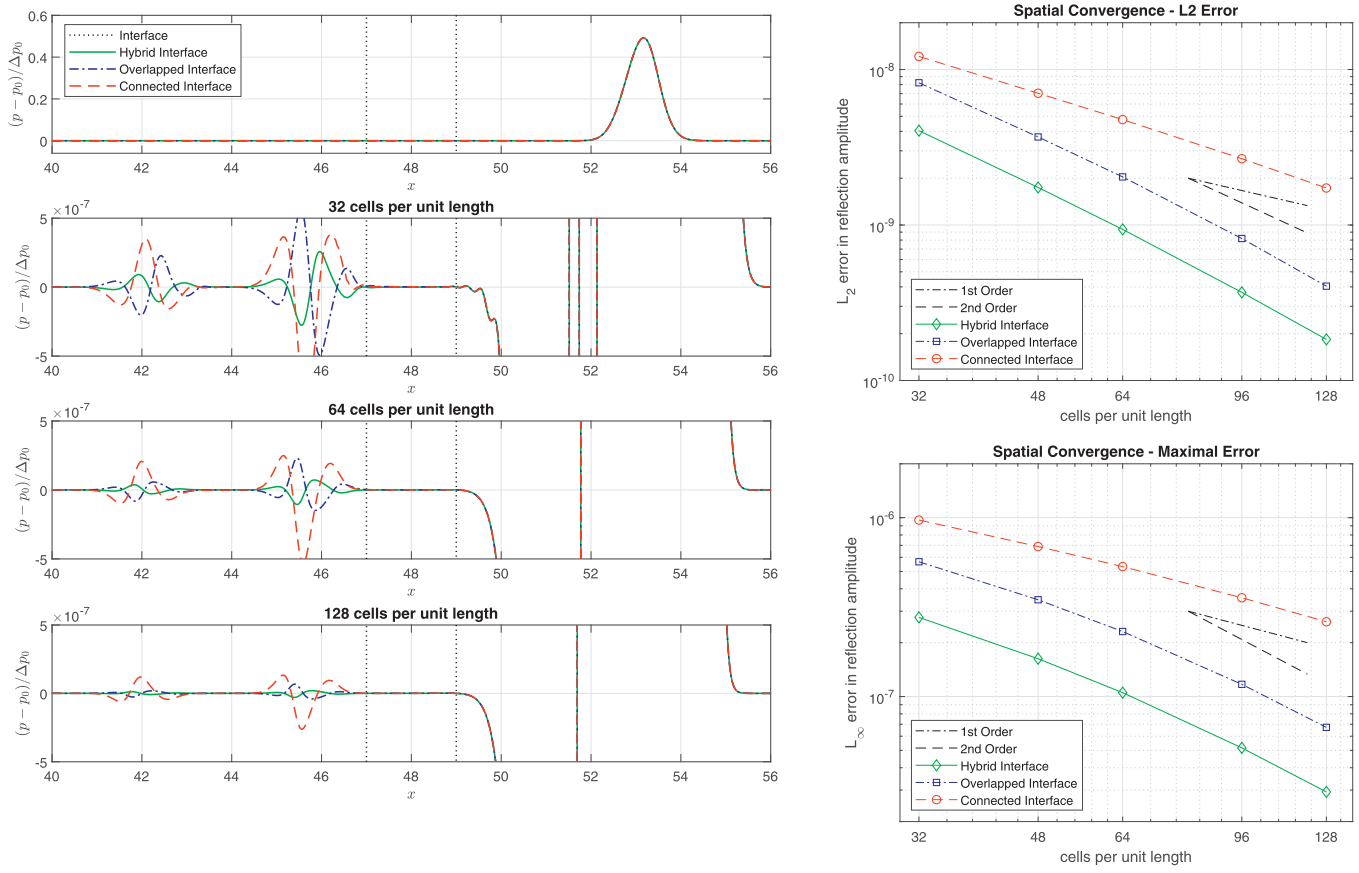

Fig. 7. Results of the wave propagation test case with nested time stepping. A Gaussian pressure impulse is initialized at $x = 32$. Left: The upper plot shows the right going wave after it passes the interface. The lower plots show the same for different resolutions with a magnified pressure axis. The reflections are visible left of the refined region. The maximal amplitude of the reflection of the *Hybrid Interface* is nearly an order of magnitude smaller than the reflection of the *Connected Interface*. Right: Convergence studies for amplitude of the reflected waves. The upper one shows the *L*2-Error and the lower one the maximal error. The *Hybrid Interface* is second order accurate.

This might be due to the non-conservation at the interface. The *Hybrid Interface* shows the same sign of the maximum as the *Connected Interface* and also reduces the amplitude of the reflection by nearly an order of magnitude compared to the *Connected Interface*. Hence, the *Hybrid Interface* shows the best result of the compared approaches for this test case.

Also, the convergence of the reflections is investigated. The L_2 and maximal errors of the pressure in the region around the reflections ($x = (40, 48)$) are shown on the right of Fig. 7. The *overlapped* and *hybrid interfaces* are close to second order, especially for high resolutions, whereas the connected interface deviates from the second order.

6.2. Channel flow

In order to analyze the present coupling algorithm for a canonic viscous flow, the velocity driven channel flow is simulated. The channel has an aspect ratio of $L/H = 32$, where top and bottom walls fulfill the no-slip condition. At the inlet a parabolic velocity profile with integral interpretation of the cell values is used, in order to obtain a prescribed mass flux. The outlet uses an extrapolation boundary for velocity and temperature. The pressure (i.e. density) at the outlet is set to a specified pressure p_0. The analytic solution for the pressure profile is known as

$$p_a(x) = p_0 + g\rho(L - x) \text{ with } g = 8\frac{U\nu}{H^2}. \tag{20}$$

The left plot in Fig. 8 shows the profile of the relative pressure error $(p(x) - p_a(x))/(g\rho L)$. The pressure error is negative in all simulations, because the viscosity error is negative. For the non-uniform meshes, the second half of the domain is refined once. The coarse non-uniform simulation ($ny = 16$) shows a small pressure jump at the interface. Also the pressure error profiles show distinct kinks at the interfaces. This is due to the fact that the error in pressure built-up is only related to the local resolution across the channel. Hence the non-uniform pressure profiles are parallel to those of the uniform simulation with the same resolution.

A convergence study is shown in Fig. 8 on the right hand side. The error is defined as the pressure difference between simulation and analytic solution at the inlet. The solution converges with second order, as is expected for this scheme.

6.3. Lid-driven cavity

Probably the most simulated benchmark for two dimensional flows is the lid driven cavity. This is due to its simplicity with respect to domain and boundary conditions. Physically it is not perfectly suitable as a benchmark, because the problem is not well posed and the solution contains pressure singularities where the lid touches the wall. This is often ignored, because the singularities have little influence on the flow field. Here we choose this benchmark, because it is well known, the domain is closed and it converges to a steady solution at $Re = \frac{UL}{\nu} = 1000$. We use a mesh with a background resolution of 64×64 cells and two refinements towards the lid (see Fig. 9). Compared to the prior two test cases, the shape of the refined regions is complex, which does not effect the present mesh coupling algorithm.

The common reference for this benchmark is the work of Ghia et al. [38]. The results obtained with the *Hybrid Interface* are shown in Fig. 9 on the right. It is evident, that the steady solution is accurately captured. We want to use this benchmark to investigate two properties of the present algorithm:

- The usage of nested time stepping drastically reduces the wall clock time to reach the steady state. This is shown in Fig. 10 on the left. The figure shows the residual change in the L2-norm between two successive snapshots, which are 1000 times steps apart. The mesh consists of 3798, 638 and 2216 cells on background mesh, first and second level, respectively. For one time step on the background mesh the *Hybrid Interface* requires 13,938 cell updates. To simulate the same time span four synchronous time steps have to be performed. They require 26,608 cells updates. Hence, the ratio of required cell updates is found to be similar to the wall clock time ratio. For this simulation the nested time stepping saves nearly half the time.
- The *Overlapped Interface* is not conservative. This is shown on the right side of Fig. 10, where the mass change $m(t)/m(0)$ of the cavity is plotted. The *Overlapped Interface* gains mass on the order of one percent per 100,000 time steps, which is a severe error and keeps the simulation from convergeing (see Fig. 10 on the left). The *Hybrid Interface* keeps the mass constant as expected.

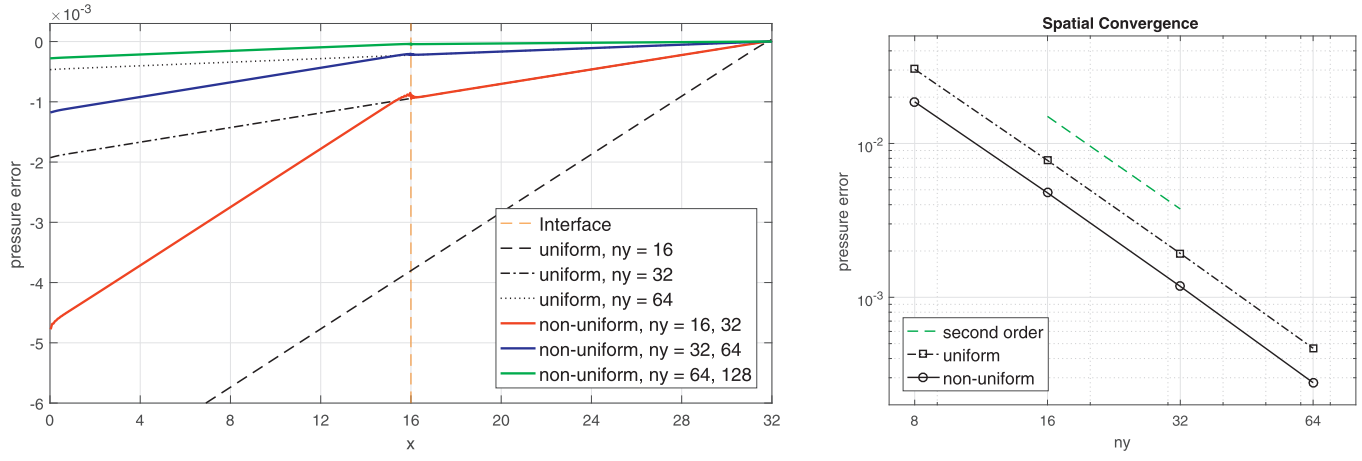

Fig. 8. Left: Relative error in the pressure profile along a channel with fixed outlet pressure. In the non-uniform simulations the second half of the domain is refined once. The non-uniform simulations show distinct kinks at the interface. It is evident that the increase in pressure error is only dependent on the local resolution. In the coarse and fine sections the slope of the pressure error is identical to the slope in the coarse and fine uniform simulations, respectively. Right: Relative error in the pressure at the inlet of the channel. The error converges with second order.

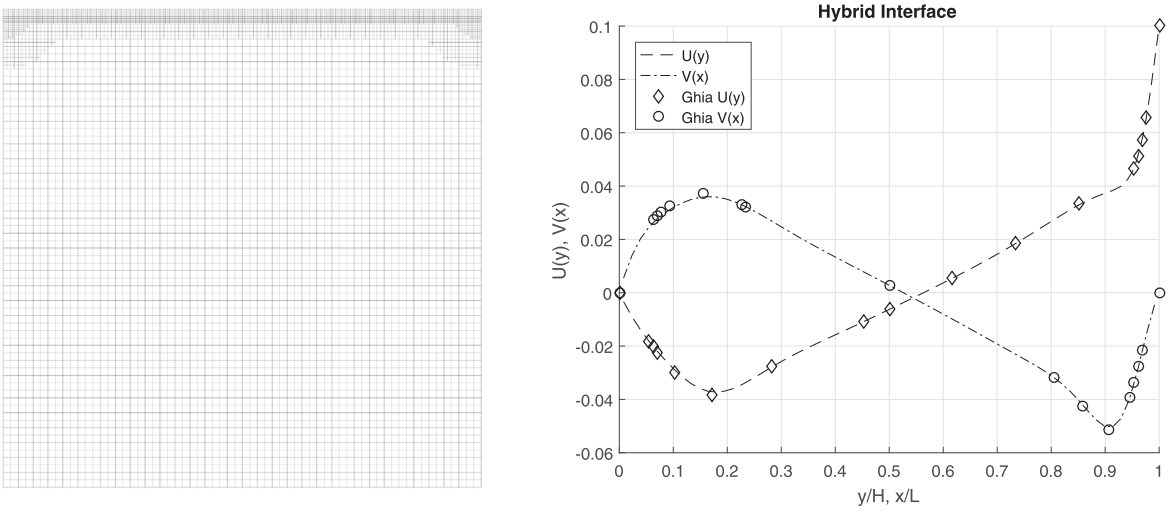

Fig. 9. Left: Mesh for the driven cavity simulations. The background mesh has 64×64 cells and is refined towards the driven lid of the cavity twice. Right: Velocity profiles along the center lines of the cavity at Reynolds number $Re = 1000$. Our simulation compares very well to reference [38].

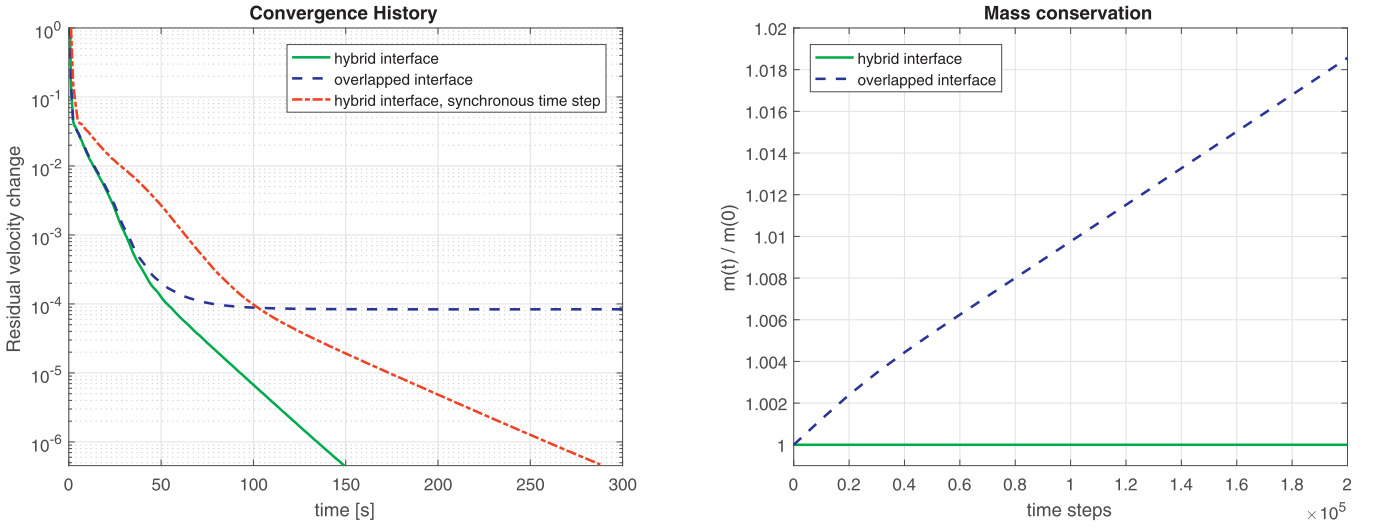

Fig. 10. Left: residual change of the flow field between two snapshots, 1000 time steps apart. Right: Integral mass in the system over time. It is evident, that the mass is conserved when using the *Hybrid Interface* and mass is gained when using the *Overlapped Interface*.

6.4. Square Cavity with differentially heated sides

The GKS shown in Section 2 converges to the compressible Navier–Stokes equations. Hence, we will also validate the present coupling algorithm for thermal compressible flows at low Mach number. High Mach number flows require a more complex flux evaluation with shock capturing and are not considered here.

A well defined benchmark with high quality reference results is given in [39–41]. The domain is a square cavity with no-slip wall at zero velocity. The left and right walls are kept at constant high and low temperature (T_h and T_c), respectively, and the top and bottom walls are insulated. The integral quantity to measure is the Nusselt number, which is a measure for the actual convection enhanced heat transfer with respect to pure heat conduction. The Nusselt number is evaluated as

$$Nu = \frac{q}{\kappa_0 \frac{T_h - T_c}{H}}, \tag{21}$$

where q is the measured heat flux and κ_0 the heat conductivity at the intermediate temperature. Furthermore, the benchmark requires a temperature dependent viscosity and heat conductivity according to Sutherland's law (cf. [40]).

The governing non-dimensional quantities for the setup are

$$\epsilon = \frac{T_h - T_c}{T_0} = 1.2, Ra = \frac{Pr\ g\ H^3\ \epsilon}{\nu^2} = 10^6 \text{ and } Ba = \frac{g\ H}{R\ T_0} = 0.001, \tag{22}$$

which are the dimensionless temperature difference, the Rayleigh number and the Barometric number, respectively. The reference temperature is $T_0 = (T_h + T_c)/2$. The Barometric number [22] is the ratio between the potential and the thermal energy of the fluid and it is a measure for hydrostatic compressibility. Traditionally, the susceptibility to compression in a fluid is evaluated based on the Mach number Ma. However, this does not always make sense as even a stationary ($Ma = 0$) atmosphere has an inhomogeneous density due to hydrostatic compression. The Barometric number can be obtained from the heat capacity ratio γ, the Richardson

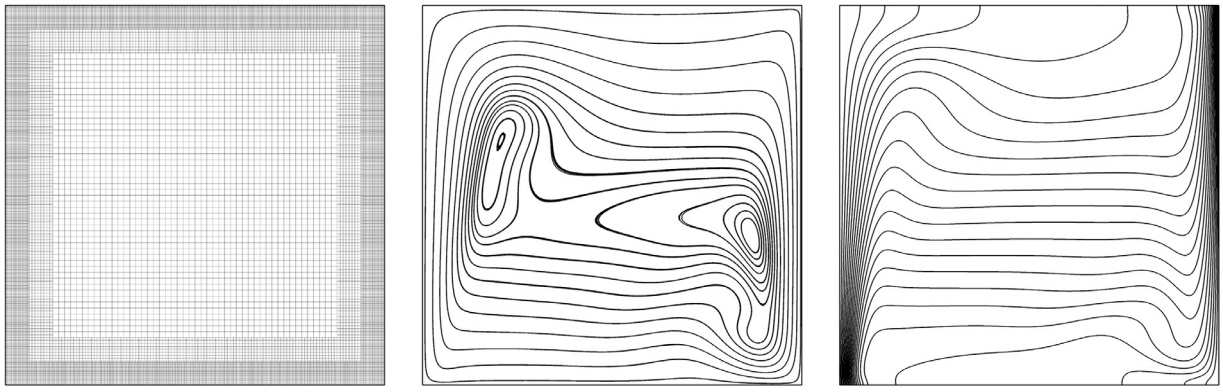

Fig. 11. Mesh and results for the square cavity with differentially heated sides. Left: Mesh with background resolution of 64×64 cells and two refinements. Middle: Streamlines; Right: temperature contours; Both streamlines and temperature contours are obtained from the resolution with 512×512 cells. The left wall is kept a high temperature and the right wall at a low temperature. The thermal expansion breaks the symmetry found in the Boussinesq limit.

number Ri and the Mach number as:

$$Ba = \gamma \; Ri \; Ma^2, \tag{23}$$

where the Richardson number is the ratio between potential and kinetic energy:

$$Ri = \frac{g\,H}{U^2}, \tag{24}$$

with U being some characteristic velocity of the flow. Note that the Richardson number is not defined for $U \to 0$ whereas the Barometric number is still defined by Eq. (22). The Barometric number is the exponent in the barometric formula $p = p_0 \; \exp(-Ba)$. Under isothermal conditions $Ba = 1$ indicates a pressure and density ratio of $\exp(-1) \sim 0.3679$ between the top and the bottom of the domain. For the present benchmark the Barometric number is chosen to be very small, to conform with the references, which do not take this compression into account.

The mesh has three levels with all walls refined in order to resolve the boundary layers. Mesh and results of this simulation are shown in Fig. 11. The flow is not symmetric, due to the thermal expansion. In the bulk of the cavity, the heat is transferred mostly by advection, such that the flow is stratified. At the side walls, where temperature gradients are large, conduction dominates heat transfer between the fluid and the wall.

The results for the Nusselt numbers for the different resolutions are shown in Table 2. The empirical convergence study shows, that the simulations are still in the pre-asymptotic regime, but the convergence order goes towards the expected $q = 2$. The simulations converge well to the reference solutions with only a small residual deviation.

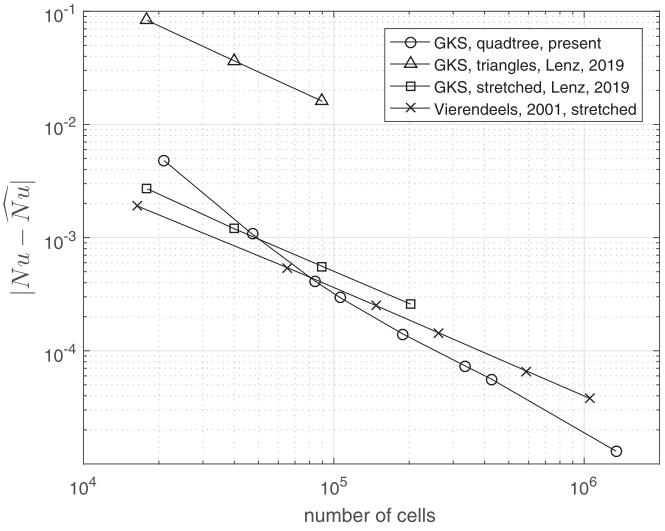

Fig. 12. Comparison of Nusselt numbers obtained on different meshes. The shown values are the absolute errors in Nusseltnumber w.r.t. to the respective Richardson extrapolations.

In order to asses the performance of the Cartesian grid method employed in the present paper with regard to utilization of degrees of freedom, we compare our results also with results from the literature (see Fig. 12). Lenz et al. [22] used a GKS (identical to the present one) on uniform triangular and stretched Cartesian

Table 2
Empirical convergence study for the heat transfer in the square cavity with differentially heated sides.

Background resolution	Nu	Values used for Richardson extrapolation			
64×64	8.681880	x	x		
96×96	8.685609	x			
128×128	8.686268		x	x	x
144×144	8.686388	x			
192×192	8.686541			x	
256×256	8.686609		x		x
288×288	8.686626			x	
512×512	8.686669				x
Richarson extrapolation		8.686594	8.686637	8.686663	8.686682
Empirical order of convergence		3.9	3.7	2.9	2.5
Vierendeels et al. [39]	8.686585				
Becker and Braack [41]	8.686609				

meshes. In the benchmark definition of Vierendeels et al. [39] also a stretched Cartesian mesh is used. We compare errors with respect to the Richardson extrapolation of the respective data set, such that each method converges to its own limit value. The uniform triangular cells show a bad performance, due to not resolving the thermal boundary layers. The methods with stretched cells and the present Cartesian grid method with local refinement show essentially results of comparable quality. The Cartesian grid method utilizes the degrees of freedom as well as the references with stretched meshes.

Hence, the GKS is valid for the simulation of thermal compressible flows on non-uniform Cartesian meshes.

6.5. Rayleigh-Bénard convection at high Rayleigh numbers

Rayleigh-Bénard convection describes natural convection between two parallel plates, where the gravitational acceleration is normal to the plates. For high Rayleigh numbers the flow becomes unstable. In this section we present a validation for such flows in the Boussinesq limit and further simulation results under non-Boussineq conditions.

6.5.1. Boussinesq limit

The Boussinesq limit is characterized by constant fluid properties. The main deviation from compressible flows is the assumption of constant density, i.e. incompressible flow. In terms of non-dimensional numbers this means $Ba \to 0$, $\epsilon \to 0$ and $Ma \to 0$. The coupling between the flow field (velocity and pressure fields) and the temperature field is weaker in the Boussinesq limit than in the full compressible Navier–Stokes equations. The temperature is transported by advection and diffusion (i.e. as a passive scalar) and affects the flow field only by a forcing term, which reacts to deviations from the mean temperature. In order to benchmark our solver, we simulate a Rayleigh-Bénard square cavity such as done by van der Poel et al. [42] and Bao et al. [43].

Starting from a compressible solver, two options exist to get close to the Boussinesq limit. First the compressible solver can be treated as an artificial compresibility method and a passive scalar field for the temperature is introduced. This approach is subsequently denoted by *PS*. Second the temperature difference and the Barometric number (see Eq. (22)) can be chosen very small, to minimize the influence of density variations. This approach is denoted by *CS*. A drawback of the second approach is, that the time steps become small and the simulations require much more time to produce a solution with converged statistics.

The solver validated in the prior sections resembles the *CS* approach. In order to implement the *PS* approach we augment the compressible solver by a scalar field S that represents the temperature in the Boussinesq approximation with the mean value $S_0 = 0$. This choice of S_0 preserves the symmetry of the scalar field. The original temperature T of the compressible flow solver is kept, but it only shows small deviations from the reference temperature T_0. The reference temperature T_0 is set in a way to obtain an Mach number of $Ma = 0.1$, which is sufficiently low to model incompressible flow. Instead of using a constant global forcing, the force F is computed locally by

$$F(\vec{x}) = \beta (S_0 - S(\vec{x}))g\rho, \tag{25}$$

where β is the coefficient of thermal expansion and $\beta \Delta S = \epsilon$ with ΔS being the difference between passive scalar at top and bottom walls. In the Boussinesq limit the Rayleigh number changes to

$$Ra = \frac{Pr\ g\ H^3\ \beta\ \Delta S}{\nu^2}. \tag{26}$$

The advection and diffusion of the scalar field S is implemented utilizing the approach of Li et al. [44] for scalar transport in GKS.

The setup features a square domain with no-slip walls. Top and bottom walls are kept at constant temperature T_0, to allow produced heat to leave the system and thereby reduce the thermal expansion effects (i.e. deviations from the Boussinesq limit) to a minimum. The side walls are insulated.

For the scalar field (i.e. temperature) top and bottom walls are subject to Dirichlet Boundary Conditions with $S = -0.1$ and $S = 0.1$, respectively. The sidewalls are zero flux boundary conditions, i.e. $\partial S / \partial \vec{n} = 0$. The *PS* approach was used to simulated Rayleigh numbers of $Ra = 10^9$, 10^{10}, 10^{11} and 10^{12}.

Additionally, the lower Rayleigh numbers of $Ra = 10^9$ and $Ra = 10^{10}$ were simulated with the *CS* approach on the same meshes. The chosen parameters are a non-dimensional temperature difference of $\epsilon = 0.01$ and a Barometric number of $Ba = 0.001$.

The Prandl number range in the references is $Pr \in \{4.3, 4.38\}$ and we choose $Pr = 4.3$. As a validation we measure the Nusselt number at both top and bottom walls and compare it to the correlation $Nu_m = 0.1 \times Ra^{0.3}$ [43]. The meshes used are non-uniform with refinements towards the top and bottom walls, in order to resolve the thermal boundary layers. The first refinement features a height of $H/16$ from the wall and subsequent refinements quarter this height, i.e. $H/64$ and $H/256$ for the second and third refinements, respectively. Further details about the meshes are given in Table 3.

In order to obtain statistically converged results, we average the Nusselt number over a long time. A non-dimensional free-fall time can be computed as [43]

$$t^* = \frac{H}{\sqrt{g\beta \Delta S H}} = \frac{H}{\sqrt{g \epsilon H}}. \tag{27}$$

The averaging intervals are given in Table 4.

Table 3
Details of the meshes for the Rayleigh-Bénard convection in the Boussinesq limit.

Ra	Background resolution	Refinements	$\Delta x/H$
10^9	512×512	2	4.88×10^{-4}
10^{10}	512×512	3	2.44×10^{-4}
10^{11}	1024×1024	2	2.44×10^{-4}
10^{12}	1024×1024	3	1.22×10^{-4}

Table 4
Nusselt numbers for high Rayleigh number Rayleigh-Bénard convection in a square cavity. The solvers are the passive scalar solver (*PS*) and the compressible solver (*CS*). The results are compared to the model $Nu_m = 0.1 \times Ra^{0.3}$ and the reference simulation from [43].

Ra	Solver	Avg. Int.	Nu_{ref} [43]	Nu_m	Nu_{GKS}	$\frac{Nu - Nu_m}{Nu_m}$
10^9	*PS*	$500\,t^*$	51.57	50.12	51.03	2.1%
10^{10}	*PS*	$500\,t^*$	100.20	100.00	101.99	2.0%
10^{11}	*PS*	$500\,t^*$	198.53	199.53	196.20	−1.7%
10^{12}	*PS*	$235\,t^*$	380.59	398.11	383.84	−3.6%
10^9	*CS*	$200\,t^*$	51.57		49.42	−1.4%
10^{10}	*CS*	$100\,t^*$	100.20		91.94	−8.1%

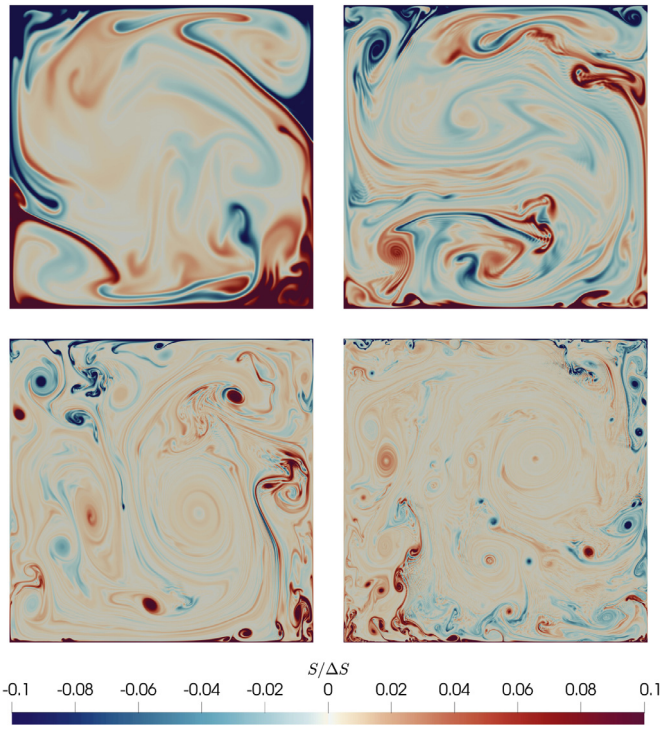

Fig. 13. Instantaneous scalar fields at the Rayleigh numbers 10^9 (top-left), 10^{10} (top-right), 10^{11} (bottom-left) and 10^{12} (bottom-right). The scalar field S is used to resemble the temperature in the Boussinesq approximation.

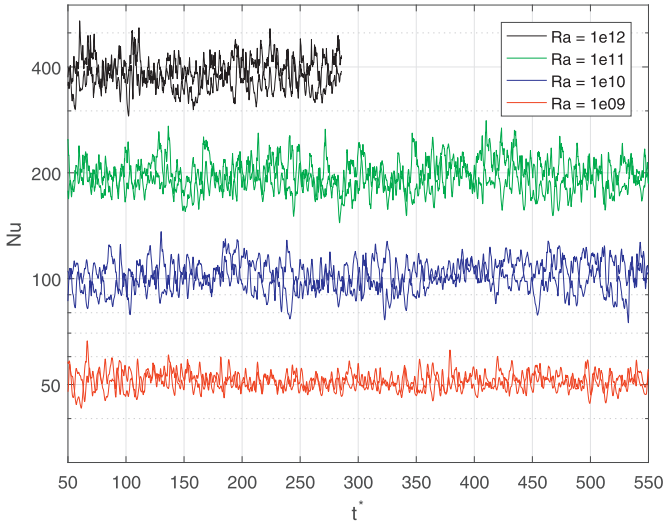

Fig. 14. Time series of the Nusselt numbers for the Rayleigh-Bénard simulation in a square cavity at different high Rayleigh numbers with the passive scalar approach. The two solid lines per Rayleigh number correspond to top and bottom walls and the pairs of lines correspond to descending Ra from top to bottom. The initial transient (i.e. $50t^*$) is cut off, such that the whole averaging interval is shown.

Fig. 13 shows the scalar fields at the four Rayleigh numbers obtained by the *PS* approach. At the lower $Ra = 10^9$ hot and cold fluid ascend and descent at the insulated side walls. At the point where the plumes meet, they turn inwards. With increasing Rayleigh number the plumes break up and isolated vortices containing hot or cold fluid appear. At $Ra = 10^{12}$ plumes are only visible in the vicinity of the wall. The heat is transferred mainly by these isolated vortices. The high Rayleigh number implies low viscosity and diffusivity, such that these vortices can persist for a long time without diffusing neither momentum nor heat. Only when they get close to other vortices or the walls, large gradients drive diffusion and thereby influence the vortices.

The scalar fields at higher Rayleigh numbers show some streaks, that are inherited from the compressible density field. In [43] such streaks are reported to be due to insufficient temporal resolution. In our case the finite Mach number of 0.1 might produce these streaks.

For the validation of the GKS implementation Nusselt numbers are compared in Table 4 and the time series obtained by the PS approach are shown in Fig. 14. Our results with the *PS* approach compare well to both the reference computations and the correlation in [43]. The relative deviation between our results and the correlation Nu_m is well below 5% for all Rayleigh numbers. The deviation to the numerical results of Bao et al. [43] is even smaller.

The result obtained with the *CS* solver for $Ra = 10^9$ is reasonable with a small deviation to Nu_m. It is noteworthy though, that the compressible solution underestimates the Nusselst number compared to Nu_m, whereas the *PS* solutions at $Ra = 10^9$ overestimate the heat transfer. The compressible solution at $Ra = 10^{10}$ shows a much larger deviation of close to 10%. This may be due to the substantially smaller averaging time used for this case which was chosen to limit compute time within reasonable bounds.

This section shows that the GKS is able to quantitatively describe complex two-dimensional turbulence in high Rayleigh number convection.

6.5.2. Non-Boussinesq flows

To the best of our knowledge, reference results of high Rayleigh number non-Boussinesq convection are not available to date. Hence, we could not validate the present scheme for such flow conditions against numerical benchmark or experimental results. The simulation we performed corresponds to a classical Rayleigh-Bénard setup with periodic side walls. We choose a very high Rayleigh number of $Ra = 10^{13}$ and a large temperature difference of $\epsilon = 1.2$. The Prandl number is $Pr = 1.0$. In physical dimensions with a cold temperature of 25°C this corresponds to a hot temperature of approximately 1200°C. With the physical viscosity of air $\nu_{\text{air}} = 1.7 \times 10^{-5}\text{m}^2/\text{s}$ and $g = 9.81\text{m}/\text{s}^2$ the Rayleigh number $Ra = 10^{13}$ correponds to a length scale of around 6m. Hence, these parameters approximately resemble the conditions of a fire at the scale of a small building.

The dimensionless time t^* for this problem can be computed from Eq. (27). The temperature fields of this simulation at times $5t^*$, $15t^*$, $30t^*$ and $100t^*$ are shown in Fig. 15.

The initial condition is an intermediate temperature. At the beginning of the simulation tiny plumes of hot and cold fluid are ejected from bottom and top walls, respectively. In time these plumes merge to larger plumes which move erratically. Some vortices detach from these plumes and rise as heat islands (in case of hot fluid) without dissipating, due to the low thermal conductivity. After the plumes reach the middle of the domain, hot and cold regions mix. Only after a long time, the flow reaches a statistical equilibrium. One turbulent plume for both top and bottom wall is present which frequently eject hot and cold islands/vortices, while the bulk remains at around the intermediate temperature. The final large scale flow field consists of two primary rolls rotating in opposite directions. Even though we cannot validate this specific setup, we find it worthwhile to demonstrate that the present scheme allows to simulate compressible high Rayleigh number flows and resolve the complex flow structures.

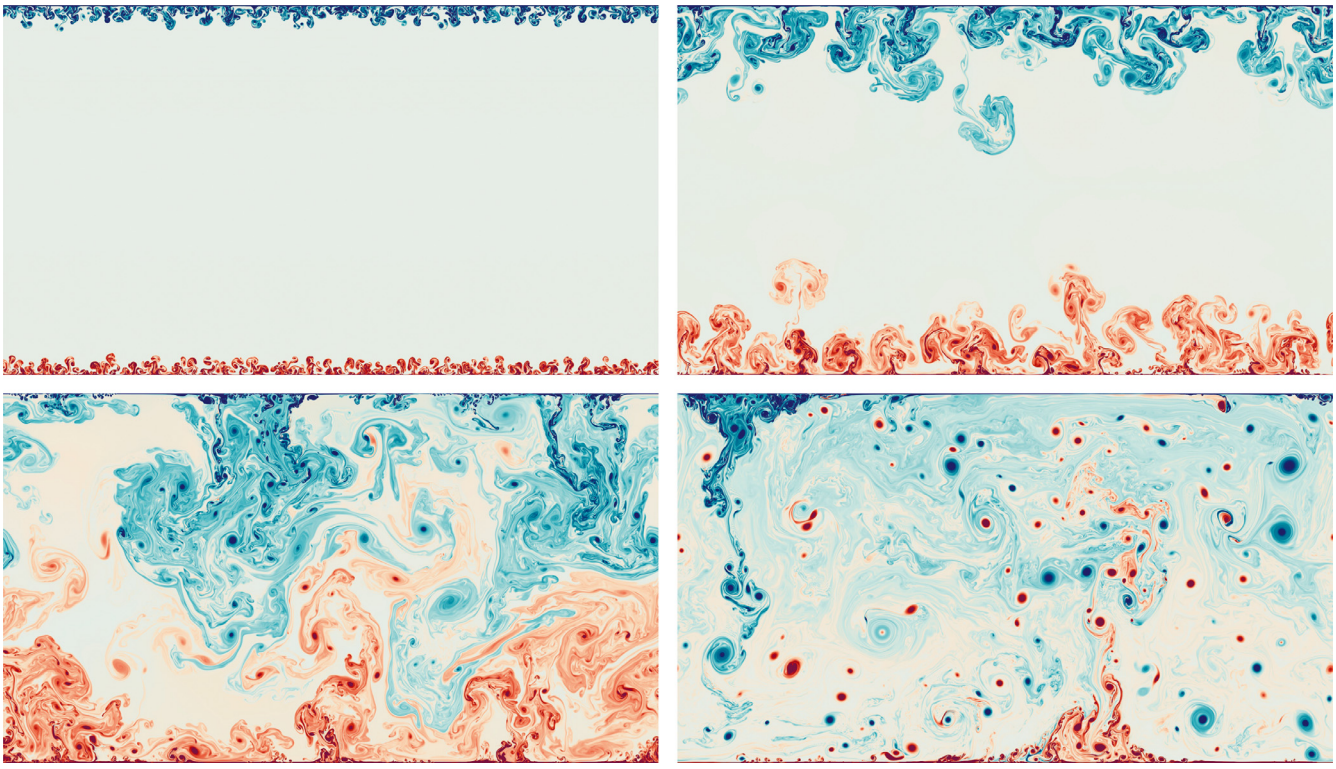

Fig. 15. Evolution of compressible Rayleigh-Bénard convection at $Ra = 10^{13}$, starting from an intermediate temperature. The times are $5t^*$ (top left), $15t^*$ (top right), $30t^*$ (bottom left) and $100t^*$ (bottom right). At the beginning very small plumes are visible, being ejected from both walls. The plumes grow in size and eject vortices. Hot and cold regions mix, such that finally one major turbulent plume for both hot and cold walls are present.

7. Conclusion

This paper introduces an efficient algorithm for a simple gas kinetic scheme for GPGPUs for nested Cartesian grids. The high computational efficiency is obtained by utilizing efficient data structures with indirect addressing allowing complex local refinement. Furthermore, a conservative second order coupling scheme with minimal impact on the flow fields was derived and validated for several incompressible and compressible test cases. The validation includes very high Rayleigh number natural convection in the Boussinesq limit as well as beyond the latter. The featured algorithm is able to recover reference heat transfer (i.e. Nusselt number) with good agreement. We also demonstrate that GKS is capable of simulating highly turbulent compressible convection at a parameter range close to building fires that are characterized by large temperature differences and very high Rayleigh numbers.

The solution of these challenging high Rayleigh number simulations is enabled by the massive parallel GPGPU implementation. The explicit structure of GKS allows straightforward massive parallelization. The choice of refinement ratios of two further allows nested time stepping to satisfy a local CFL condition. The computational performance of the featured implementation was measured on a state-of-the-art NVIDIA Tesla P100 GPGPU. For reasonably large mesh sizes we show a performance of more than 10^9 cell updates per second.

Our comparison to results of the same method on stretched quad meshes and on triangular meshes indicate that a comparable accuracy for a comparable number of degrees of freedom can be obtained on nested Cartesian grids even in the presence of boundary layers. This is significant as Cartesian grids with unity aspect ratio are commonly claimed to be poor discretizations of boundary layer flow due to there perceived inability to distribute the degrees of freedom according to the physics of the problem. Our results favor the interpretation that the advantages of nested Cartesian meshes outweigh their disadvantages compared to unstructured meshes or stretched meshes. The extension of the numerical approach to three dimensions is essentially straightforward and will be the subject of a future publication.

Acknowledgment

The research is supported by Deutsche Forschungsgemeinschaft in the framework of the Research Training Group GRK 2075.

References

[1] Peskin CS. The immersed boundary method. Acta Numer 2002;11. doi:10.1017/S0962492902000077.
[2] Chen S, Xu K, Li Z. Cartesian grid method for gas kinetic scheme on irregular geometries. J Comput Phys 2016;326:862–77. doi:10.1016/j.jcp.2016.09.018.
[3] Hartmann D, Meinke M, Schröder W. An adaptive multilevel multigrid formulation for Cartesian hierarchical grid methods. Comput Fluids 2008;37(9):1103–25. doi:10.1016/j.compfluid.2007.06.007.
[4] Schönherr M, Kucher K, Geier M, Stiebler M, Freudiger S, Krafczyk M. Multithread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs. Comput Math Appl 2011;61(12):3730–43. doi:10.1016/j.camwa.2011.04.012.
[5] Godenschwager C, Schornbaum F, Bauer M, Köstler H, Rüde U. A framework for hybrid parallel flow simulations with a trillion cells in complex geometries. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis; 2013. p. 1–12. doi:10.1145/2503210.2503273.
[6] Kutscher K, Geier M, Krafczyk M. Multiscale simulation of turbulent flow interacting with porous media based on a massively parallel implementation of the cumulant lattice Boltzmann method. Comput Fluids 2018. doi:10.1016/j.compfluid.2018.02.009.
[7] Geier M, Greiner A, Korvink JG. Bubble functions for the lattice Boltzmann method and their application to grid refinement. Eur Phys J Spec Top 2009;171(1):173–9. doi:10.1140/epjst/e2009-01026-6.

[8] Qi J, Klimach H, Roller S. Implementation of the compact interpolation within the octree based Lattice Boltzmann solver Musubi. Comput Math Appl 2016. doi:10.1016/j.camwa.2016.06.025.
[9] Bouzidi M, Firdaouss M, Lallemand P. Momentum transfer of a Boltzmann-lattice fluid with boundaries. Phys Fluids 2001;13(11):3452–9. doi:10.1063/1.1399290.
[10] Geier M, Schönherr M, Pasquali A, Krafczyk M. The cumulant lattice Boltzmann equation in three dimensions: theory and validation. Comput Math Appl 2015;70(4):507–47. doi:10.1016/j.camwa.2015.05.001.
[11] Pasquali A, Schönherr M, Geier M, Krafczyk M. LBMHexMesh: an OpenFOAM based grid generator for the Lattice Boltzmann Method (LBM). In: 7th Open Source CFD International Conference; 2013.
[12] Safari H, Krafczyk M, Geier M. A Lattice Boltzmann model for thermal compressible flows at low Mach numbers beyond the Boussinesq approximation. Comput Fluids 2018. doi:10.1016/j.compfluid.2018.04.016.
[13] Xu K. A Gas-Kinetic BGK scheme for the Navier–Stokes equations and its connection with artificial dissipation and Godunov method. J Comput Phys 2001;171(1):289–335. doi:10.1006/jcph.2001.6790.
[14] Xu K, Mao M, Tang L. A multidimensional gas-kinetic BGK scheme for hypersonic viscous flow. J Comput Phys 2005;203(2):405–21. doi:10.1016/j.jcp.2004.09.001.
[15] May G, Srinivasan B, Jameson A. An improved gas-kinetic BGK finite-volume method for three-dimensional transonic flow. J Comput Phys 2007;220(2):856–78. doi:10.1016/j.jcp.2006.05.027.
[16] Righi M. A gas-kinetic scheme for turbulent flow. Flow Turbul Combust 2016;97(1):121–39. doi:10.1007/s10494-015-9677-2.
[17] Xu K, Huang J-C. A unified gas-kinetic scheme for continuum and rarefied flows. J Comput Phys 2010;229(20):7747–64. doi:10.1016/j.jcp.2010.06.032.
[18] Guo Z, Wang R, Xu K. Discrete unified gas kinetic scheme for all Knudsen number flows II. Thermal compressible case. Phys Rev E 2015;91(3):033313. doi:10.1103/PhysRevE.91.033313.
[19] Xu K, Lui SH. Rayleigh-Bénard simulation using the gas-kinetic Bhatnagar-Gross-Krook scheme in the incompressible limit. Phys Rev E 1999;60(1):464–70. doi:10.1103/PhysRevE.60.464.
[20] Su M, Xu K, Ghidaoui M. Low-Speed flow simulation by the gas-kinetic scheme. J Comput Phys 1999;150(1):17–39. doi:10.1006/jcph.1998.6162.
[21] Wang P, Guo Z. A semi-implicit gas-kinetic scheme for smooth flows. Comput Phys Commun 2016;205:22–31. doi:10.1016/j.cpc.2016.04.005.
[22] Lenz S, Krafczyk M, Geier M, Chen S, Guo Z. Validation of a two-dimensional gas-kinetic scheme for compressible natural convection on structured and unstructured meshes. Int J Therm Sci 2019;136:299–315. doi:10.1016/j.ijthermalsci.2018.10.004.
[23] Li W, Kaneda M, Suga K. An implicit gas kinetic BGK scheme for high temperature equilibrium gas flows on unstructured meshes. Comput Fluids 2014;93:100–6. doi:10.1016/j.compfluid.2014.01.015.
[24] Pan D, Zhong C, Li J, Zhuo C. A gas-kinetic scheme for the simulation of turbulent flows on unstructured meshes. Int J Numer Methods Fluids 2016;82(11):748–69. doi:10.1002/fld.4239.
[25] Olshanskii MA, Terekhov KM, Vassilevski YV. An octree-based solver for the incompressible Navier–Stokes equations with enhanced stability and low dissipation. Comput Fluids 2013;84:231–46. doi:10.1016/j.compfluid.2013.04.027.
[26] Pletzer A, Jamroz B, Crockett R, Sides S. Compact cell-centered discretization stencils at fine–coarse block structured grid interfaces. J Comput Phys 2014;260:25–36. doi:10.1016/j.jcp.2013.12.020.
[27] Guittet A, Theillard M, Gibou F. A stable projection method for the incompressible Navier–Stokes equations on arbitrary geometries and adaptive Quad/Octrees. J Comput Phys 2015;292:215–38. doi:10.1016/j.jcp.2015.03.024.
[28] Batty C. A cell-centred finite volume method for the Poisson problem on non-graded quadtrees with second order accurate gradients. J Comput Phys 2017;331:49–72. doi:10.1016/j.jcp.2016.11.035.
[29] Yuan R, Zhong C, Zhang H. An immersed-boundary method based on the gas kinetic BGK scheme for incompressible viscous flow. J Comput Phys 2015;296:184–208. doi:10.1016/j.jcp.2015.04.052.
[30] Lyu F, Xiao T, Yu X. A fast and automatic full-potential finite volume solver on Cartesian grids for unconventional configurations. Chin J Aeronaut 2017;30(3):951–63. doi:10.1016/j.cja.2017.03.001.
[31] NVIDIA. CUDA C programming guide; 2018. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[32] Parashar N, Srinivasan B, Sinha SS, Agarwal M. GPU-accelerated direct numerical simulations of decaying compressible turbulence employing a GKM-based solver. Int J Numer Methods Fluids 2017;83(10):737–54. doi:10.1002/fld.4291.
[33] Liu J., Hu F.Q., Li X.. Performance comparison on parallel CPU and GPU algorithms for unified gas-Kinetic scheme 2018; https://arxiv.org/abs/1810.08137.
[34] Bhatnagar PL, Gross EP, Krook M. A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-Component systems. Phys Rev 1954;94(3):511–25. doi:10.1103/PhysRev.94.511.
[35] Tian CT, Xu K, Chan KL, Deng LC. A three-dimensional multidimensional gas-kinetic scheme for the Navier–Stokes equations under gravitational fields. J Comput Phys 2007;226(2):2003–27. doi:10.1016/j.jcp.2007.06.024.
[36] NVIDIA. NVIDIA Tesla P100: whitepaper. 2016. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.
[37] Deakin T, Price J, Martineau M, McIntosh-Smith S. GPU-STREAM v20: benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models. In: Taufer M, Mohr B, Kunkel JM, editors. High performance computing. Lecture notes in computer science, vol. 9945. Cham: Springer International Publishing; 2016. p. 489–507. doi:10.1007/978-3-319-46079-6_34.
[38] Ghia U, Ghia K, Shin C. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. J Comput Phys 1982;48(3):387–411. doi:10.1016/0021-9991(82)90058-4.
[39] Vierendeels J, Merci B, Dick E. Numerical study of natural convective heat transfer with large temperature differences. Int J Numer MethodsHeat Fluid Flow 2001;11(4):329–41. doi:10.1108/09615530110389117.
[40] Vierendeels J, Merci B, Dick E. Benchmark solutions for the natural convective heat transfer problem in a square cavity with large horizontal temperature differences. Int J Numer MethodsHeat Fluid Flow 2003;13(8):1057–78. doi:10.1108/09615530310501957.
[41] Becker R, Braack M. Solution of a stationary benchmark problem for natural convection with large temperature difference. Int J Therm Sci 2002;41(5):428–39. doi:10.1016/S1290-0729(02)01335-2.
[42] van der Poel EP, Stevens RJAM, Lohse D. Comparison between two and three dimensional Rayleigh-Bénard convection. J Fluid Mech 2013;736:177–94. doi:10.1017/jfm.2013.488.
[43] Bao Y, Luo J, Ye M. Parallel direct method of DNS for two-dimensional turbulent Rayleigh-Bénard convection. J Mech 2017;66:1–8. doi:10.1017/jmech.2017.54.
[44] Li Q, Fu S, Xu K. A compressible Navier–Stokes flow solver with scalar transport. J Comput Phys 2005;204(2):692–714. doi:10.1016/j.jcp.2004.10.026.