NATriuM – An Extensible Framework for Lattice Boltzmann Simulations on Irregular Grids

Andreas Krämer[1], Dominik Wilde[1], Knut Küllmer[1], Dirk Reith[1], Holger Foysi[1], Wolfgang Joppich[1]

[a] *Institute for Technology, Renewables and Energy-efficient Engineering, Bonn-Rhein-Sieg University of Applied Sciences, Grantham-Allee 20, 53757 Sankt Augustin, Germany*
[b] *Department of Mechanical Engineering, University of Siegen, Paul-Bonatz-Straße 9-11, 57076 Siegen-Weidenau, Germany*

Abstract

The lattice Boltzmann method is a modern approach to simulate fluid flow. In its original formulation, it is restricted to regular grids, second-order discretizations, and a unity CFL number. This paper describes our new off-lattice Boltzmann solver NATriuM, an extensible and parallel C++ code to perform lattice Boltzmann simulations on irregular grids. NATriuM also allows high-order spatial discretizations and non-unity CFL numbers to be used. We demonstrate how these features can efficiently decrease the number of grid points required in a simulation and ~~by these means~~ thus reduce the computational time, compared to the standard lattice Boltzmann method.

Keywords: computational fluid dynamics, unstructured grid, off-lattice Boltzmann, semi-Lagrangian, high-performance computing

1. Introduction

The Lattice Boltzmann method (LBM) is a modern approach to computational fluid dynamics. While classical fluid flow solvers directly discretize the governing equations of fluid motion, the LBM has emerged out of kinetic theory. Accordingly, it models the dynamics of particle distribution functions due to streaming and collision of particles. This detour into the mesoscopic world is justified in that it simplifies the numerical procedure. Other than classical solvers, the LBM does not have to solve a Poisson equation for the pressure or calculate spatial derivatives. Instead, it alternately (1) streams particle distributions along a fixed set of discrete directions to the neighbor nodes on the lattice and (2) performs collision steps locally at each grid point. First of all, both the streaming step and the collision step are explicit and exceptionally efficient on lattices, i.e. regular computation grids. Furthermore, the collision model can be tailored to give the correct physics on the continuum scale and may even be used to include microscopic interactions that exceed the scope of continuum-based simulation.
On the one hand, these properties suit the simulation of complex flows on highly parallel computers. Prominent examples are flows through porous media [?], flows with multiple phases and constituents [?], and turbulent flows [?]. On the other hand, they limit the geometrical and algorithmic flexibility of the method. The geometrical restriction to

regular computation grids is the method's most unsatisfactory property. Most importantly, it retards the LBM to be applied to complex wall-bound flows, many of which are ubiquitous in science and engineering. Consequently, generalizations of the LBM were developed to extend ~~the LBM's~~ its domain of applicability. Besides hierarchical grid refinements ~~like~~ such as the one by Filippova and Hänel [**?**], many authors combined the LBM concept with classical discretization schemes. They developed various finite-difference LBMs [**?** **?**], finite-volume LBMs [**?** **?**], finite-element LBMs [**?** **?**], and discontinuous-Galerkin LBMs [**?** **?**].

In contrast to the standard LBM, these off-lattice Boltzmann methods (OLBMs) decouple the discretization of the particle velocity space and the physical space. The particle velocities are modeled using the customary LBM stencils (D2Q7, D2Q9, D3Q15, D3Q19, D3Q27). This discretization leads to the discrete Boltzmann equation, which can be solved with an appropriately chosen numerical scheme on an irregular grid. The operator splitting into streaming and collision ~~step~~ steps can also be formulated in general terms [**?**], introducing the well-known 0.5-shift into the relaxation time (cf. **??**) and rendering the OLBM schemes stable at high Reynolds numbers. With these preliminaries, ~~they have overcome~~ the teething troubles of the first finite volume LBMs [**?**] have been overcome.

The number of published OLBMs is growing steadily. They extend the standard LBM by several appealing properties. ~~First of all, they~~ They support simulations on unstructured grids and higher-order spatial convergence [**?**]. ~~OLBMs~~ They have already proven that they can reduce the number of required grid points remarkably, when compared to the standard LBM [**?**]. Their temporal and spatial error can be controlled independently by dropping the ~~unity-CFL~~ unity-Courant-Friedrichs-Lewy (CFL) number restriction; and finally, they can operate with larger time steps via either implicit time integration [**?**] or a recently developed semi-Lagrangian streaming step [**?**].

Despite ~~of~~ these promising properties, most OLBMs have only been applied to simple test problems in two dimensions. Simulations in three dimensions, performance studies, parallel implementations, and comparisons to other CFD methods remain rare~~to this date~~. The future prospects of OLBMs will depend on their success ~~,~~ when being applied to complex flows in three dimensions. However, while there ~~is~~ are an increasing number of publications on standard LBM codes, a comprehensive and extensible tool for OLBM simulations ~~has not~~ does not appear to have been published, ~~to the knowledge of the authors~~yet.

The present paper attempts to fill this gap. We introduce our extensible and parallel OLBM solver NATriuM. NATriuM ~~is a *C++* code that is~~ was first developed to perform **N**umerical simulations for **A**pplications in **Tri**bology **u**sing **M**ulticore processors. It developed from the original idea into a multi-purpose flow solver written in C++ and based on the finite element library deal.II [**?**]. It uses deal.II's quadrilateral grid structure and interface to the parallel environment Trilinos [**?**] and the octree library p4est [**?**]. The code is modular and follows a dimension-independent programming approach, which makes it easy to include new spatial discretization schemes, time integrators, boundary conditions, particle velocity stencils, and collision models. The complete setup makes NATriuM an ideal tool for exploring the potential of OLBMs.

This article proceeds as follows. In the Methodology section, the LBM and its generalization to unstructured grids are briefly summarized. Then, the most important features of the presented tool are consecutively described~~and demonstrated~~ , demonstrated and

discussed. These features are the efficient utilization of third-party libraries, the modular code structure, the high-order convergent spatial discretization, the use of arbitrary quadrilateral and hexagonal grids, and the dimension-independent programming concept. Each of the Sections ?? – ?? addresses one of these features and divides into two subsections "Description" and "Results and Discussion". The simulations within this manuscript include a performance study, a convergence test, shear flows in sinusoidal geometries, and a three-dimensional turbulent channel flow on an irregular grid.

2. Methodology

2.1. LBM

The LBM solves a discrete form of the Boltzmann equation

$$\frac{Df}{Dt} = \Omega(f), \tag{1}$$

where f denotes the particle distribution function. While the total derivative on the ~~left hand~~ left-hand side accounts for free-flight (advection) of particles, the term on the ~~right hand~~ right-hand side accounts for particle collisions. The concrete collision term Ω determines the hydrodynamics and can take various forms.
The LBM discretizes the particle-velocity space into a set of discrete velocities $e_0, \ldots, e_{Q-1}$, resulting in a vector of particle distribution functions $f = (f_0, \ldots, f_{Q-1})$ and a discrete collision operator $\Omega = (\Omega_0, \ldots, \Omega_{Q-1})$, $Q \in \mathbb{N}$. The most important particle-velocity stencils are the D2Q9 ~~stencil~~ (dimension $D = 2$, number of velocities $Q = 9$)~~and the~~, D3Q15, D3Q19, and D3Q27 stencils.
Equation (??) can be split into a collision and an streaming step, cf. ??,

$$\tilde{f}_\alpha(x,t) = f_\alpha(x,t) + \Omega_\alpha\left(f(x,t)\right) \quad \text{and} \tag{2}$$

$$f_\alpha(x,t) = \tilde{f}_\alpha(x - \delta_t e_\alpha, t - \delta_t), \quad \text{respectively.} \tag{3}$$

This splitting introduces a second-order temporal error ~~,~~but greatly facilitates the numerical procedure. Most importantly, it renders the collision step (i.e. Equation (??)) local in space and time. Notably, the splitting does not affect the spatial accuracy of the scheme, which offers the potential for high-order spatial convergence. This property follows from the theoretical derivation but is also demonstrated later for an exemplary flow.

2.2. Advection Operator

On regular grids, the advection step is exact as $x - \delta_t e_\alpha$ represents a ~~neighbor~~ neighboring node on the lattice; the advection reduces to a simple index shift. On irregular grids, the streaming step has to be replaced by a more sophisticated procedure.
~~Some previous works~~ Previous work [? ?] utilized that Equation (??) ~~equates~~ is equivalent to solving the initial value problem

$$\frac{Df}{Dt} = 0, \quad f_\alpha(t_{i-1}) = \tilde{f}_\alpha(t_{i-1}), \tag{4}$$

for t_i, where t_{i-1} denotes the previous time step. Among the various numerical approaches to this linear advection equation, NATriuM implements the one by Min and Lee [**?**] in using their spectral-element discontinuous Galerkin scheme.
As an alternative to solving (**??**), we have recently developed a semi-Lagrangian streaming step [**?**]~~.~~, cf. **??**. Our new approach uses a semi-Lagrangian method to approximately solve Equation (**??**), interpolating the distribution functions at the departure points $x - \delta_t e_\alpha$ by finite elements. This procedure circumvents ~~to use~~ using a time integrator, which advantageously eliminates the pervasive CFL constraint. Moreover, it requires only one evaluation of the spatial operator per time step. Those appealing properties of the semi-Lagrangian streaming step are discussed in detail in a separate publication [**?**].

2.3. Collision Model

The form of the collision operator Ω determines the hydrodynamic equations that are solved by the lattice Boltzmann simulation. The most important models are single-relaxation time [**?**], multiple-relaxation time [**?**], entropic models [**?**] and their respective extensions. We restrict the description in this subsection to the widely-used ~~Bhatnagar-Groß-Krook~~ Bhatnagar-Gross-Krook (BGK) model [**?**].
The BGK model is defined as $\Omega(f) = -\frac{1}{\tau}(f - f^{\mathrm{eq}})$ with

$$f_\alpha^{\mathrm{eq}} = w_\alpha \rho \left(1 + \frac{u_i e_{\alpha i}}{c_S^2} - \frac{u_i u_i}{2c_S^2} + \frac{u_i e_{\alpha i} u_j e_{\alpha j}}{2c_S^4}\right)$$

~~and~~ (Einstein summation over i and j implied) and $\tau = \frac{\nu}{c_S^2 \delta_t} + 0.5$. Here, ν, c_S, and w_α denote the kinematic viscosity, speed of sound, and lattice-specific weights, respectively. The conservative moments, $\rho := \sum_\alpha f_\alpha$ and $\rho u := \sum_\alpha f_\alpha e_\alpha$, recover the compressible Navier-Stokes equations in the weakly compressible limit for Mach number $Ma \ll 1$.

The unstructured streaming step and the collision step are basic ingredients of the off-lattice Boltzmann approach. They are supplemented by appropriate boundary conditions for the distribution functions and time integrators for the advection equation (**??**). Each of these components can take various ~~concrete~~ forms that are surveyed in more detail as the paper proceeds. The following sections explain the most important aspects of the off-lattice Boltzmann code NATriuM, starting with a description of the incorporated third-party libraries.

3. Third-Party Libraries

3.1. Description

NATriuM uses four different third-party libraries. It is largely based on deal.II, a comprehensive, award-winning finite element library. Deal.II implements the basic components of finite element and discontinuous Galerkin methods, such as computational grids, shape functions, numerical quadratures, sparse matrices, etc. It also bases a lot of its functionality on other libraries, among which NATriuM uses boost, p4est [**?**], and Trilinos [**?**].

Boost is ~~necessarily~~ required from deal.II. NATriuM also uses its unit test framework to maintain the code quality at each stage of the developing process.
P4est accounts for parallel partitioning of the mesh. It employs distributed octrees to balance the workload among processors on distributed memory, scaling up to thousands of cores. Quadrilateral (2D) and hexagonal (3D) meshes are supported.
Trilinos is a collection of packages for large-scale scientific computing. Most importantly, it offers linear algebra functionality on distributed memory. All large-scale matrices and vectors in NATriuM are built on Trilinos' data structures, ~~which ensures~~ ensuring efficient and parallel matrix-vector operations. This efficiency is crucial ~~,~~ since the linear algebra operations usually make up the major part of NATriuM's runtime.

3.2. Results and Discussion

To demonstrate the benefits of using highly parallel third-party libraries, the parallel performance is evaluated for simulations of a three-dimensional Couette flow. The flow domain $\tilde{D} := [0, L] \times [0, 1] \times [0, 1]$ was discretized into cells of equal size, with periodic boundaries along the x- and y-directions and solid walls at $z = 0$ and $z = 1$, cf. [**?**] for the wall boundary condition. The flow was simulated with BGK collisions and a discontinuous Galerkin streaming with classical Runge-Kutta time stepping, as described in [**?**]. The order of finite elements was set to $p = 1$. The problem size was varied by the refinement level N and the length of the domain, resulting in a total of $N_{\text{points}} = (p+1)^3 \cdot 2^{3N} \cdot L$ grid points, i.e. the number of quadrature nodes per cell times the number of cells. Each compute node consisted of two Intel Xeon X5650 six-core processors and 48 GB RAM. One MPI process was assigned to each core.

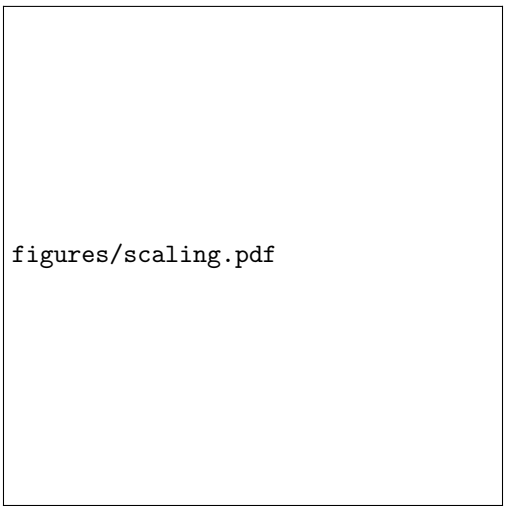


Figure 1: Weak and strong scaling on up to 64 nodes, compared to an ideal linear speedup. The number of cells per node and number of total cells were 131 072 and 1 048 576 for the weak and strong scaling, respectively.

The parallel performance was studied for a fixed problem size per node (weak scaling) and overall fixed problem size (strong scaling). Figure **??** shows the speedup. The simulations for the weak scaling had $N = 5$ and $L = 4 \cdot \#\text{Nodes}$. Varying the number of

nodes from one to 64, a close-to-ideal speedup was obtained. On 64 nodes, the parallel efficiency was still 84%. The simulations for the strong scaling had $N = 5$ and $L = 32$. From two to 24 nodes, a super-linear speedup was observed. On 64 nodes, the parallel efficiency was still 90%.
The scalability of NATriuM is due to the interplay of deal.II, Trilinos and p4est. The combination of these three has been shown to scale well to more than 16 000 processors [**?**]. Their incorporation in NATriuM facilitates quickly writing highly parallel code.

4. Modular Code Structure

4.1. Description

NATriuM is composed as ~~a~~ an object-oriented library, allowing for easy extensions through a modular program structure~~.~~, cf. Fig. **??**. The two central classes are called `ProblemDescription<dim>` and `CFDSolver<dim>`. While the problem description contains the computational grid and all relevant information about the flow, the CFD solver steers the simulation. This separation between flow description and simulation facilitates both using different solver configurations for the same flow and easily setting up new flows. These two basic components of NATriuM are described in more detail in the following.

4.1.1. Implementation of the flow definition

The definition of the flow includes the relevant physical fluid measures (i.e., the kinematic viscosity for incompressible, isothermal flow), the mesh, the initial and boundary conditions~~as well as~~, and external forces. The class `ProblemDescription<dim>` serves as a container for this information. Each concrete flow can ~~either be~~ be either implemented as a subclass of `ProblemDescription<dim>` or read in from a file.
The boundary conditions are defined in a ~~so called~~ so-called `BoundaryCollection<dim>` that handles three different types of boundary conditions: Periodic boundaries, linear flux boundaries, and "standard LBM boundaries". They are passed to the boundary collection in an analogous manner but their effect on the code differs greatly. Periodic boundaries contain a list of cell and face pairs that are neighbors across the boundary. Linear boundaries are incorporated into the sparse matrix and a constant system vector (upon assembly of the spatial operator), and "standard LBM boundaries" are processed separately in each iteration, as in the standard LBM. Currently, NATriuM supports only four types of boundary conditions: periodic boundaries, velocity bounce-back boundaries for the discontinuous Galerkin advection (cf. [**?**]), Grad's boundaries for velocities (cf. [**?**]), and Grad's boundaries for pressure.
Before the simulation starts, the problem description is passed to the CFD solver.

4.1.2. Implementation of the flow solver

The `CFDSolver<dim>` class coordinates the simulation. The solver's configuration is defined in a separate class `SolverConfiguration` that has to be specified either in a configuration file or through setter functions. Upon construction, the `CFDSolver<dim>` sets up all components of the simulation code according to the solver configuration and manages their interplay. The pivotal components are the abstract classes `Stencil`, `CollisionModel`, `AdvectionOperator<dim>` and `TimeIntegrator`.

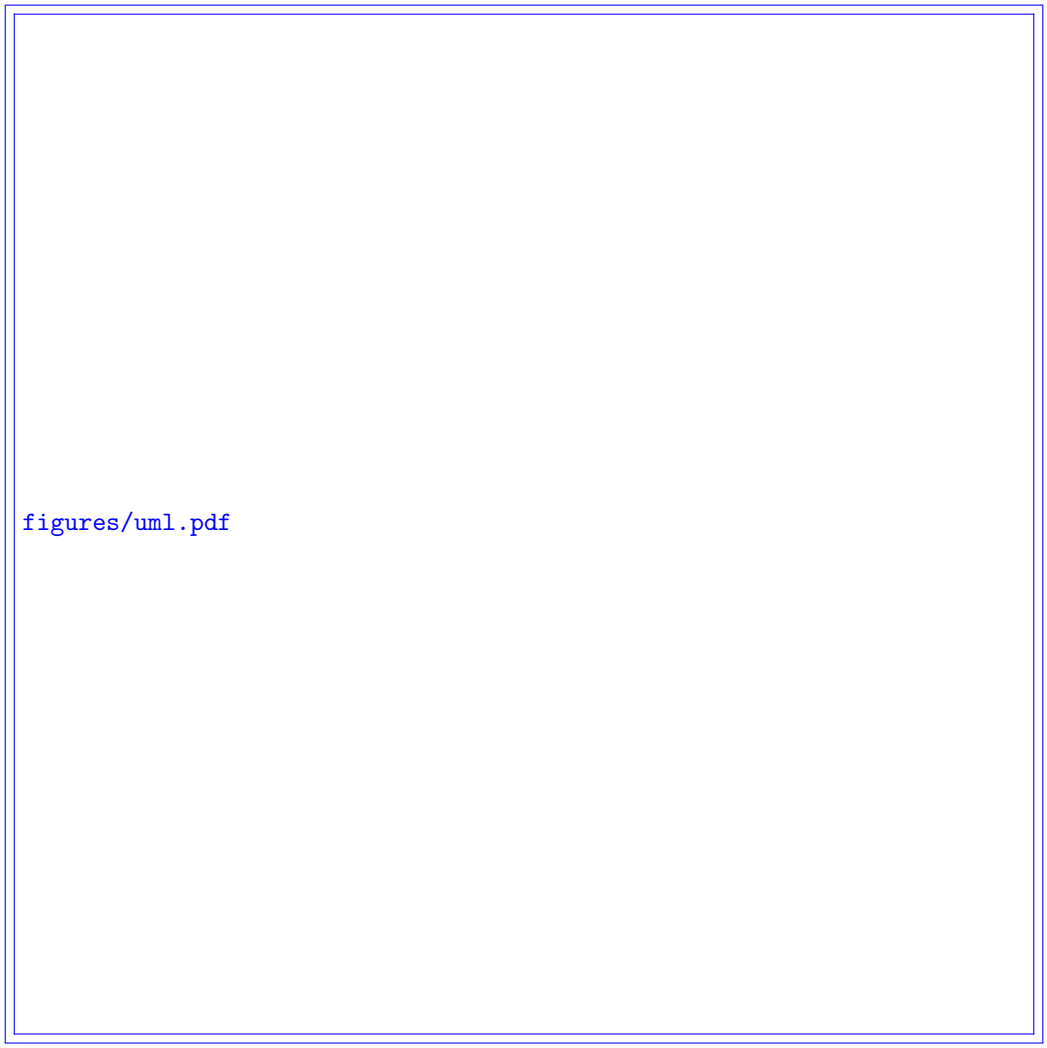
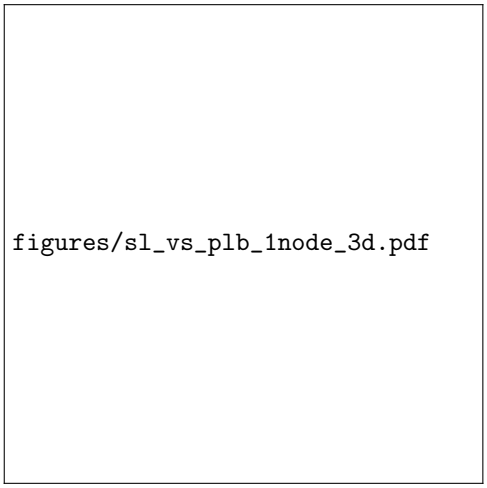


Figure 2: NATriuM's modular concept. The class `CFDSolver<dim>` manages the interplay of the `Stencil`, `AdvectionOperator<dim>`, and `CollisionModel` to simulate a flow that is defined in the `ProblemDescription<dim>`. The latter owns an instance of `BoundaryCollection<dim>` that defines each boundary conditions as a `Boundary<dim>` object. The `AdvectionOperator<dim>` performs the streaming step, which can be done using a `TimeIntegrator`. Italic text indicates abstract classes that are specialized by other classes, e.g., `Stencil` is specialized by `D2Q9`, `D3Q15`, `D3Q19`, and `D3Q27`.

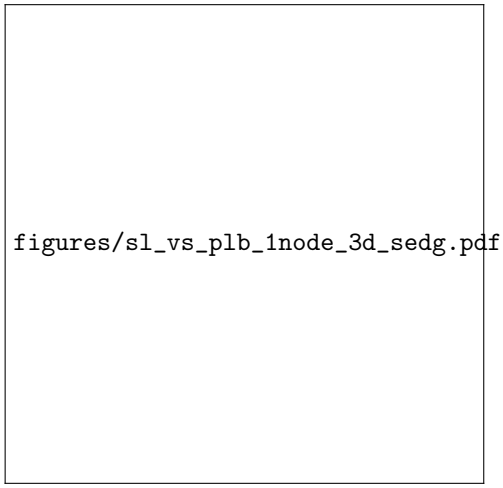
The `Stencil` contains the set of discrete velocities, weights, and a moment matrix. Currently, NATriuM ~~support~~ supports the most important velocity stencils: D2Q9, D3Q15, D3Q19, and D3Q27.
The collision model performs the collision step and, in doing so, determines the physics of the simulation. Each collision model is derived from the abstract class `CollisionModel` and has to implement its virtual method ~~collideAll~~`collideAll()`. Most collision models can be formulated generally so that they are valid for all stencils. Yet, it has often proven beneficial to provide optimized stencil-specific codes to speed up the collision step. In the current version, NATriuM supports the BGK scheme, a recent entropic multiple-relaxation-time scheme by Karlin et al. (cf. [**?**]), a preconditioned scheme for steady simulations [**?**], and a newly developed pseudo-entropic model [**?**].
The advection operator performs the spatial discretization for the streaming step on an arbitrary quadrilateral or hexagonal grid. Currently, two different advection operators are implemented: the discontinuous Galerkin scheme described in [**?**] and the recently developed semi-Lagrangian streaming step [**?**]. The time integrator performs the temporal discretization. NATriuM supports various explicit, implicit, and embedded Runge-Kutta time stepping as well as an exponential integrator. The modular structure of NATriuM allows ~~to easily incorporate~~ easy incorporation of new stencils, collision models, advection schemes and time integrators.

4.2. Results and Discussion

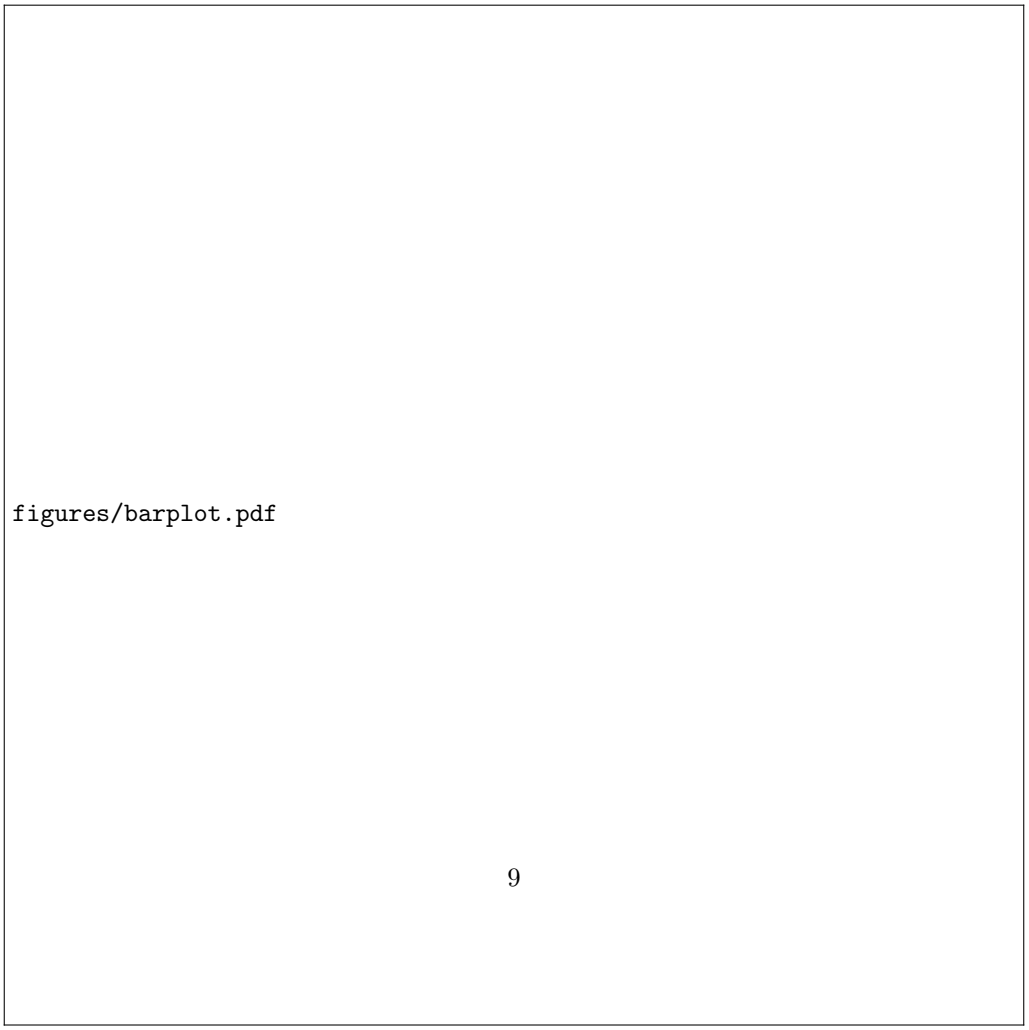
Several examples could appear here to demonstrate the use of interchangeable modules within NATriuM. The most important ~~choice the user has to make is on~~ user choice is which advection operator ~~he wants to use. Therefor~~ should be used. Therefore we present a performance comparison of the two implemented advection operators. It is clear that both the discontinuous Galerkin and the semi-Lagrangian advection step complicate the streaming, compared to the standard LBM. In the following study, both schemes are compared to Palabos[**?**], a well-known standard LBM library.
The performance of LBM codes is usually measured in million lattice updates per second (MLUPS). Although NATriuM does not operate on lattices, we still use the same measure, replacing the number of lattice nodes by the number of cells times the number of grid points owned by each cell.
The simulations for this comparison operated on a D3Q19 stencil with a BGK collision scheme. The discontinuous Galerkin advection was performed with different time integrators, most importantly a classical fourth-order Runge-Kutta method. All simulations used MPI for the parallelization and ran on two Intel Xeon E5-2680v3 with 128 GB DDR4 RAM. The three-dimensional simulation domain had periodic boundary conditions in all directions. The computational grid had cells of equal size. The number of cells and order of finite element p were varied.
Figures **??** and **??** depict the performance of NATriuM when used with a semi-Lagrangian streaming step and the discontinuous Galerkin streaming step that was introduced by Min and Lee [**?**]. The semi-Lagrangian ~~streaming step is~~ lattice Boltzmann method reached 10 MLUPS, which was 5 to 10 times faster than the discontinuous Galerkin scheme with classical Runge-Kutta time stepping. In addition, the performance ~~does~~ did not decrease as quickly ~~when~~ for growing p~~grows~~. As expected, the number of grid point updates per second ~~is~~ was significantly lower than for standard LBM codes. ~~For~~ By comparison,



(a) NATriuM's performance with semi-Lagrangian streaming at different orders of finite element p and grid sizes (the semi-Lagrangian streaming does not require a time integrator).



(b) NATriuM's performance with discontinuous Galerkin streaming and a classical fourth-order Runge-Kutta method at different orders of finite element p and grid sizes.



(c) Performance of semi-Lagrangian LBM vs discontinuous Galerkin LBM with different time integrators for a fixed problem size of 110 592 and second-order finite elements. From left to right: semi-Lagrangian streaming; (explicit Runge-Kutta methods:) explicit Euler, third-order Runge-Kutta, classical fourth-order Runge-Kutta; (implicit Runge-Kutta methods:) implicit Euler, implicit midpoint, Crank-Nicolson, singly-diagonally implicit Runge-Kutta; exponential integrator [**?**]; (embedded Runge-Kutta method:) Cash-Karp.

Palabos reached 140 MLUPS on the same hardware. The degraded performance of OLBMs is ~~clear from their formulation and~~ due to increased code complexity and memory access patterns and might still be acceptable ~~,~~ because they can reduce the number of required grid points by multiple orders of magnitude, cf. the next two Sections.
Figure **??** compares the semi-Lagrangian LBM and the discontinuous Galerkin LBM with various different time integrators, including explicit, implicit, and embedded Runge-Kutta methods as well as an explicit integrator. Embedded Runge-Kutta methods ~~like~~ such as the Cash-Karp method adapt the time step size dynamically to the problem during the course of the simulation. When using embedded Runge-Kutta methods, NATriuM automatically adjusts the relaxation time τ to the current time step size. The exponential integrator generally follows a previous publication [**?**].
The performance of NATriuM with a discontinuous Galerkin time stepping varied strongly with respect to the time integrator. The semi-Lagrangian LBM was faster than the discontinuous Galerkin LBM with all time integrators. It should be noted that the performance of implicit time integrators depends strongly on the time step size. In spite of their low number of less than one million lattice updates per second, they can still be useful when the problem allows large CFL numbers

$$CFL := \frac{\min \delta_x}{\max ||e_\alpha|| (p+1)^2 \delta_t}.$$

Prominent examples ~~for~~ of such problems are low-Re flows, stationary flows and heterogeneously refined-grids. In these situations, the implicit integrators, as well as the semi-Lagrangian streaming step, can exploit the fact that they do not incorporate a strict CFL constraint, cf. Section **??**.
In sum, this performance study advertises the use of our newly developed semi-Lagrangian streaming step within NATriuM. Moreover, it demonstrates ~~that~~ how the extensible code structure can boost the incorporation of new modules, e.g. advection operators and time integrators~~, can be easily incorporated into the existing modular code structure~~.

5. High-Order Discretization

5.1. Description

Both advection operators that are currently implemented in NATriuM can be used with an arbitrary order p of finite elements, allowing the spatial convergence order to be increased. This is a useful feature to attain highly accurate results. A high spatial order can also facilitate a drastic reduction of grid points, while retaining the same accuracy.

5.2. Results and Discussion

The Taylor-Green decaying vortex is a standard benchmark to test a solver's convergence. In the doubly-periodic domain $[0, 2\pi]^2$, the incompressible solution is given by

$$u_1^{\text{ref}}(x, y, t) = \sin(x)\cos(y)\exp(-2\nu t),$$
$$u_2^{\text{ref}}(x, y, t) = -\cos(x)\sin(y)\exp(-2\nu t),$$

where ν is the kinematic viscosity. To demonstrate the convergence, the spatial resolution was varied at constant Reynolds, Mach and CFL number $Re = \frac{UL}{\nu} = \frac{2\pi}{\nu} :=$

Figure 4: Convergence of the errors for the two-dimensional Taylor-Green vortex. $Ma := 10^{-4}$ and $CFL = 0.1$ were kept constant and the order of finite elements and grid spacing were varied.

$10, Ma = 10^{-4}, CFL = 0.1$, respectively. The ~~Errors~~ errors $||u - u^{\text{ref}}||_2$ were measured at ~~$t_{\text{max}} := TBD.$~~ $t_{\text{max}} := -\ln(0.1)/(2\nu)$, i.e. the eddy-turnover time.
Figure **??** shows the convergence with respect to the order p of shape functions and cell width δ_x. The errors decay with increasing p and with decreasing δ_x~~, respectively~~. The dependence on p is exponential, as in previous studies [**?** **?**].
This result underlines the theoretical finding that the spatial convergence order of OLBMs depends solely on the discretization of the streaming step. The advection operators implemented in NATriuM support high-order spatial convergence. As demonstrated by the above result, this convergence property can give highly accurate results even on coarse grids.

6. Irregular Grids

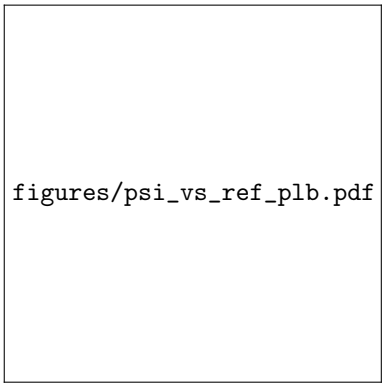
6.1. Description

NATriuM can handle arbitrary quadrilateral and hexagonal grids in 2D and 3D, respectively. It can both ~~read in grids from~~ import grids of different formats and generate (or manipulate) them ~~by~~ using deal.II's grid generating functions. Internally, the grids are stored as distributed octrees via the library p4est, yielding cache-efficient and highly-parallel data structures. NATriuM writes out checkpoints during a simulation. When restarting the simulation from a checkpoint with a differently refined grid, it automatically interpolates the distribution functions from the previous grid to the present grid. The possibility to use non-Cartesian grids is one of the major advantages of NATriuM over other lattice Boltzmann tools.

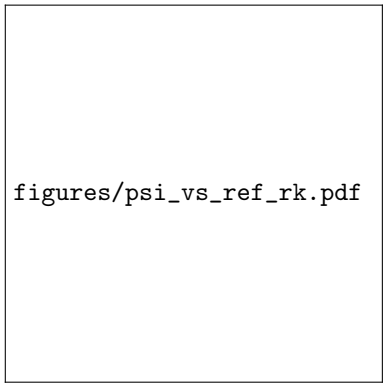
Table 1: Sinusoidal shear flow configurations that were studied.

Conf.	L	a
1	5	0.05
2	1	0.05
3	5	0.1
4	1	0.1

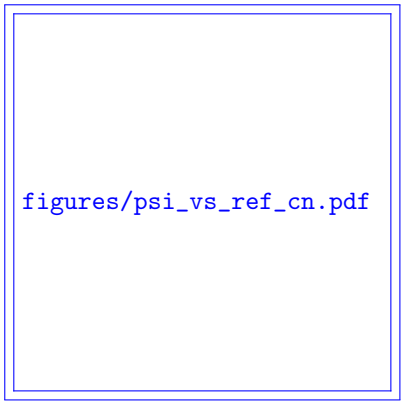
Figure 5: Computation grid for sinusoidal shear flow with $4 \cdot 4^3$ cells. The depicted sinusoidal geometry has an average height of $\bar{h} = 0.3$, an amplitude of $a = 0.1$ and a wavelength of $L = 1$, i.e. Conf. 4 in Table ??.



(a) Shear flow factors versus runtime for Palabos (standard LBM) simulations in sinusoidal geometries.



(b) Shear flow factors versus runtime for NATriuM (OLBM) simulations in sinusoidal geometries using an explicit fourth-order Runge-Kutta method with $CFL = 0.4$



(c) Shear flow factors versus runtime for NATriuM(OLBM) simulations using an implicit time integrator, the Crank-Nicolson method with $CFL = 100$. The fastest simulations took only a few seconds and gave very accurate results for all configurations.

Figure 6: Shear flow factors for different resolutions and orders of finite elements. The four configurations Conf. 1-4 are specified in Table **??**. Results are compared to the analytic results by Letalleur [**?**].

6.2. Results and Discussion

~~Shear flow factors for simulations with NATriuM using an implicit time integrator, the Crank-Nicolson method with $CFL = 100$. The fastest simulations took only a few seconds and gave very accurate results for all configurations.~~

To assess the benefits of arbitrary quadrilateral grids over regular lattices, both NATriuM and the standard LBM library Palabos ~~[?]~~ were used to simulate shear flows in sinusoidal geometries. These flows were simulated previously by Al-Zoubi and Brenner [**?**]. The upper wall was one period of a sine wave with wavelength L. The average height was $\bar{h} = 0.3$ and the amplitude a, making a flow domain

$$D_{\sin} := \{(x, y) : \ 0 \leq x < 2\pi, \ 0 \leq y \leq \bar{h} + a \sin(2\pi x/L)\},$$

cf. Figure **??**. The curved wall was kept stationary, while the lower moved with $u_w = 0.1$, with periodic boundaries along the x-axis. Four different channel configurations Conf. 1 - 4 were simulated (cf. Table **??**) to evaluate the shear flow factors

$$\Psi_x = \frac{\sqrt{2} \cdot \bar{h}}{a} \left(2 \cdot \frac{u_w - \bar{u}_x}{u_w} - 1 \right).$$

For long channels, inertial effects can be disregarded to give an analytic expression for the shear flow factor [**?**]

$$\Psi_x = \frac{3\sqrt{2}\frac{a}{\bar{h}}}{2 + \left(\frac{a}{\bar{h}}\right)^2}.$$

The simulations were stopped when the flow factor converged below a threshold of $|\Psi_x^{(i)} - \Psi_x^{(i-100)}| < 10^{-6}$, where i denotes the iteration number. The Reynolds and Mach number were set to 1 and 0.01, respectively.

For Palabos, the domain was $[0, L] \times [0, \bar{h} + a]$ with a regular grid. The flat wall was implemented with the regularized BC [?], while the curved wall and the solid part were composed of simple bounce-back nodes. The number of grid points per unit square was 4^N, $N = 5, \ldots, 10$. An average over all bulk nodes (boundaries excluded) was used to determine $\bar{u}_x$.

For NATriuM, the discontinuous Galerkin scheme was used ~~with~~ on a grid with $4 \cdot 4^N$ cells, $N \in \{2, 3, 4\}$. The order of finite elements $p \in \{2, 4, 6, 8\}$ was also varied. A classical fourth-order Runge-Kutta method was used for time integration, with a CFL number of 0.4.

It is expected that, in long channels, the shear flow factors approach the analytic solution. In short channels, a vortex may form inside the bulge. This effect decreases the throughput, leading to an increase of shear flow factors. Figure **??** shows the shear flow factors versus runtime for simulations with different spatial resolutions.

With Palabos, only for the highest resolution did the shear flow factors ~~agreed~~ agree with the analytic solution. The required runtime to get acceptable results was roughly 10^3 s. With NATriuM, however, ~~already~~ even the smallest discretization delivered accurate results. The sufficiency of a coarse, irregular grid lowered the required runtime by two orders of magnitude compared to Palabos, demonstrating the advantage of unstructured grids. NATriuM could ~~even~~ further improve over these results by using an implicit time

integrator: Figure **??** (right) shows the results for the same simulation setup with Crank-Nicolson time stepping and $CFL = 100$. Due to the large time step, it was possible to be even three orders of magnitude faster than Palabos while still giving accurate results. The advantage of implicit over explicit OLBMs for steady flows accords with recent work by Li and Luo [**?**].
In sum, NATriuM was much more efficient than Palabos ~~,~~due to the use of geometry-adaptive quadrilateral grids and implicit time integration. This result illustrates the usefulness of OLBMs. ~~This~~ The advantages of irregular ~~lattices~~ grids may well been ~~transferred~~ transferable to other flows in complex-shaped domains.

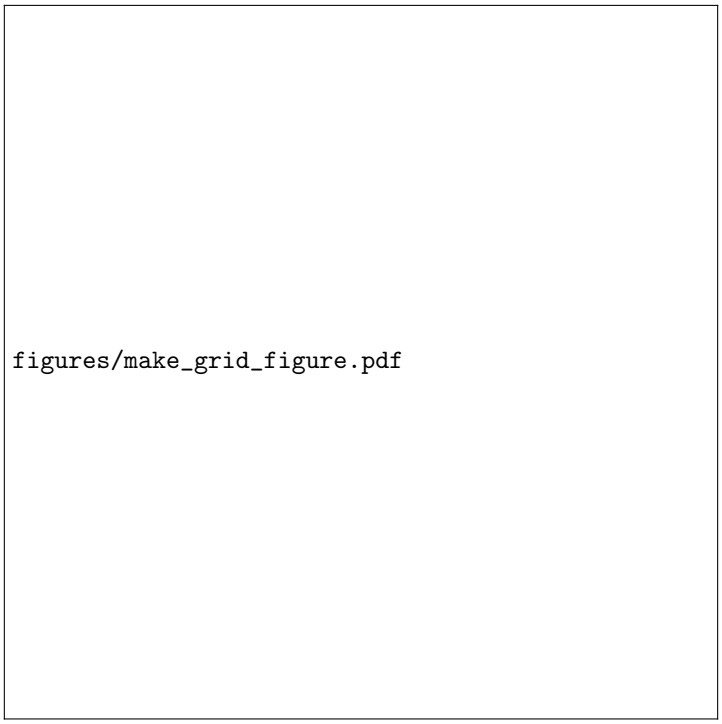
7. Dimension-Independent Programming

7.1. Description

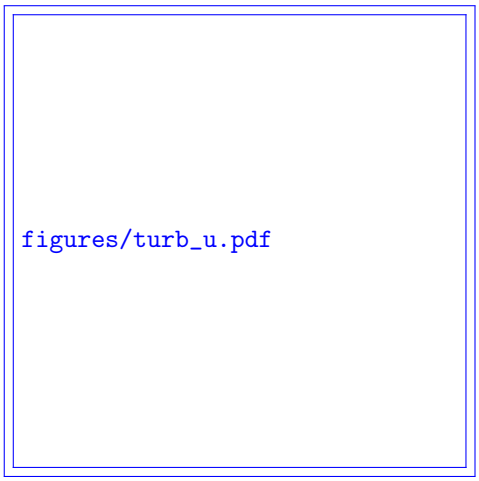
NATriuM supports simulations in 2D and 3D that are largely based on the same programming code. The massive use of C++ templates facilitates dimension-independent programming, where most classes take the dimension as a template parameter. Deal.II uses the same concept~~that~~ , which leads to a significant reduction of code. Most functionality in NATriuM is easily transferred from 2D to 3D, preventing duplicate code and obviating bugs.

7.2. Results and Discussion

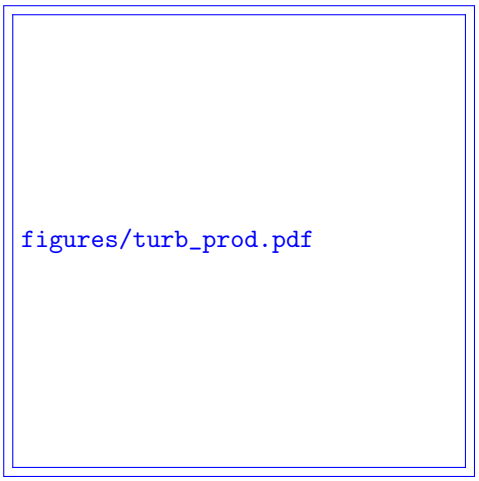
To support this feature by tangible results, we present the simulation of a three-dimensional turbulent channel flow. The computational domain $D = [0, 2\pi] \times [0, 1] \times [0, 2\pi/3]$ was divided into $32 \times 32 \times 32$ cells that were refined at the solid walls (cf. ~~Figure~~ Fig. **??**). The flow was driven by an external force with ~~a prescribed $Re_\tau = 180$~~ $Re_\tau = 140$. Periodic boundary conditions were used into the x and z directions~~and the flow was initialized with a turbulent profile~~.
The collision step was performed using the BGK scheme. The streaming step was performed using a spectral element discontinuous Galerkin scheme and an exponential integrator with $CFL = 2.0$ and $Ma = 0.05$. After an equilibration (400 000 time steps), the turbulence statistics were averaged over every tenth of 430 000 time steps ~~,~~and over the homogeneous directions.
Unfavorable initial conditions were used to check the robustness of the implementation, in combination with a short but still sufficient time period for calculating the statistical averages. A minimum channel configuration was used instead of the typically larger domain size, to demonstrate the stability of the present OLBM. (Notably, the turbulent simulations ran stable even on a very coarse grid with $8 \times 8 \times 8$ cells). The following results were obtained on a grid with $32 \times 32 \times 32$ cells.
Figure **??** shows some ~~statistics~~ statistical averages of this simulation, compared to the ~~reference~~ references of Moser et al. [**?**] ~~.~~and Tsukahara et al. [**?**] that were obtained at Reynolds numbers $Re_\tau = 180$ and $Re_\tau = 150$, respectively. The average streamwise velocities (~~Figure~~ Fig. **??**) agreed well with ~~the reference velocities~~ [**?**] and reproduced the viscous sublayer and the log-layer. ~~The viscous and Reynolds stresses at the wall were also in close match (Figure **??**). The~~ The production of turbulence kinetic energy (~~Figure **??**) was slightly underestimated at the peak, i. e. $y^+ \approx 10$, but accurate otherwise. Note that all statistics are given with respect to the actual $Re_\tau \approx 140$, which explains the~~



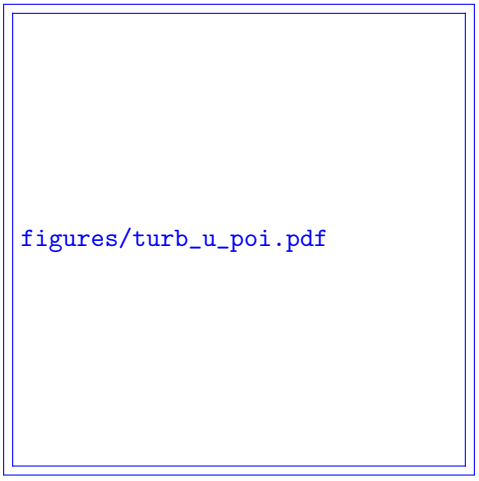
(a) Irregular computational grid for simulations of the turbulent channel flow. The plot shows one slice of cells from the lower wall ($y = 0$) to the channel center ($y = 0.5$). The grid has $32 \times 32 \times 32$ cells. The cells are equidistant in the x- and z-directions.



(a) Average streamwise velocities in wall units compared to [**?**]. In the reference, the wall Reynolds number was $Re_\tau = 180$. The logarithmic law of the wall ($u_1^+ = 0.41 \cdot \ln(y^+) + 5.5$) and y^+ are shown as a guide to the eye.



(b) Average production of kinetic energy in wall units compared to [**?**]. In the reference, the wall Reynolds number was $Re_\tau = 180$. The production is defined as TBD.



(c) Average streamwise velocities, compared to [**?**]. In the reference, the wall Reynolds number was $Re_\tau = 150$.

Figure 8: ~~Simulation of a~~ Results for the turbulent channel flow at ~~$Re_\tau = 180$. The results are~~ $Re_\tau = 140$, compared to ~~the reference by Moser et al.~~ references [**?**] and [**?**].

~~deviations in the center of the channel. (The considerably lower actual Re_τ is due to inaccurate initial conditions and a rather short equilibration.) While~~ Fig. ??) also matched the reference data [?]. While ~~While~~ the simulation by Moser et al. [?] used $128 \times 129 \times 128$ grid points, starting at $y^+ = 0.053$, the present simulation used only $64 \times 64 \times 64$ grid points, starting at $y^+ = 0.33$. In spite of the low resolution~~and the inaccurate Reynolds number~~, the major characteristics of the turbulent channel flow were captured.
This result demonstrates that the use of irregular grids in the lattice Boltzmann method can dramatically reduce the number of grid points, cf. [? ?] for comparison. Furthermore, it underlines the capability of NATriuM to simulate three-dimensional flows on irregular grids using the lattice Boltzmann method.

8. Conclusions

This paper presents a survey of our new off-lattice Boltzmann solver NATriuM. NATriuM is a modular and parallel C++ code that facilitates the simulations of flows on irregular grids using the lattice Boltzmann method. The possibility of using arbitrary quadrilateral and hexagonal grids is achieved by replacing the streaming step by a more sophisticated procedure, such as a semi-Lagrangian or a discontinuous Galerkin streaming. As demonstrated in the simulations within this manuscript, the use of irregular grids as well as high-order spatial discretizations can be a decisive advantage, especially when it comes to modeling wall-bound flows. These features help to greatly reduce the number of required grid points, offsetting the additional cost that is introduced by the sophisticated streaming step. NATriuM's scalability and extensibility make it an ideal tool to explore the future prospects of off-lattice Boltzmann methods and their potential to study complex scientific and industrial flows.

9. Acknowledgments

The authors thank Natan Zawadzki for implementing the turbulent channel flow, Siamak Bayat for contributing to the 3D implementation, ~~and~~ Vincent Frisch and Tobias Fraatz for assistance with the Palabos simulations, Thomas Brandes for helpful discussions about the efficiency and parallelization~~.~~, Anne Wegner for proofreading the manuscript, and Rudolf Berrendorf and Javed Razzaq for assistance with BRSU's WR cluster.

Appendix A. Splitting the discrete Boltzmann equation into collision and advection step

A pivotal prerequisite for NATriuM is that the solution of the discrete Boltzmann equation can be split into streaming and collision without affecting the spatial accuracy. To prove this ~~preliminary~~prerequisite, we integrate the discrete Boltzmann equation (??) along characteristics, giving

$$\begin{aligned} f_\alpha(t+\delta_t, x+\delta_t e_\alpha) - f_\alpha(t,x) &= \int_t^{t+\delta_t} \Omega_\alpha(f)dt \\ &= \frac{\delta_t}{2}\left[\Omega_\alpha\big(f(t,x)\big) + \Omega_\alpha\big(f(t+\delta_t, x+\delta_t e_\alpha)\big)\right] + O\left(\delta_t^2\right). \end{aligned}$$

This implicit equation is transformed into an explicit equation by $\bar{f}_\alpha := f_\alpha - \frac{\delta_t}{2}\Omega_\alpha(f)$:

$$\bar{f}_\alpha(t+\delta_t, x+\delta_t e_\alpha) - \bar{f}_\alpha(t,x) = \delta_t \Omega_\alpha\big(f(x,t)\big) + O\left(\delta_t^2\right). \tag{A.1}$$

The final step is to express the collision operator in terms of $\bar{f}$. ~~Therefor~~ Therefore the latter is required to take the form

$$\Omega(f) = Rf + R^{\mathrm{eq}}(f),$$

where R is linear and $R^{\mathrm{eq}}(\bar{f}) = R^{\mathrm{eq}}(f)$. This form includes the most important collision operators in the literature, most importantly the BGK model. Rewriting

$$\Omega(f) = Rf + R^{\mathrm{eq}}(f) = R\left(\bar{f} + \frac{\delta_t}{2}\Omega(f)\right) + R^{\mathrm{eq}}(\bar{f}) = \Omega(\bar{f}) + \frac{\delta_t}{2}R\Omega(f),$$

we get

$$\Omega(f) = \left(I - \frac{\delta_t}{2}R\right)^{-1}\Omega(\bar{f}).$$

Inserting into Equation (**??**) gives the difference equation

$$\bar{f}_\alpha(t+\delta_t, x+\delta_t e_\alpha) - \bar{f}_\alpha(t,x) = \delta_t\left(I - \frac{\delta_t}{2}R\right)^{-1}\Omega(\bar{f}) + O\left(\delta_t^2\right),$$

that is easily split into collision and advection; and for the ~~rest of the~~ whole paper (except Equation (**??**)) we redefine

$$f := \bar{f}$$
$$\text{and} \quad \Omega := \left(I - \frac{\delta_t}{2}R\right)^{-1}\Omega.$$

Note that for the BGK collision term, this transformation leads to the well-known 0.5-shift in the relaxation time.

Appendix B. Semi-Lagrangian Streaming Step

The semi-Lagrangian streaming step is discussed in depth in a separate publication [**?**]. It represents the distribution functions by a finite element instead of a point-wise approximation. The finite element representation uses Lagrangian tensor-product shape functions $\tilde{\psi}_i$ that interpolate the function values at the Gauß-Lobatto-Legendre support points [**?**] $\tilde{x}_i$, $i = 1, \ldots, (p+1)^{\dim}$, of a unit cell $\tilde{D} := [0,1]^{\dim}$, where p is the polynomial order. The shape functions and support points are transferred to an arbitrary quadrilateral grid cell D_c using a multilinear mapping function $M_c : D_c \to \tilde{D}$, yielding $x_{ci} := M_c^{-1}(\tilde{x}_i)$ and

$$\psi_{ci}(x) := \begin{cases} \tilde{\psi}_i(M_c(x)), & \text{if } x \in D_c \\ 0, & \text{else.} \end{cases}$$

The distribution functions are approximated by

$$f_\alpha(x,t) \approx \sum_{c=1}^{\#\text{cells}} \sum_{i=1}^{(p+1)^{\dim}} \hat{f}_{\alpha ci}(t)\psi_{ci}(x),$$

where $\hat{f}_{\alpha ci}$ are the degrees of freedom. Applying this to the lattice Boltzmann equation at $x = x_{ci}$ gives the semi-Lagrangian streaming

$$\hat{f}_{\alpha ci}(t) = \sum_{\zeta=1}^{\#\text{cells}} \sum_{j=1}^{(p+1)^{\dim}} \hat{f}_{\alpha\zeta j}(t-\delta_t)\ \psi_{\zeta j}(x_{ci} - \delta_t e_\alpha).$$

References