

# ADVANCED AUTOMATIC CODE GENERATION FOR MULTIPLE RELAXATION-TIME LATTICE BOLTZMANN METHODS

FREDERIK HENNIG\*, MARKUS HOLZER\*†, AND ULRICH RÜDE\*†

**Abstract.** The scientific code generation package *lbmpy* supports the automated design and the efficient implementation of lattice Boltzmann methods (LBMs) through metaprogramming. It is based on a new, concise calculus for describing multiple relaxation-time LBMs, including techniques that enable the numerically advantageous subtraction of the constant background component from the populations. These techniques are generalized to a wide range of collision spaces and equilibrium distributions. The article contains an overview of *lbmpy*'s front-end and its code generation pipeline, which implements the new LBM calculus by means of symbolic formula manipulation tools and object-oriented programming. The generated codes have only a minimal number of arithmetic operations. Their automatic derivation rests on two novel Chimera transforms that have been specifically developed for efficiently computing raw and central moments. Information contained in the symbolic representation of the methods is further exploited in a customized sequence of algebraic simplifications, further reducing computational cost. When combined, these algebraic transformations lead to concise and compact numerical kernels. Specifically, with these optimizations, the advanced central moment- and cumulant-based methods can be realized with only little additional cost as when compared with the simple BGK method. The effectiveness and flexibility of the new *lbmpy* code generation system is demonstrated in simulating Taylor-Green vortex decay and the automatic derivation of an LBM algorithm to solve the shallow water equations.

**Key words.** Lattice Boltzmann Method, Metaprogramming, Code Generation

**MSC codes.** 65Y20, 82C40

**1. Introduction.** Modern scientific computing is characterized by growing complexity on several fronts. On the one side, the mathematical models become more advanced and powerful so that the corresponding numerical methods and algorithms grow increasingly intricate and involved. At the same time, contemporary computer architectures become more difficult to program. This is especially true in high end parallel computing that is characterized by rapidly evolving new hardware architectures, such as general-purpose graphics processing units (GPUs).

Thus the central step of scientific computing, i.e., the step from a model to executable software, faces increasing challenges. The developers of algorithms and scientific software are squeezed in from two sides: Increasingly complex simulation models must be realized in software for increasingly complex advanced supercomputer systems. We note here also that legacy software often lacks performance portability. For example it may present an almost prohibitive effort to port a legacy flow solver to a modern multi-GPU system. It may be speculative, but likely enormous computational resources are underused world wide, since outdated software is underperforming on the given computer systems on which it runs.

At the same time, we observe that modern mathematical methods have evolved to a state where their derivation and analysis depends increasingly on researchers making use of symbolic formula manipulation packages. For the present article, we will consider the class of lattice Boltzmann methods (LBMs), whose most advanced variants could hardly be derived without using symbolic mathematical software. Given this situation, it is a natural step to integrate the algebraic manipulations routinely into the workflow of designing and implementing these methods. A first step of this

---

\*Chair for System Simulation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, 91058 Germany ([frederik.hennig@fau.de](mailto:frederik.hennig@fau.de)),

†CERFACS, 42 Avenue Gaspard Coriolis, 31100 Toulouse, France ([holzer@cerfacs.fr](mailto:holzer@cerfacs.fr)).

is presented for the *lbmpy* package by Bauer et al. in [5].

We emphasize that the effect is twofold: In this approach, the derivation of the methods is consolidated by giving method developers a powerful tool to describe, analyze, and experiment with a range of method variants. But at the same time, the automated symbolic derivations can be used beyond just deriving the discrete approximations. Conventionally, this would then only be another set of mathematical formulas that still have to be transformed into software manually. Here, instead, the symbolic manipulation will be taken a step further to produce operational code directly. Through this automatic code generation approach, numerical kernels can be produced targeting different architectures (such as CPUs or GPUs), using different data structures, and also different memory layouts.

Furthermore, symbolic manipulation offers the possibility to optimize the codes in various ways, both mathematically by simplifying expressions, but also with hardware aware transformations, e.g., with the goal of providing an automatic vectorization. A specific advantage here is that the optimizations can go much beyond traditional optimizing or vectorizing compilers. The domain-specific code generator may leverage information contained in the numerical method's symbolic form to apply highly specific simplifying transformations. This enables automatic optimizations that would otherwise be accessible only to an expert programmer who is at the same time a highly educated model developer —a situation which in our experience is rarely found in scientific computing practice. In *lbmpy* this rare synthesis of expertises is encapsulated within the automatic code generation package itself.

The LBM is a mesoscopic method for the simulation of transport phenomena, originally developed as an extension of lattice gas automata [7]. More modernly it is derived by the discretization of the Boltzmann equation [28]. It discretizes the continuum of a fluid on a cartesian lattice, modelling its state by the local distribution of particle velocities at the lattice sites. The first application of the LBM was as a solver of the Navier-Stokes equations (NSE), but the LBM has also been applied to a variety of problems. These include the advection-diffusion equation [28], thermal flows [10], the shallow water equations [54, 43, 49], as well as multiphase and multicomponent flows [28, 14, 26, 21, 47].

At the core of the LBM algorithm lies the *collision* step, where the local particle distributions are relaxed towards a given equilibrium distribution. The equilibrium state depends on the macroscopic physics that the method is meant to simulate, and thus differs significantly between applications. The classical variant of the relaxation process is the discrete BGK collision operator [6, 7, 28], which employs a single relaxation rate. It has been generalized to multiple relaxation-time (MRT) collision operators, relaxing different statistical moments of the distributions with individual rates, thus improving the representation of simulated physics, as well as numerical stability [9, 20, 15, 19, 28]. Originally employing raw moments as a collision space, MRT methods were later extended to central moment [15, 39, 44] and cumulant [19] space.

As a complex numerical method, the LBM is subject to floating-point roundoff error, which may pollute solutions when the floating point precision is insufficient and not properly controlled. Then the accuracy of the simulation may deteriorate [48, 16, 33]. Early in the development of LBMs it has been realized that, in hydrodynamic simulations, populations deviate only little from a background distribution typically given by the lattice weights [48]. Since this background distribution is invariant under collision, it may be subtracted from the population vector, thus significantly increasing the number of digits available for an accurate floating point presentation

of the populations. We use the term the *zero-centered* storage format for this algebraic transformation. It has been found to improve the LBM's numerical accuracy substantially [19, 16, 33].

Implementing all relevant variants of collision operators and application use cases in a single, generic software framework poses a significant challenge. The challenge becomes even greater when different hardware architectures require different optimizations. The creation of the modular and object-oriented software design of frameworks such as Palabos [31], OpenLB [27], TCLB [29], and waLBerla [3, 45, 46] is a key step to solve this problem. However, as elsewhere, the generality of a software often inhibits specific performance optimizations. An automatic code generation approach, as described above, constitutes a potentially more flexible alternative to an extensive but rigid hand-coded framework. This idea has been successfully employed in the context of classical finite element discretizations, e.g. in [40, 35], or for general stencil codes [36, 41, 23, 34]. Metaprogramming techniques can also be found in some of the aforementioned LBM frameworks [29, 27]. The original version of *lbmpy* adopts this paradigm in the form of a symbolic domain-specific language based on the raw moment MRT formalism. Its key functionality is the automatic generation of optimized implementations of single relaxation-time (SRT) [7], two relaxation-time (TRT) [20] and raw-moment MRT methods from their symbolic description.

The aim of this work is the extension of *lbmpy* to go beyond the existing functionality and support both central moment and cumulant collision spaces, to integrate the zero-centered storage format, and to provide the transformations needed to generate LBM codes for various applications beyond the Navier-Stokes equations. During the development process, we found the original structure of *lbmpy* to pose severe restrictions. Therefore it was necessary to re-design the original architecture of *lbmpy*. The outcome of this process will be presented in this article. It has three essential components:

First, we introduce a consolidated theoretical framework for modelling MRT methods. This defines a generalized formalism to specify the collision spaces and the equilibrium. Additionally, as a novel feature, this framework integrates the zero-centered representation of the distribution functions. In particular, we develop and present a generalization of zero-centered storage to collision spaces, and formalize its application to the equilibrium distribution.

Second, we present the front-end of *lbmpy* as an application programming interface (API) in the Python programming language for modelling LBMs on an abstract level. This constitutes a domain-specific language that closely reflects the theoretical framework of LBMs. It is implemented by means of the computer algebra package *SymPy* [37].

Third, we describe the modularized automatic procedure for symbolically deriving and optimizing the collision rule. This includes the presentation of two novel Chimera transforms between populations and the raw and central moment spaces, designed to minimize the number of arithmetic operation for the mappings between these spaces. We furthermore elaborate on an extensive collection of LBM-specific algebraic simplifications. Due to these improvements, our new code generator can produce efficient implementations of raw moment, central moment, and cumulant LBMs. As a surprising key result we find that the latter of these advanced LBMs require only little arithmetic overhead as compared to simple SRT and raw moment MRT methods. Thus, in many cases the advanced methods are expected to execute almost equally fast as the simpler ones.

In summary, our article presents a versatile and flexible framework for the mod-

elling and prototyping of complex LBM methods, combined with the techniques for the automatic generation of highly optimized computational kernels for these methods. The software framework itself is open-source and freely available under the GNU AGPLv3 license.

The remainder of this paper is structured as follows. In section 2, we introduce our theoretical framework and present our generalization of zero-centered storage. Section 3 presents the modelling front-end of *lbmpy*. Section 4 elaborates on the automatic derivation procedure for collision rules, including our novel raw and central moment Chimera transforms. The same section introduces the extensive simplification procedure applied by *lbmpy* and shows its effectiveness for optimizing collision kernels. Finally, we demonstrate the versatility of the revised framework for modelling advanced LBMs, its applicability to their rapid implementation and testing, and the numerical advantages of zero-centering in MRT methods in section 5. Section 6 concludes the paper.

**2. Theoretical Background.** This section presents the theoretical framework that serves as the basis of our code generation system. To this end, we first introduce our formalized approach to modelling populations and the zero-centered storage format. The multiple relaxation-time LBM collision process is introduced and generalized to arbitrary collision spaces. In particular, we discuss raw moment, central moment and cumulant spaces, as well as the reflection of zero-centered storage therein. Finally, we introduce a novel abstract interpretation of the equilibrium distributions.

**2.1. Discrete Lattice Structure and Storage Format.** The lattice Boltzmann method acts on a  $d$ -dimensional cartesian lattice, where a local particle distribution vector  $\mathbf{f}$  with  $q$  entries is stored on each lattice site. Entry  $f_i$  is understood to model the relative number of particles moving at a velocity  $\xi_i \in \{-1, 0, 1\}^d$ . We adopt writing  $\bar{i}$  to refer to the population index of  $f_{\bar{i}}$  moving in the direction opposite to  $f_i$ . The lattice structure is specified by the common D $d$ Q $q$  stencil notation. The two- and three-dimensional first-neighborhood stencils found most prevalently in literature are D2Q9, D3Q15, D3Q19 and D3Q27 [28]. Our work will also focus on these stencils. For illustration, we employ D2Q9 throughout this paper. D2Q9 comprises the full set of first-neighborhood velocities  $\xi_i \in \{-1, 0, 1\}^2$ ; while their exact ordering is not relevant to our discussion, we adopt  $\xi_0 = \mathbf{0}$ .

The lattice Boltzmann algorithm comprises the *streaming* step, where populations are advected according to their velocities; and the *collision* step, rearranging every node's populations by some collision operator  $\Omega$ . Often, the algorithm also includes some type of source term  $\mathbf{f}^F$ , which might be a momentum source modelling body forces acting on the fluid [22, 25], or a mass source in an advection-diffusion setting [28]. The LBM update equations read

$$(2.1a) \quad \mathbf{f}^*(\mathbf{x}, t) = \Omega(\mathbf{f}(\mathbf{x}, t)) + \mathbf{f}^F,$$

$$(2.1b) \quad f_i(\mathbf{x} + \xi_i, t + \Delta t) = f_i^*(\mathbf{x}, t).$$

In the remainder of this paper, we shall focus on the collision step; the streaming step will only be mentioned briefly in section 3.

The original and major application of the LBM is as an algorithm to simulate incompressible, Newtonian fluid flow as modelled by the NSE. In such simulations, the local distribution vector is found to deviate only little from a certain background distribution [48, 24, 19, 33]. We denote this background distribution  $\mathbf{f}^0$ ; its exact form will be discussed in subsection 2.4. This fact can be exploited to improve the

LBM's numerical accuracy: instead of storing the absolute values of the population vector, only its deviation component  $\delta \mathbf{f} = \mathbf{f} - \mathbf{f}^0$  can be stored per site. As noted in section 1, we denote this as the *zero-centered* storage format, to set it apart from the classical approach, which we dub *absolute* storage format. The zero-centered storage format may be less useful in non-hydrodynamic LBMs whenever the assumption of small deviations from the background distribution does not hold.

From the discrete distribution vector, a set of macroscopic quantities can be computed on each lattice site. The sum over a site's populations yields the local particle density  $\rho$  while their weighted mean becomes the macroscopic velocity  $\mathbf{u}$ , which are written as

$$(2.2) \quad \rho := \sum_{i=0}^{q-1} f_i, \quad \mathbf{u} := \frac{1}{\rho} \sum_{i=0}^{q-1} f_i \boldsymbol{\xi}_i$$

If a background distribution is given, the density can be decomposed as  $\rho = \rho^0 + \delta\rho$ , with

$$(2.3) \quad \rho^0 := \sum_{i=0}^{q-1} f_i^0, \quad \delta\rho := \sum_{i=0}^{q-1} \delta f_i, \quad \mathbf{u} := \frac{1}{\rho} \sum_{i=0}^{q-1} \delta f_i \boldsymbol{\xi}_i.$$

**2.2. The Collision Step.** The site-local LBM collision rule constitutes a relaxation process of the population vector against an equilibrium state. In the MRT paradigm, relaxation occurs in some collision space isomorphic to  $\mathbb{R}^q$ , wherein the equilibrium state is given by a vector  $\mathbf{q}^{\text{eq}}$ . The population vector is transformed to and from collision space by some bijective mapping  $\mathcal{T}$ . Moreover the source term may also be represented in collision space, denoted by  $\mathbf{q}^F$  [28, 11, 12, 21]. This leads to the general MRT collision equation

$$(2.4) \quad \begin{aligned} \mathbf{q} &= \mathcal{T}(\mathbf{f}) \\ \mathbf{f}^* &= \mathcal{T}^{-1} (\mathbf{q} + \mathbf{S} (\mathbf{q}^{\text{eq}} - \mathbf{q}) + \mathbf{q}^F). \end{aligned}$$

where  $\mathbf{S} = \text{diag}(\omega_0, \dots, \omega_{q-1})$  is the relaxation matrix, specifying relaxation rates separately for each quantity  $q_i$ . These quantities, and their equilibrium counterparts, usually correspond to different properties of the fluid, and their distinct relaxation is meant to independently model different physical processes [28, 19].

To take full advantage of zero-centered storage, we aim to express (2.4) in deviation components only. This is only straightforwardly possible if  $\mathcal{T}$  is linear. In this case, the decomposition of  $\mathbf{f}$  can be reflected in the MRT collision space, by  $\mathbf{q} = \mathbf{q}^0 + \delta\mathbf{q}$ , with  $\mathbf{q}^0 = \mathcal{T}(\mathbf{f}^0)$  and  $\delta\mathbf{q} = \mathcal{T}(\delta\mathbf{f})$ . Subtracting the background component, we also obtain the deviation component  $\delta\mathbf{q}^{\text{eq}} = \mathbf{q}^{\text{eq}} - \mathbf{q}^0$  of the equilibrium vector. We thus arrive at the deviation-only update equations

$$(2.5) \quad \begin{aligned} \delta\mathbf{q} &= \mathcal{T}(\delta\mathbf{f}), \\ \delta\mathbf{f}^* &= \mathcal{T}^{-1} (\delta\mathbf{q} + \mathbf{S} (\delta\mathbf{q}^{\text{eq}} - \delta\mathbf{q}) + \mathbf{q}^F). \end{aligned}$$

By definition, a source term cannot affect the constant background component; thus we can include  $\mathbf{q}^F$  in the deviation-only update without alteration.

For nonlinear  $\mathcal{T}$ , the background-deviation-decomposition can not be transformed into collision space in this manner. Instead, (2.4) is modified. The absolute population

vector is re-obtained by adding the background distribution before collision, so that it must be subtracted again, leading to

$$(2.6) \quad \begin{aligned} \mathbf{q} &= \mathcal{T}(\delta\mathbf{f} + \mathbf{f}^0), \\ \delta\mathbf{f}^* &= \mathcal{T}^{-1}(\mathbf{q} + \mathbf{S}(\mathbf{q}^{\text{eq}} - \mathbf{q}) + \mathbf{q}^F) - \mathbf{f}^0. \end{aligned}$$

In the following, we will introduce the common MRT collision spaces relevant to this work, and discuss the shape  $\mathcal{T}$  takes therein.

**2.3. Collision Spaces.** Assuming a  $DdQq$  stencil, we consider  $q$ -dimensional collision spaces spanned by either raw moments  $m$ , central moments  $\kappa$ , or cumulants  $C$  of the discrete population vector. We continue to use the symbol  $q$  in place of  $m$ ,  $\kappa$  and  $C$  whenever we discuss general properties of these statistical quantities. We furthermore adopt a method of specifying statistical quantities using polynomials in the variables  $x$ ,  $y$  and  $z$ . This notation is reflected exactly within the *lbmpy* transformation system. Given such a polynomial  $p$ , the respective quantity is denoted  $q_p$ . For clarity, we denote *monomial* quantities, defined by a monomial  $x^\alpha y^\beta z^\gamma$ , as  $q_{\alpha\beta\gamma}$ . A collision space based on a specific type of quantity is now defined through its *basis*, where a basis is a sequence  $(p_i)_{i=0,\dots,q-1}$  of linearly independent polynomials.

**2.3.1. Raw Moments.** Originally, the MRT methods based on raw moments were developed to overcome deficiencies of the discrete BGK operator [28, 9]. Among their advantages are the possibility to separate shear from bulk viscosity, and the ability to tune higher-order relaxation rates in order to improve stability and accuracy [28].

The raw moment-generating function  $M$  of a distribution  $f$  is defined using the multidimensional Laplace transform  $\mathcal{L}$  as

$$(2.7) \quad M(\boldsymbol{\Xi}) := \mathcal{L}\{f\}(-\boldsymbol{\Xi}).$$

Since  $\mathcal{L}$  is an integral operator,  $f$  must be integrable. Here the discrete distribution vector is made integrable by means of the Dirac delta distribution as  $f(\boldsymbol{\xi}) := \sum_i f_i \delta(\boldsymbol{\xi} - \boldsymbol{\xi}_i)$ . The monomial raw moments  $m_{\alpha\beta\gamma}$  of  $f$  are now defined as the mixed derivatives of  $M$ , evaluated at zero

$$(2.8) \quad m_{\alpha\beta\gamma} := \partial_1^\alpha \partial_2^\beta \partial_3^\gamma M(\boldsymbol{\Xi}) \Big|_{\boldsymbol{\Xi}=0}.$$

In the discrete case, this evaluates to a simple summation over the entries of the population vector. Polynomial raw moments  $m_p$  are obtained as linear combinations of their monomial components. This linear combination can be combined with the sum obtained from the generating function. Together, monomial and polynomial quantities may be computed as

$$(2.9) \quad m_{\alpha\beta\gamma} = \sum_i f_i \xi_{i,x}^\alpha \xi_{i,y}^\beta \xi_{i,z}^\gamma, \quad m_p = \sum_i f_i p(\boldsymbol{\xi}_i).$$

For a given basis  $(p_i)_i$ , the transformation to discrete raw moment space can now be represented as an invertible matrix  $\mathbf{M}$ . Thus,  $\mathbf{m}_p = \mathcal{T}(\mathbf{f}) = \mathbf{M}\mathbf{f}$  is a linear mapping, allowing us to reflect the background-deviation-decomposition in raw moment space.

**2.3.2. Central Moments.** LBMs based on raw moments violate Galilean invariance. To correct for this, MRT methods based on relaxing central moments of the discrete distribution function have been developed [15, 13, 39, 8]. In contrast to raw moments, central moments are taken with respect to the co-moving frame of reference, given by the macroscopic velocity  $\mathbf{u}$ .

The velocity shift is reflected in Laplace frequency space by multiplication with  $\exp(-\boldsymbol{\Xi} \cdot \mathbf{u})$ , which we use to define the central moment-generating function

$$(2.10) \quad K(\boldsymbol{\Xi}) := \exp(-\boldsymbol{\Xi} \cdot \mathbf{u}) M(\boldsymbol{\Xi}).$$

Its derivatives, in turn, give rise to the monomial central moments  $\kappa_{\alpha\beta\gamma}$ , from which polynomial central moments  $\kappa_p$  are again obtained as linear combinations. The shifted frame of reference appears again in the explicit equations via subtraction

$$(2.11) \quad \kappa_{\alpha\beta\gamma} = \sum_i f_i (\xi_{i,x} - u_x)^\alpha (\xi_{i,y} - u_y)^\beta (\xi_{i,z} - u_z)^\gamma, \quad \kappa_p = \sum_i f_i p(\boldsymbol{\xi}_i - \mathbf{u}).$$

These transform equations are again linear, giving rise to the central moment matrix  $\mathbf{K}$  which defines  $\mathcal{T}$  for the basis  $(p_i)_i$ .

**2.3.3. Cumulants.** Cumulants, other than any kind of moment, characterize a distribution's shape independent of any frame of reference [19]. They thus share the velocity-independence of central moments, but additionally introduce mutual statistical independence. Similar to moments, monomial cumulants  $c_{\alpha\beta\gamma}$  are the mixed derivatives of the cumulant-generating function at the origin. The cumulant-generating function is defined as the natural logarithm of the moment-generating function

$$(2.12) \quad C(\boldsymbol{\Xi}) := \log M(\boldsymbol{\Xi}), \quad c_{\alpha\beta\gamma} := \partial_1^\alpha \partial_2^\beta \partial_3^\gamma C(\boldsymbol{\Xi}) \Big|_{\boldsymbol{\Xi}=0}, \quad C_{\alpha\beta\gamma} := \rho c_{\alpha\beta\gamma}.$$

We follow Geier et al. [19] in defining the rescaled cumulants  $C_{\alpha\beta\gamma}$ . We give no explicit equations for monomial or polynomial cumulants in terms of populations; instead, we discuss a more practical approach to their computation in section 4. Due to the cumulant transform's nonlinearity, no reflection of the population's decomposition in cumulant space is possible.

**2.4. Equilibrium and Background Distributions.** The basis of most LBMs is a variant of the Maxwell-Boltzmann distribution [28] that defines the equilibrium. Depending on density  $\rho$ , velocity  $\mathbf{u}$ , and the speed of sound parameter  $c_s$ , its continuous form reads

$$(2.13) \quad \Psi(\rho, \mathbf{u}, \boldsymbol{\xi}) = \rho \left( \frac{1}{2\pi c_s^2} \right)^{3/2} \exp \left( -\frac{\|\boldsymbol{\xi} - \mathbf{u}\|^2}{2c_s^2} \right).$$

Since, in hydrodynamic simulations, the fluid density deviates only little from a background density  $\rho^0$ , we take as a background distribution the fluid rest state at  $\rho = \rho^0, \mathbf{u} = \mathbf{0}$ . We hence denote  $\Psi^0 = \Psi(\rho^0, \mathbf{0})$  and define the deviation component of the equilibrium distribution as  $\delta\Psi = \Psi - \Psi^0$ . The background density is typically set to unity. Figure 2.1 illustrates the relation between  $\Psi$ , its background and deviation component, with density and velocity chosen from the typical simulation range. We observe a large difference in magnitude between background and deviation component; overall, the background component dominates the distribution.

Representations of  $\Psi$  and  $\Psi^0$  in raw moment, central moment, and cumulant space can be obtained by algebraically computing and differentiating the respective

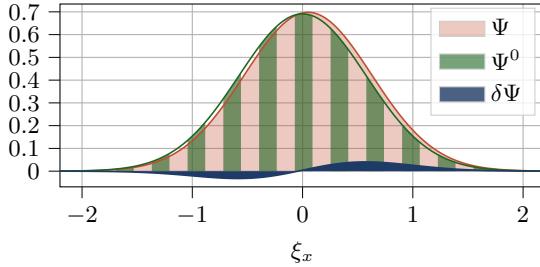


FIGURE 2.1. Plot of  $\Psi$ ,  $\Psi^0$  and  $\delta\Psi$  in one dimension at  $\rho = 1.01$ ,  $u_x = 0.05$ .

generating functions. In case of raw and central moments, this reduces to the well-known moment integrals. The deviation component  $\delta\Psi$  has representations in both moment spaces, but not in cumulant space, since the cumulants of  $\delta\Psi$  are undefined at the singularity  $\rho = \rho^0$ , where its area under the curve is zero. This can be verified by expanding cumulants in terms of raw moments: Applying the chain rule to evaluate (2.12) produces equations containing divisions by  $\delta\rho$ .

Given e.g. a vector  $m^0$  of  $q$  independent raw moments of  $\Psi^0$ , the background distribution in population space may be obtained as  $f^0 = M^{-1}m^0$ . In the case of the standard hydrodynamic equilibrium, this vector reduces to the lattice weights  $w$ .

Alternatively the equilibrium state may also be specified as a discrete distribution  $f^{eq}$ . In this case, its representation in collision space is simply obtained as  $q^{eq} = \mathcal{T}(f^{eq})$ .

**3. MRT Method Modelling Frontend.** The main component our code generation system is an automatic procedure that takes an abstract LBM specification and derives from it a sequence of symbolic equations implementing one of the collision rules (2.4)–(2.6). This specification is formulated using a flexible Python API. Both the modelling API and the derivation system itself rely on the computer algebra package *Sympy* [37] to represent and manipulate all components of an LBM in symbolic, mathematical form. The automatic derivation and thus the software performing these transformations follows closely the theory of section 2. This section introduces the user front-end for modeling, while the next section will focus on the derivation and code generation procedures.

The parameter space of *lbmpy*'s abstract method specification is shown in Figure 3.1. The lattice structure is defined by selecting a stencil and a storage format (absolute or zero-centered). Next, the collision space is specified by fixing one type of statistical quantity and stating its basis as a sequence of polynomials. Additionally, corresponding relaxation rates must be specified. Each relaxation rate can be given either as a fixed numerical value or in symbolic form, allowing its value to remain undetermined until runtime of the generated code.

The definition's final three components require a significantly more elaborate structure. To model equilibrium distributions, the computation of macroscopic quantities, and force models, *lbmpy* provides specific hierarchies of Python classes. Abstract base classes define their respective interfaces. These components do not only encapsulate information about the method, but also take an active role during the code generation process.

For instance, the equilibrium object must not only produce an algebraic form (discrete or continuous) of its distribution; it must also distinguish between absolute

LB Method Definition	
Stencil	D2Q9   D3Q15   D3Q19   D3Q27   ...
Zero-Centered Storage	True   False
Collision Space	RAW_MOMENTS   CENTRAL_MOMENTS   CUMULANTS
Basis Polynomials	(1, $x$ , $y$ , $z$ , $x^2 - y^2$ , $x^2 - z^2$ , $x^2 + y^2 + z^2$ , ...)
Relaxation Rates	(0, 0, 0, 0, $\omega_s$ , $\omega_s$ , $\omega_b$ , ...)
Equilibrium Distribution	GenericDiscrete   ContinuousHydrodynamic   ...
Conserved Quantities	DensityVelocityComputation   ...
Force Model	Guo   He   ...

FIGURE 3.1. Configuration space for modelling MRT-type methods in *lbmpy*. Method parameters are split into three groups. The first group governs structure and storage format of the discrete lattice, the second group describes the method’s collision space, and the third group comprises those components governing the actual collision process.

form and delta-equilibrium and provide the background distribution in the latter case. Furthermore, it must provide methods to compute its raw moments, central moments, and cumulants. These methods are invoked during the symbolic derivation of the collision rule. The same holds for force models and conserved quantity computation; both components will provide parts of the equations making up the collision rule later on.

Apart from functional requirements, there is another significant advantage to using classes. While certain components are already implemented in the current version of *lbmpy*, such as the Maxwellian equilibrium, the Guo [22] and He [25] force models, etc., this structure makes *lbmpy* flexible and extensible. In particular, a developer using *lbmpy* may use the interfaces of the respective base classes in order to implement arbitrary custom equilibrium distributions, force models, or computation procedures for the macroscopic quantities. An example of this will be shown in subsection 5.2.

The user may freely combine various options for different method parameters, with some restrictions. For instance, deviation-only equilibrium instances may only be used in junction with zero-centered storage. Furthermore, as already discussed in previous sections, cumulant space is incompatible with deviation-only equilibria. As we will demonstrate in subsection 5.2, this interface is versatile and flexible both for implementing common hydrodynamic LBMs, and also for rapidly constructing more specialized methods that are not natively realized within *lbmpy*.

**4. Derivation Procedure and Code Generation.** The front-end API described in the previous section can be used to specify a concrete instance of an LB method. This specification serves as input to our automatic derivation and code generation pipeline. This pipeline comprises several steps, ultimately emitting an optimized numerical kernel implementing the collision rule for either CPU or GPU architectures. It relies on the domain-specific language of our code generation framework *pystencils* [4], which in turn is based on *SymPy*. In the following, we shall describe the stages of this pipeline as implemented in version 1.1 of *lbmpy*<sup>1</sup>.

**4.1. The Collision Rule.** The first stage of the pipeline takes the abstract definition to derive the equations constituting the collision rule. Depending on the

<sup>1</sup>[110git.cs.fau.de/pycodegen/lbmpy/-/tree/release/1.1](https://git.cs.fau.de/pycodegen/lbmpy/-/tree/release/1.1)

combination of storage and equilibrium format, an implementation of either (2.4)–(2.6) is derived in symbolic form. Manipulating the collision equations on the mathematical level, we can leverage all information available about the method to simplify and optimize the equations. The derivation system is modular, combining equations generated by several components. Among these are the equilibrium and force model instances, providing algebraic expressions of their respective representations in the given collision space; as well as the conserved quantity computation, which produces equations for density (or its analogues) and velocity (cf. (2.2) and (2.3)).

The equations forming the collision space transformation  $\mathcal{T}$  are provided by a set of dedicated classes within *lbmpy*. Apart from the simple symbolic matrix-vector-multiplication, *lbmpy* provides two types of highly efficient Chimera-based transforms for raw and central moment space, which we will discuss in the following subsections.

The equations produced by these various components are combined to produce the collision rule as a sequence of assignments of mathematical expressions to symbolic variables. This assignment collection is passed on to *lbmpy*'s sophisticated simplification procedure, described in subsection 4.3, and will ultimately be transformed into a numerical kernel (cf. subsections 4.4 and 4.5).

**4.1.1. Raw Moment Transform.** To compute raw moments from pre-collision populations, we first implement a Chimera transform to obtain monomial raw moments, which are afterward recombined to polynomials. Chimera transforms were first introduced in [19] for transforming to central moment space. Using intermediate ‘chimera’ quantities  $m_{x|\beta\gamma}$  and  $m_{xy|\gamma}$ , our raw-moment Chimera transform reads

$$(4.1) \quad \begin{aligned} f_{xyz} &:= \begin{cases} f_i & \text{if } \xi_i = (x, y, z)^T \text{ is contained in stencil} \\ 0 & \text{otherwise} \end{cases} \\ m_{xy|\gamma} &:= \sum_{z \in \{-1, 0, 1\}} f_{xyz} \cdot z^\gamma \\ m_{x|\beta\gamma} &:= \sum_{y \in \{-1, 0, 1\}} m_{xy|\gamma} \cdot y^\beta \\ m_{\alpha\beta\gamma} &:= \sum_{x \in \{-1, 0, 1\}} m_{x|\beta\gamma} \cdot x^\alpha. \end{aligned}$$

Due to its recursive nature, the Chimera transform minimizes the number of arithmetic operations, since each possible combination of populations and velocities is evaluated exactly once.

To transform from raw moments back to populations, we first decompose the polynomial moments into their monomial components. We then derive  $\mathbf{f}^* = \mathbf{M}^{-1} \mathbf{m}^*$  by symbolic matrix-vector-multiplication. Those equations we simplify by splitting the expressions for populations  $f_i$  and  $f_i^*$  moving in opposite directions into their symmetric and antisymmetric parts, such that

$$(4.2) \quad f_i^* = f_i^+ + f_i^-, \quad f_i^* = f_i^+ - f_i^-.$$

This split roughly cuts the number of arithmetic operations in half.

**4.2. Central Moment Transform.** In *lbmpy*, central moments are not transformed from and to populations directly, but using monomial raw moments as intermediates. To find an efficient transform between monomial raw and central moments,

we observe that they are bidirectionally related through binomial expansions

$$(4.3a) \quad \kappa_{\alpha\beta\gamma} = \sum_{a,b,c=0}^{\alpha,\beta,\gamma} \binom{\alpha}{a} \binom{\beta}{b} \binom{\gamma}{c} (-u_x)^{\alpha-a} (-u_y)^{\beta-b} (-u_z)^{\gamma-c} m_{abc},$$

$$(4.3b) \quad m_{\alpha\beta\gamma} = \sum_{a,b,c=0}^{\alpha,\beta,\gamma} \binom{\alpha}{a} \binom{\beta}{b} \binom{\gamma}{c} u_x^{\alpha-a} u_y^{\beta-b} u_z^{\gamma-c} \kappa_{abc}.$$

In both directions we may separate the nested sums, introducing them as Chimera quantities. This leads us to the binomial Chimera transforms between raw and central moments, which we state

$$(4.4) \quad \begin{aligned} \kappa_{ab|\gamma} &:= \sum_{c=0}^{\gamma} \binom{\gamma}{c} (-u_z)^{\gamma-c} m_{abc} & m_{ab|\gamma}^* &:= \sum_{c=0}^{\gamma} \binom{\gamma}{c} u_z^{\gamma-c} \kappa_{abc}^* \\ \kappa_{a|\beta\gamma} &:= \sum_{b=0}^{\beta} \binom{\beta}{b} (-u_y)^{\beta-b} \kappa_{ab|\gamma} & m_{a|\beta\gamma}^* &:= \sum_{b=0}^{\beta} \binom{\beta}{b} u_y^{\beta-b} m_{ab|\gamma}^* \\ \kappa_{\alpha\beta\gamma} &:= \sum_{a=0}^{\alpha} \binom{\alpha}{a} (-u_x)^{\alpha-a} \kappa_{a|\beta\gamma} & m_{\alpha\beta\gamma}^* &:= \sum_{a=0}^{\alpha} \binom{\alpha}{a} u_x^{\alpha-a} m_{a|\beta\gamma}^* \end{aligned}$$

Again, as every combination of moments and velocities is evaluated exactly once, this results in expressions that require only a minimal number of arithmetic operations. Polynomial central moments are combined from monomial quantities after the forward transform and decomposed before the backward transform.

**4.2.1. Cumulant Transform.** Similar to their original formulation in [19], cumulants in *lbmpy* are obtained from central moments. To derive their nonlinear relation, we re-express the cumulant-generating function (2.12) in terms of the central moment-generating function, to obtain the bidirectional relations

$$(4.5) \quad C(\boldsymbol{\Xi}) = (\boldsymbol{\Xi} \cdot \mathbf{u}) + \log K(\boldsymbol{\Xi}) \Leftrightarrow K(\boldsymbol{\Xi}) = \exp(C(\boldsymbol{\Xi}) - \boldsymbol{\Xi} \cdot \mathbf{u}).$$

We then employ *SymPy*'s symbolic differentiation capabilities to obtain expressions for the derivatives of  $C$  in terms of derivatives of  $K$ , and vice versa. Substituting monomial central moments and cumulants for these derivatives, we arrive at the equations for both transform directions. It must be noted that the equations obtained this way contain logarithms and exponential functions, whose presence is undesirable in a numerical kernel. However, they are only associated with the zeroth-order cumulant  $C_{000}$ , allowing us to eliminate them in a global simplification step.

**4.3. A Posteriori Simplifications.** Once the derivation of the collision kernel is complete, there will still be several possibilities to simplify and optimize the resulting kernel by symbolic manipulations. Our simplification procedure in *lbmpy* combines generic algebraic simplifications with optimizations designed specifically for LBMs, utilizing information about the method available during code generation. In particular, *lbmpy* applies the following simplification steps to the generated sequence of assignments. For advanced users of *lbmpy* it is additionally possible to disable these transformations or to customize them.

**Conserved Quantity Rewriting** Equations computing  $\rho$ ,  $\mathbf{u}$ , etc. from populations directly are replaced by equations involving zeroth- and first-order raw moments.

**Collapsing Conserved Central Moments** We collapse the equations for zeroth- and first-order central moments, e.g. obtaining,  $\kappa_{000} = \rho$  and  $\kappa_{100} = -F_x/2$ .

**Propagation of Logarithms** To simplify logarithmic and exponential expressions in cumulant-based collision rules, we propagate certain assignments containing logarithms to their usages; thus canceling them with the corresponding exponential functions.

**Common Subexpression Elimination** We use *SymPy*'s common subexpression elimination (CSE) feature to extract common terms into separate assignments, as previously described in [5].

**Expression Propagation** Assignments whose right-hand sides are constant, single symbols, products of macroscopic quantities, or multiples of body force components, are propagated to their usages.

**Unused Subexpression Elimination** A simple dependency analysis based on the static single assignment (SSA) form allows us to eliminate any assignments whose left-hand side symbols are never used.

While all of these simplifications can help in reducing kernel complexity, we found the combination of the final two to have the largest impact. Their strength hinges on the fact that *SymPy*, as a computer algebra system, automatically evaluates any constant terms, and attempts to cancel equal but opposite terms in any constructed expression. Our propagation steps trigger that behaviour automatically. Apart from minor eliminations occurring throughout the equations, this has two major effects.

First, quantities that are invariant under relaxation are propagated through the relaxation step, and inserted directly into the backward transform equations. This, in turn, leads to an automatic simplification of these equations. Apart from reducing overall operation count, this step also serves to eliminate algebraically idempotent operations. To give an example: Without this propagation, relaxation and backward binomial Chimera transform would include assignments similar to

$$(4.6) \quad \begin{aligned} \kappa_{000} &= \rho \\ \kappa_{100} &= -F_x/2 \\ \kappa_{000}^* &= \kappa_{000} \\ \kappa_{100}^* &= \kappa_{100} + F_x \\ m_{10|0}^* &= \kappa_{000}^* u_x + \kappa_{100}^*. \end{aligned}$$

The propagation steps described above will cause most of these assignments to be dropped, leaving us with just

$$(4.7) \quad m_{10|0}^* = \rho u_x + F_x/2.$$

The second and more significant effect of alias and constant propagation occurs whenever some relaxation rates are set to unity. In this case, propagation enables the elimination of large portions of the forward collision space transform, as well as significant algebraic simplification of the backward transform. Substituting  $\omega = 1$ , a relaxation equation  $q_p^* = q_p + \omega (q_p^{\text{eq}} - q_p)$  is immediately simplified to  $q_p^* = q_p^{\text{eq}}$ . The forward transform equations for  $q_p$  are then no longer required, allowing us to drop them. This has the potential to massively decrease the overall operation count, since the transform equations especially for higher-order quantities constitute a significant portion of the collision kernel. This will be discussed in further detail below.

If  $q_p^{\text{eq}}$  is a simple expression, it can be propagated into the backward transform equations. The effectiveness of this process is improved by the fact that many statistical quantities of common equilibrium distributions are zero. Propagating a zero

TABLE 4.1

*Arithmetic operation counts in the symbolic representation of compute kernels generated by *lbmpy* for several method definitions on the D2Q9, D3Q19 and D3Q27 stencils. Numbers for kernels derived without simplification (N), standard simplification (S) and simplification plus CSE (S+CSE) are listed.*

Method	D2Q9			D3Q19			D3Q27		
	N	S	S+CSE	N	S	S+CSE	N	S	S+CSE
SRT	432	113	<b>91</b>	1156	312	<b>204</b>	3842	428	<b>285</b>
TRT	437	191	<b>111</b>	1161	480	<b>239</b>	3847	668	<b>343</b>
O-MRT	196	142	<b>122</b>	554	397	<b>324</b>	928	596	<b>488</b>
R-O-MRT	196	105	<b>89</b>	554	290	<b>232</b>	928	407	<b>329</b>
WO-MRT	178	117	<b>102</b>	507	339	<b>283</b>	843	484	<b>415</b>
R-WO-MRT	178	87	<b>75</b>	507	244	<b>196</b>	843	316	<b>265</b>
CM	236	156	<b>132</b>	638	415	<b>344</b>	1140	747	<b>600</b>
R-CM	236	108	<b>95</b>	638	255	<b>216</b>	1140	417	<b>342</b>
K	676	167	<b>142</b>	1854	454	<b>376</b>	7623	1035	<b>820</b>
R-K	676	114	<b>100</b>	1854	272	<b>231</b>	7623	489	<b>397</b>

into the backward transform equations will typically eliminate only a small number of summands; in fact, for the backward Chimera transform, only a single term is removed from one innermost Chimera. Still, the more quantities are relaxed with  $\omega_i = 1$ , the more significant this effect will grow. Propagation of zeroes is most effective with the cumulant backward transform, due to its nonlinearity. The equations computing higher-order post-collision central moments from post-collision cumulants are by far the most complex derived anywhere within our framework. They do, however, contain a significant number of products of post-collision cumulants. Substituting those cumulants' equilibrium values, which are mostly equal to zero, several such multiplications may be eliminated.

To illustrate the efficacy of our simplification strategy, we count the total number of operations in the collision kernels generated by *lbmpy* from a number of different method definitions. All methods employ zero-centered storage and the default continuous Maxwellian equilibrium, whose moments are truncated at the second order in velocity. No force model is employed. Kernels are derived for non-weighted orthogonal MRT (denoted O-MRT) and weighted orthogonal MRT (WO-MRT) in raw moment space [28]; as well as polynomial central moment (CM) [15, 42, 44] and polynomial cumulant (K) [19] methods. We compare these with implementations of the SRT and TRT collision operators, also derived by *lbmpy* using the original procedure published in [5]. The prefix ‘R-’ denotes a regularized method; all but the relaxation rates governing shear viscosity are therein set to unity. Such regularization is permissible since higher-order ‘ghost’ modes do not affect simulated physics [19, 17]; further, it is a common measure to improve a method’s stability [30, 16, 8]. Otherwise, every relaxation rate is represented by a dedicated symbol  $\omega_i$  ( $i = 0, \dots, q - 1$ ). The population and raw moment space methods use the delta-equilibrium for relaxation, while the absolute equilibrium is used in central moment and cumulant space.

Table 4.1 lists the arithmetic operation counts of kernels generated for those methods, with three different levels of simplification intensity: None at all; full simplification without CSE; and full simplification including CSE. The table shows that *lbmpy* is capable of producing kernel implementations for complex and advanced lat-

TABLE 4.2

*Savings in arithmetic operations due to regularization for several methods on the D3Q27 stencil. We display the savings relative to the operation count of the kernel variant with purely symbolic relaxation rates. ‘Higher Order Regularized’ implies that only the relaxation rates for statistical quantites of order 5 and 6 are set to unity. ‘Fully Regularized’ methods have all but their shear relaxation rates set to one. All kernels were simplified as far as possible, without applying CSE.*

Method	Operations Savings	
	Higher Order Regularized	Fully Regularized
O-MRT	8%	30%
WO-MRT	9%	33%
CM	11%	44%
K	17%	53%

tice Boltzmann methods with only little overhead when compared to simple SRT or TRT kernels. In fact, the regularized WO-MRT kernels are even visibly smaller in size than the basic SRT kernels. This observation holds also in comparison with carefully hand-crafted implementations, as in [52], where Wellein et al. report about 200 operations for the D3Q19 SRT kernel. Note that, for non-regularized raw moment MRT, we observe vast improvements (almost 50 % fewer operations) compared to the results originally published for *lbmpy* in [5]. Furthermore, considering the cumulant-based LBM, we surpass the carefully optimized implementation of Geier et al., who in [16] report 432 arithmetic operations just for the forward transform. Note, however, that these numbers describe implementations in a high-level programming language or, in our case, in symbolic form. The actual number of floating-point operations in the compiled code may differ due to transformations applied by modern optimizing compilers.

Comparing the kernel sizes of methods using purely symbolic relaxation rates with their regularized variants, the effectiveness of our expression propagation steps becomes apparent. Table 4.2 lists the savings in operation counts with respect to the non-regularized version for orthogonal raw-moment, central moment and cumulant methods on the D3Q27 stencil. We compare both the fully regularized methods, whose operation counts are shown in Table 4.1, and only partially (higher-order) regularized methods, with only the relaxation rates for quantities of order  $\geq 5$  set to unity. The reduction in operation count achieved through regularization grows with the complexity of the collision equations. The largest part of these savings comes from the elimination of transform equations. In the central moment method’s kernel, the forward transform equations for the fifth and sixth order moments alone make for roughly eight percent of the total operation count, which can simply be omitted with regularization. The largest potential for simplification exists for cumulant-based methods: on D3Q27, ten percent of arithmetic operations can be attributed to just the four equations computing the monomial pre-collision cumulants  $C_{122}$ ,  $C_{212}$ ,  $C_{221}$  and  $C_{222}$  from central moments.

**4.4. Streaming Patterns and Update Rule.** The collision rule generated by stage one constitutes the relaxation process for a single cell, whose populations are represented purely symbolically. In the next stage, we employ the field abstraction of *pystencils* to map the rule over the cells of a  $d$ -dimensional field data structure, holding  $q$  values per cell. Therein, the iteration over the field is abstracted through *field accesses*; special kinds of symbols that model accesses relative to the current

cell. A field access reading  $\mathbf{f}[\mathbf{x}, \mathbf{y}, \mathbf{z}](\mathbf{i})$  corresponds to the  $i$ -th entry of the local cell's neighbor with integer offset  $(x, y, z)$ . The correspondence between pre- and post-collision populations, and relative field accesses, is governed by the *streaming pattern*. At the time of writing, *lbmpy* supports the classical pull- and push-patterns, which require separate arrays for reading and writing to avoid data conflicts [53]; as well as four different in-place streaming patterns. With these techniques the double data structures can be avoided. They are the AA-pattern [2], Esoteric Twist [18], Esoteric Pull, and Esoteric Push [32]. The field update rule is constructed from the collision rule by substituting field accesses for symbolic populations, according to the specified streaming pattern.

**4.5. Code Generation.** As a last step, the update rule is passed on to the *pystencils* code generator [4]. It transforms its symbolic equations into compilable C or CUDA code; wrapping it either with nested loops over the field, or prepending GPU indexing code. Furthermore, the code generation system may optionally apply acceleration techniques such as OpenMP parallelization [38], loop splitting, or loop blocking [4]. Finally, the generated kernel can be compiled and loaded directly within the Python environment, or it can be written to a file to be integrated into separate code bases. The former option allows for rapid development and testing of LB methods, while the latter enables integration with large-scale simulation frameworks, like *walBerla* [5, 3, 26].

**5. Applications.** In this section, we present a numerical benchmark involving Taylor-Green vortex decay to assess the impact of zero-centered storage. As a second example we employ *lbmpy* to generate an LBM for the shallow water equations to illustrate its extensibility to novel application fields.

**5.1. Taylor Green Vortex.** In order to assess the effectiveness of our generalization of zero-centered storage to moment spaces in improving numerical accuracy and reducing round-off error, we replicate a test case recently published in [33] involving the Taylor-Green vortex flow. Therein, a periodic box of vortices with velocity magnitude  $u_0$  is initialized, their transient decay simulated, and compared to the known analytic solution. The analytic solution, which also specifies the initial flow field, reads

$$(5.1) \quad \begin{aligned} u_x(t) &= u_0 \cos(\kappa x) \sin(\kappa y) \exp(-2\nu\kappa^2 t) \\ u_y(t) &= -u_0 \sin(\kappa x) \cos(\kappa y) \exp(-2\nu\kappa^2 t) \\ u_z(t) &= 0 \\ \rho(t) &= 1 - \frac{3u_0^2}{4} (\cos(2\kappa x) + \cos(2\kappa y)) \exp(-4\nu\kappa^2 t) \end{aligned}$$

The system is initialized at  $t = 0$  with  $u_0 = 0.25$ . We set the kinematic shear viscosity to  $\nu = \frac{1}{6}$ , which results in the viscosity-governing relaxation rate  $\omega = 1$ . Furthermore,  $\kappa = \frac{2\pi}{L}$  and  $L = 256$  is the side length of the squared domain. The initial state of the flow field is displayed in Figure 5.1.

The kinetic energy is calculated as

$$(5.2) \quad E(t) = \int_0^L \int_0^L \int_0^L \frac{\rho}{2} (u_x^2 + u_y^2 + u_z^2) dx dy dz = u_0^2 \pi^2 \exp(-4\nu\kappa^2 t),$$

and we denote the initial kinetic energy as  $E_0 = E(0)$ .

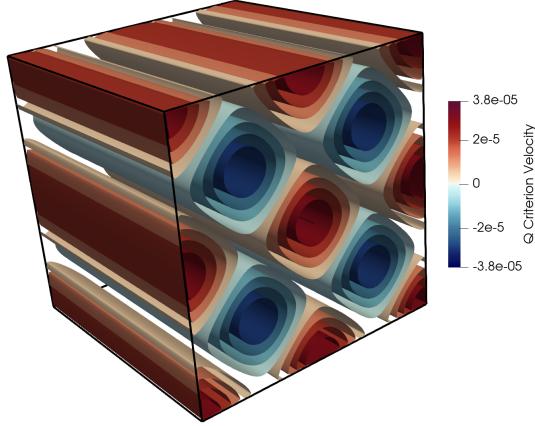


FIGURE 5.1. Initialization of the Taylor-Green vortex test case with  $u_0 = 0.25$ ,  $\nu = \frac{1}{6}$ ,  $\omega = 1$ ,  $\kappa = \frac{2\pi}{L}$  and  $L = 256$ .

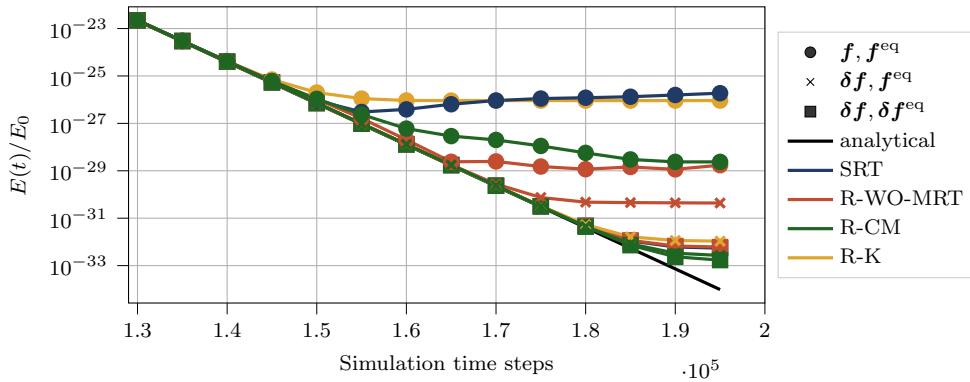


FIGURE 5.2. Relativ energy  $E(t)/E_0$  for various D3Q27 LB methods. The PDFs are either stored in the absolute (circular marker) or zero-centered format, the latter relaxed against the absolute ( $x$  marker) or delta-equilibrium (square marker). We show only a range of timesteps where deviation from the analytical solution occurs.

We simulated the vortex decay using the SRT, R-WO-MRT, R-CM and R-K methods (cf. subsection 4.3), with all admissible combinations of storage and equilibrium format. In Figure 5.2 the analytical solution of the kinetic energy is compared to simulated results. The vortex velocity and thus the kinetic energy are expected to decay exponentially due to viscous friction. This is matched by the simulation until, at some point, the simulated energy stagnates on a plateau. The reason for this phenomenon is that lower velocities can no longer be represented due to the truncation error caused by the floating point number format. The truncation error for the IEEE-754 double precision format [1] is  $\epsilon = 2.2 \cdot 10^{-16}$ . Due to the squared calculation for the kinetic energy, the plateau's location is expected to occur at around  $\epsilon^2$ . Comparing different compute kernels derived by *lbmpy*, we observe that some are able to reach the predicted plateau, while others stagnate visibly earlier. The relative energy  $E(t)/E_0$  after 200 000 time steps is also shown in Table 5.1. We consistently

TABLE 5.1

*Relativ energy  $E(t)/E(E_0)$  for various D3Q27 LB methods after 200 000 timesteps for all admissible combinations of storage and equilibrium format.*

Collision space	absolute storage	zero-centered	
		$f^{\text{eq}}$	$\delta f^{\text{eq}}$
SRT	$1.9 \cdot 10^{-26}$	-	$5.4 \cdot 10^{-33}$
R-WO-MRT	$1.7 \cdot 10^{-29}$	$4.4 \cdot 10^{-31}$	$6.1 \cdot 10^{-33}$
R-CM	$2.4 \cdot 10^{-29}$	$2.7 \cdot 10^{-33}$	$1.7 \cdot 10^{-34}$
R-K	$9.1 \cdot 10^{-27}$	$1.1 \cdot 10^{-32}$	-

attain much lower levels of the plateau using zero-centered storage in combination with the delta-equilibrium. Relaxing zero-centered populations against the absolute equilibrium, the R-CM and R-K methods still yield very high precision, while R-WO-MRT stagnates earlier. For both the SRT-, and the cumulant method, the difference between absolute and zero-centered storage is the most drastic.

**5.2. Shallow Water LBMs.** The shallow water equations (SWE) are an approximation to the Navier-Stokes equations for flow regimes where the horizontal characteristic length scale is significantly larger than the vertical length scale, making it permissible to neglect vertical flow phenomena [51]. They are obtained by integrating the NSE in  $z$ -direction; fluid density is replaced by water column height  $h$  and velocity is averaged across the third dimension [54]. The SWE are therefore purely two-dimensional. A number of LBM solvers for the SWE have been proposed [54, 43, 51] and successfully applied to problems of hydraulic engineering [50]. Here, we consider the central moment-based shallow water LBM by de Rosis [43] and the cumulant-based method presented by Venturi et al. [51, 50]. We show how these methods may be implemented using the *lbmpy* modelling framework introduced in section 3 in just a few lines of Python code, automatically generating a highly optimized kernel implementation of the respective collision operators. We then present simulation results obtained with the generated kernels.

The method of de Rosis is based on the discrete shallow water equilibrium proposed by Zhou [54] for the D2Q9 stencil, which reads

$$(5.3) \quad \begin{aligned} f_0^{\text{eq}} &= h \left( 1 - \frac{5gh}{6} - \frac{2\mathbf{u} \cdot \mathbf{u}}{3} \right), \\ f_i^{\text{eq}} &= \lambda_i h \left( \frac{gh}{6} + \frac{\boldsymbol{\xi}_i \cdot \mathbf{u}}{3} + \frac{(\boldsymbol{\xi}_i \cdot \mathbf{u})^2}{2} - \frac{\mathbf{u} \cdot \mathbf{u}}{3} \right) \quad (i = 1, \dots, 8). \end{aligned}$$

Here,  $g$  denotes gravitational acceleration in lattice units, and  $\lambda_i = 1$  if  $\|\boldsymbol{\xi}_i\|_1 = 1$ , otherwise  $\lambda_i = 1/4$ . Listing 1 shows how a central moment-based method using this equilibrium may be set up within *lbmpy*. In lines 3 to 13, we create an equilibrium instance from the discrete equations, which enters the abstract representation of the shallow water method in lines 15 to 20. We specify the same polynomial basis as used in [43], fix central moments as the collision space, and define a regularized set of relaxation rates. Finally, line 22 invokes the code generation pipeline introduced in section 4, generating and compiling a C implementation of the collision kernel, which is made available to the user as a Python function.

The method of Venturi et al. [51] may be recreated even more compactly, using the continuous Maxwellian and setting its squared speed of sound parameter to  $c_s^2 =$

---

```

1 x, y, g, h, u_x, u_y, omega_s = sp.symbols('x y g h u_x u_y omega_s')
2
3 def f_eq(xi):
4     xi_sum = sum(abs(xi_i) for xi_i in xi)
5     if xi_sum == 0:
6         return h * (1 - (5 * g * h) / 6 - (2 * dot(u, u)) / 3)
7     else:
8         lambda_ = 1 if xi_sum == 1 else sp.Rational(1, 4)
9         common = (g * h) / 6 + dot(xi, u) / 3 + dot(xi, u)**2 / 2 - dot(u, u) / 6
10        return lambda_ * h * common
11
12 eq_populations = [f_eq(xi) for xi in d2q9]
13 swe_eq = GenericDiscreteEquilibrium(d2q9, eq_populations, h, u)
14
15 polys = (1, x, y, x * y, x**2 - y**2, x**2 + y**2, x * y**2, y * x**2, x**2 * y**2)
16 r_rates = [0, 0, 0, omega_s, omega_s, 1, 1, 1, 1]
17 rr_dict = dict(zip(polys, r_rates))
18 cqc = DensityVelocityComputation(d2q9, density_symbol=h, ...)
19 cspace = CollisionSpaceInfo(CollisionSpace.CENTRAL_MOMENTS)
20 swe_method = create_from_equilibrium(d2q9, swe_eq, cqc, rr_dict, cspace)
21
22 lb_kernel = create_lb_function(lb_method=swe_method)

```

---

LISTING 1

*Instantiation of the discrete shallow water equilibrium using `lbmpy`'s modelling framework; construction of the central moment-based shallow water LBM, plus automatic derivation, generation and compilation of the collision kernel.*

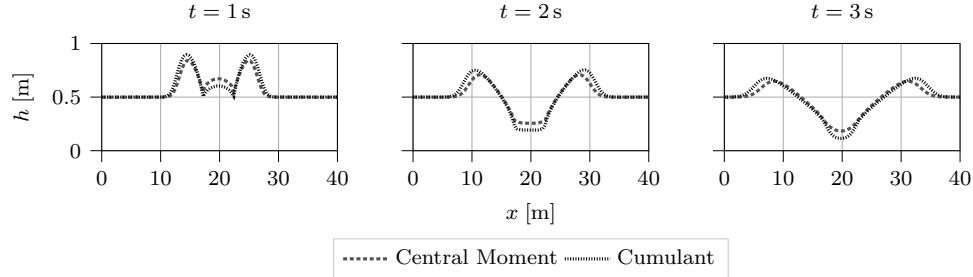


FIGURE 5.3. Comparison of water depth in circular dam break simulations using the central-moment based (CM) and cumulant-based (K) shallow water LBMs.

$\frac{1}{2}gh$ . The second half of the code in listing 1 is then identical, except for specifying CUMULANTS as a collision space, instead of central moments.

We put the kernels thus generated to work in simulating a circular dam break scenario, using the same setup as in [43]. We place a water column of radius 2.5 m and height 2.5 m at the center of a cubic domain with side length 40 m, which is otherwise filled with 0.5 m of water on top of an even and frictionless bed. The domain is discretized with  $100^2$  lattice cells and delimited by periodic boundary conditions; the simulated time step is  $\Delta t = 0.05$  s per step. The kinematic viscosity is set to unity; this yields a shear viscosity-governing relaxation rate of  $\omega_s \approx 0.696$ . The entire simulation is set up very rapidly in Python code using the additional facilities of `pystencils` and `lbmpy`. The full Python code for this setup is available as part of `lbmpy`'s online documentation<sup>2</sup>.

Figure 5.3 shows water column height in a cross-section of the domain at  $y = 20$  m, at 1, 2 and 3 seconds of simulated time. While both methods agree qualitatively, the

---

<sup>2</sup>[pycodegen.pages.i10git.cs.fau.de/lbmpy](http://pycodegen.pages.i10git.cs.fau.de/lbmpy)

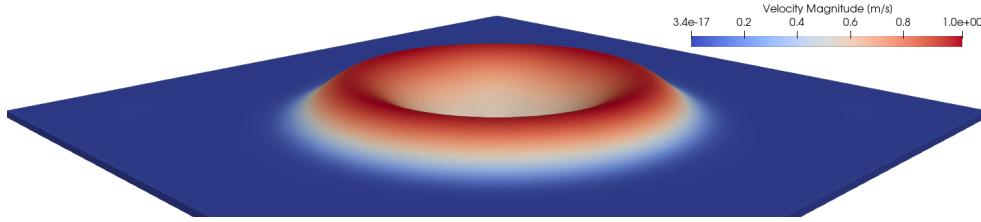


FIGURE 5.4. Circular dam break simulation at  $t = 2$  s using the central moment-based method.

cumulant-based method shows a visibly deeper trough and a more spread-out wave front. The wave front after two seconds, as predicted by the central moment-based method, is visualized in Figure 5.4.

**6. Conclusion and Outlook.** This article presents *lbmpy* as a sophisticated software architecture that supports the algebraic modelling of advanced MRT lattice Boltzmann methods. Its key feature is the ability to generate efficient computer code automatically from abstract specifications of the methods. The methodology is based on an extensive theoretical framework for the concise description of MRT collision rules. The framework formalizes the zero-centered storage format as a decomposition of the populations into background and deviation components. This decomposition we generalize to raw and central moment space, applying it also to the equilibrium distribution. Thus we arrive at three general collision equations, relaxing absolute populations against the absolute equilibrium and zero-centered populations against both the absolute and delta-equilibrium.

We present the front-end of *lbmpy* as a Python-based software incarnation of this theoretical calculus. Utilizing a computer algebra system, the software supports manipulating LBMs on the purely symbolic level. We further describe how object-oriented programming can be used to represent complex components, like the equilibrium distribution and force models. This approach makes *lbmpy* flexible and extensible.

We discuss in detail the way optimized collision rules are derived within *lbmpy* through symbolic manipulation. Our derivation system generates efficient implementations for collision rules in raw moment, central moment, and cumulant space, including all permissible combinations of storage and equilibrium formats. We provide specialized derivations for the collision space transforms since they constitute the most expensive computations. The linear mappings between populations, raw, and central moments are realized using two novel Chimera transforms: One recursively decomposing the calculation of discrete raw moments, and one a split-up version of a threefold binomial expansion. Both are designed to leverage the respective transforms' recursive nature in order to minimize the amount of arithmetic operations. We furthermore introduce a method of deriving the transforms between central moments and cumulants by symbolic differentiation of the respective generating functions.

Combining domain-specific knowledge with information contained in the symbolic equations, we succeed in significantly reducing arithmetic operation counts of complex LBM kernels. We observe that especially the common step of regularization permits aggressive optimizations leading to drastically reduced computational cost. This serves to produce remarkably low operation counts; in fact, we were able to produce cumulant-based kernels with only between thirteen (on D3Q19) and forty percent (on D3Q27) more operations than the respective SRT kernel.

We conduct a test case involving decaying Taylor-Green vortex flow, showing the effectiveness of the generalized zero-centered storage format in improving numerical accuracy. Finally, we illustrate *lbmpy*'s versatility and fitness for rapid prototyping by setting up an LB solver for the shallow water equations in just a few lines of Python code.

The reduction of arithmetic complexity in implementations of advanced LBMs is just one first step toward efficient large-scale fluid dynamics simulations. In the past, *lbmpy*-generated kernels have already been shown to be performant, both individually on single CPUs, as well as powering massively parallel simulations on clusters of CPUs and GPUs [5, 26]. With the present revision of the framework, we provide a central ingredient for more time- and energy-efficient implementations of sophisticated lattice Boltzmann solvers employing the promising central moment and cumulant methods. Therein, we aim to minimize time and energy consumed by both software developers and computing hardware. Therefore, the evaluation of the performance characteristics of the generated kernels on diverse modern hardware architectures as well as the assertion of their fitness for latest peta- and exascale supercomputers shall be subjects of future work.

**Acknowledgements.** The authors acknowledge funding by the SCALABLE project. SCALABLE has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956000. The authors are grateful to the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for funding projects 408062554 and 433735254. The authors also gratefully acknowledge financial support by the Bavarian State Ministry of Science and the Arts through the Competence Network for Scientific High Performance Computing in Bavaria (KONWIHR).

## REFERENCES

- [1] *IEEE standard for floating-point arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008), (2019), pp. 1–84, <https://doi.org/10.1109/IEEESTD.2019.8766229>.
- [2] P. BAILEY, J. MYRE, S. WALSH, D. LILJA, AND M. SAAR, *Accelerating lattice Boltzmann fluid flow simulations using graphics processors*, in 2009 International Conference on Parallel Processing, IEEE, sep 2009, <https://doi.org/10.1109/icpp.2009.38>.
- [3] M. BAUER, S. EIBL, C. GODENSCHWAGER, N. KOHL, M. KURON, C. RETTINGER, F. SCHORNBAUM, C. SCHWARZMEIER, D. THÖNNES, H. KÖSTLER, AND U. RÜDE, *waLBerla: A block-structured high-performance framework for multiphysics simulations*, Comput. Math. Appl., 81 (2021), pp. 478–501, <https://doi.org/10.1016/j.camwa.2020.01.007>.
- [4] M. BAUER, J. HÖTZER, D. ERNST, J. HAMMER, M. SEIZ, H. HIERL, J. HÖNIG, H. KÖSTLER, G. WELLEIN, B. NESTLER, AND U. RÜDE, *Code generation for massively parallel phase-field simulations*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, nov 2019, <https://doi.org/10.1145/3295500.3356186>.
- [5] M. BAUER, H. KÖSTLER, AND U. RÜDE, *lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods*, Journal of Computational Science, 49 (2021), 101269, <https://doi.org/10.1016/j.jocs.2020.101269>.
- [6] P. L. BHATNAGAR, E. P. GROSS, AND M. KROOK, *A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems*, Physical Review, 94 (1954), pp. 511–525, <https://doi.org/10.1103/physrev.94.511>.
- [7] H. CHEN, S. CHEN, AND W. H. MATTHAEUS, *Recovery of the navier-stokes equations using a lattice-gas Boltzmann method*, Phys. Rev. A, 45 (1992), pp. R5339–R5342, <https://doi.org/10.1103/PhysRevA.45.R5339>, <https://link.aps.org/doi/10.1103/PhysRevA.45.R5339>.
- [8] C. COREIXAS, B. CHOPARD, AND J. LATT, *Comprehensive comparison of collision models in the lattice Boltzmann framework: Theoretical investigations*, Phys. Rev. E, 100 (2019), 033305, <https://doi.org/10.1103/physreve.100.033305>.

- [9] D. d'HUMIÈRES, *Multiple-relaxation-time lattice Boltzmann models in three dimensions*, Philos. Trans. Roy. Soc. A, 360 (2002), pp. 437–451, <https://doi.org/10.1098/rsta.2001.0955>.
- [10] B. DÜNWEG, U. D. SCHILLER, AND A. J. C. LADD, *Statistical mechanics of the fluctuating lattice Boltzmann equation*, Phys. Rev. E, 76 (2007), 036704, <https://doi.org/10.1103/physreve.76.036704>.
- [11] L. FEI AND K. H. LUO, *Consistent forcing scheme in the cascaded lattice Boltzmann method*, Phys. Rev. E, 96 (2017), p. 053307, <https://doi.org/10.1103/physreve.96.053307>.
- [12] L. FEI, K. H. LUO, AND Q. LI, *Three-dimensional cascaded lattice Boltzmann method: Improved implementation and consistent forcing scheme*, Physical Review E, 97 (2018), p. 053309, <https://doi.org/10.1103/physreve.97.053309>.
- [13] M. GEIER, *Ab initio derivation of the cascaded lattice Boltzmann automaton*, PhD thesis, University of Freiburg, 2006.
- [14] M. GEIER, A. FAKHARI, AND T. LEE, *Conservative phase-field lattice Boltzmann model for interface tracking equation*, Phys. Rev. E, 91 (2015), 063309, <https://doi.org/10.1103/physreve.91.063309>.
- [15] M. GEIER, A. GREINER, AND J. G. KORVINK, *Cascaded digital lattice Boltzmann automata for high reynolds number flow*, Phys. Rev. E, 73 (2006), 066705, <https://doi.org/10.1103/physreve.73.066705>.
- [16] M. GEIER, S. LENZ, M. SCHÖNHERR, AND M. KRAFCZYK, *Under-resolved and large eddy simulations of a decaying taylor-green vortex with the cumulant lattice Boltzmann method*, Theor. Comput. Fluid Dyn., 35 (2020), pp. 169–208, <https://doi.org/10.1007/s00162-020-00555-7>.
- [17] M. GEIER, A. PASQUALI, AND M. SCHÖNHERR, *Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part i: Derivation and validation*, J. Comput. Phys., 348 (2017), pp. 862–888, <https://doi.org/10.1016/j.jcp.2017.05.040>.
- [18] M. GEIER AND M. SCHÖNHERR, *Esoteric twist: An efficient in-place streaming algorithmus for the lattice Boltzmann method on massively parallel hardware*, Computation, 5 (2017), p. 19, <https://doi.org/10.3390/computation5020019>.
- [19] M. GEIER, M. SCHÖNHERR, A. PASQUALI, AND M. KRAFCZYK, *The cumulant lattice Boltzmann equation in three dimensions: Theory and validation*, Comput. Math. Appl., 70 (2015), pp. 507–547, <https://doi.org/10.1016/j.camwa.2015.05.001>.
- [20] I. GINZBURG, F. VERHAEGHE, AND D. D'HUMIÈRES, *Two-relaxation-time lattice Boltzmann scheme: about parametrization, velocity, pressure and mixed boundary conditions*, Commun. Comput. Phys., 3 (2008), pp. 427–478.
- [21] G. GRUSZCZYŃSKI, T. MITCHELL, C. LEONARDI, Ł. ŁANIEWSKI-WOŁĘK, AND T. BARBER, *A cascaded phase-field lattice Boltzmann model for the simulation of incompressible, immiscible fluids with high density contrast*, Comput. Math. Appl., 79 (2020), pp. 1049–1071, <https://doi.org/10.1016/j.camwa.2019.08.018>.
- [22] Z. GUO, C. ZHENG, AND B. SHI, *Discrete lattice effects on the forcing term in the lattice Boltzmann method*, Phys. Rev. E, 65 (2002), 046308, <https://doi.org/10.1103/physreve.65.046308>.
- [23] T. GYSI, T. GROSSER, AND T. HOEFLER, *MODESTO: Data-centric analytic optimization of complex stencil programs on heterogeneous architectures*, in Proceedings of the 29th ACM on International Conference on Supercomputing, ICS '15, New York, NY, USA, 2015, Association for Computing Machinery, p. 177–186, <https://doi.org/10.1145/2751205.2751223>.
- [24] X. HE AND L.-S. LUO, *Lattice Boltzmann model for the incompressible navier–stokes equation*, J. Stat. Phys., 88 (1997), pp. 927–944, <https://doi.org/10.1023/b:joss.0000015179.12689.e4>.
- [25] X. HE, X. SHAN, AND G. D. DOOLEN, *Discrete Boltzmann equation model for nonideal gases*, Phys. Rev. E, 57 (1998), pp. R13–R16, <https://doi.org/10.1103/physreve.57.r13>.
- [26] M. HOLZER, M. BAUER, H. KÖSTLER, AND U. RÜDE, *Highly efficient lattice Boltzmann multiphase simulations of immiscible fluids at high-density ratios on CPUs and GPUs through code generation*, The International Journal of High Performance Computing Applications, 35 (2021), pp. 413–427, <https://doi.org/10.1177/10943420211016525>.
- [27] M. J. KRAUSE, A. KUMMERLÄNDER, S. J. AVIS, H. KUSUMAATMAJA, D. DAPELO, F. KLEMENS, M. GAEDTKE, N. HAFEN, A. MINK, R. TRUNK, J. E. MARQUARDT, M.-L. MAIER, M. HAUSSMANN, AND S. SIMONIS, *OpenLB—open source lattice Boltzmann code*, Comput. Math. Appl., 81 (2021), pp. 258–288, <https://doi.org/10.1016/j.camwa.2020.04.033>.
- [28] T. KRÜGER, H. KUSUMAATMAJA, A. KUZMIN, O. SHARDT, G. SILVA, AND E. M. VIGGEN, *The Lattice Boltzmann Method*, Springer International Publishing, Switzerland, 2017, <https://doi.org/10.1007/978-3-319-44649-3>.

- [29] Ł. ŁANIEWSKI-WOŁŁK AND J. ROKICKI, *Adjoint Lattice Boltzmann for topology optimization on multi-GPU architecture*, Computers & Mathematics with Applications, 71 (2016), <https://doi.org/10.1016/j.camwa.2015.12.043>.
- [30] J. LATT AND B. CHOPARD, *Lattice Boltzmann method with regularized non-equilibrium distribution functions*, 2005, <https://doi.org/10.48550/ARXIV.PHYSICS/0506157>, <https://arxiv.org/abs/physics/0506157>.
- [31] J. LATT, O. MALASPINAS, D. KONTAXAKIS, A. PARMIGIANI, D. LAGRAVA, F. BROGI, M. B. BELGACEM, Y. THORIMBERT, S. LECLAIRE, S. LI, F. MARSON, J. LEMUS, C. KOT-SALOS, R. CONRADIN, C. COREIXAS, R. PETKANTCHIN, F. RAYNAUD, J. BENY, AND B. CHOPARD, *Palabos: Parallel lattice Boltzmann solver*, Comput. Math. Appl., 81 (2021), pp. 334–350, <https://doi.org/10.1016/j.camwa.2020.03.022>.
- [32] M. LEHMANN, *Esoteric pull and esoteric push: Two simple in-place streaming schemes for the lattice Boltzmann method on GPUs*, Computation, 10 (2022), p. 92, <https://doi.org/10.3390/computation10060092>.
- [33] M. LEHMANN, M. J. KRAUSE, G. AMATI, M. SEGA, J. HARTING, AND S. GEKLE, *Accuracy and performance of the lattice Boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats*, Phys. Rev. E, 106 (2022), <https://doi.org/10.1103/physreve.106.015308>.
- [34] C. LENGAUER, S. APEL, M. BOLTEN, S. CHIBA, U. RÜDE, J. TEICH, A. GRÖSSLINGER, F. HANNIG, H. KÖSTLER, L. CLAUS, A. GREBAHN, S. GROTH, S. KRONAWITTER, S. KUCKUK, H. RITTICH, C. SCHMITT, AND J. SCHMITT, *Exastencils: Advanced multigrid solver generation*, in Software for Exascale Computing - SPPEXA 2016-2019, H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel, eds., Cham, 2020, Springer International Publishing, pp. 405–452.
- [35] A. LOGG AND G. N. WELLS, *Dolfin: Automated finite element computing*, ACM Trans. Math. Softw., 37 (2010), <https://doi.org/10.1145/1731022.1731030>.
- [36] N. MARUYAMA, T. NOMURA, K. SATO, AND S. MATSUOKA, *Physis: An implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers*, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’11, New York, NY, USA, 2011, Association for Computing Machinery, <https://doi.org/10.1145/2063384.2063398>.
- [37] A. MEURER, C. P. SMITH, M. PAPROCKI, O. ČERTÍK, S. B. KIRPICHEV, M. ROCKLIN, A. KUMAR, S. IVANOV, J. K. MOORE, S. SINGH, T. RATHNAYAKE, S. VIG, B. E. GRANGER, R. P. MULLER, F. BONAZZI, H. GUPTA, S. VATS, F. JOHANSSON, F. PEDREGOSA, M. J. CURRY, A. R. TERREL, V. ROUČKA, A. SABOO, I. FERNANDO, S. KULAL, R. CIMRMAN, AND A. SCOPATZ, *Sympy: symbolic computing in python*, PeerJ Computer Science, 3 (2017), p. e103, <https://doi.org/10.7717/peerj-cs.103>, <https://doi.org/10.7717/peerj-cs.103>.
- [38] OPENMP ARCHITECTURE REVIEW BOARD, *OpenMP Application Programming Interface Version 5.2*, Nov. 2021.
- [39] K. N. PREMNATH AND S. BANERJEE, *On the three-dimensional central moment lattice Boltzmann method*, J. Stat. Phys., 143 (2011), pp. 747–794, <https://doi.org/10.1007/s10955-011-0208-9>.
- [40] F. RATHGEBER, D. A. HAM, L. MITCHELL, M. LANGE, F. LUPORINI, A. T. T. MCRAE, G.-T. BERCEA, G. R. MARKALL, AND P. H. J. KELLY, *Firedrake: Automating the finite element method by composing abstractions*, ACM Trans. Math. Softw., 43 (2016), <https://doi.org/10.1145/2998441>.
- [41] P. RAWAT, M. KONG, T. HENRETTY, J. HOLEWINSKI, K. STOCK, L.-N. POUCHET, J. RAMANUJAM, A. ROUNTEV, AND P. SADAYAPPAN, *SDSLc: A multi-target domain-specific compiler for stencil computations*, WOLFHPC ’15, New York, NY, USA, 2015, Association for Computing Machinery, <https://doi.org/10.1145/2830018.2830025>.
- [42] A. D. ROSIS, *Non-orthogonal central moments relaxing to a discrete equilibrium: A d2q9 lattice Boltzmann model*, Europhysics Letters, 116 (2016), 44003, <https://doi.org/10.1209/0295-5075/116/44003>.
- [43] A. D. ROSIS, *A central moments-based lattice Boltzmann scheme for shallow water equations*, Computer Methods in Applied Mechanics and Engineering, 319 (2017), pp. 379–392, <https://doi.org/10.1016/j.cma.2017.03.001>.
- [44] A. D. ROSIS, *Nonorthogonal central-moments-based lattice Boltzmann scheme in three dimensions*, Phys. Rev. E, 95 (2017), 013310, <https://doi.org/10.1103/physreve.95.013310>.
- [45] F. SCHORNBAUM AND U. RÜDE, *Massively parallel algorithms for the lattice Boltzmann method on NonUniform grids*, SIAM J. Sci. Comput., 38 (2016), pp. C96–C126, <https://doi.org/10.1137/15m1035240>.
- [46] F. SCHORNBAUM AND U. RÜDE, *Extreme-scale block-structured adaptive mesh refinement*,

- SIAM J. Sci. Comput., 40 (2018), pp. C358–C387, <https://doi.org/10.1137/17m1128411>.
- [47] Y. P. SITOMPUL AND T. AOKI, *A filtered cumulant lattice Boltzmann method for violent two-phase flows*, J. Comput. Phys., 390 (2019), pp. 93–120, <https://doi.org/10.1016/j.jcp.2019.04.019>.
- [48] P. A. SKORDOS, *Initial and boundary conditions for the lattice Boltzmann method*, Phys. Rev. E, 48 (1993), pp. 4823–4842, <https://doi.org/10.1103/physreve.48.4823>.
- [49] S. VENTURI, S. D. FRANCESCO, M. GEIER, AND P. MANCIOLA, *Forcing for a cascaded lattice Boltzmann shallow water model*, Water, 12 (2020), p. 439, <https://doi.org/10.3390/w12020439>.
- [50] S. VENTURI, S. D. FRANCESCO, M. GEIER, AND P. MANCIOLA, *Modelling flood events with a cumulant CO lattice Boltzmann shallow water model*, Natural Hazards, 105 (2020), pp. 1815–1834, <https://doi.org/10.1007/s11069-020-04378-x>.
- [51] S. VENTURI, S. D. FRANCESCO, M. GEIER, AND P. MANCIOLA, *A new collision operator for lattice Boltzmann shallow water model: a convergence and stability study*, Advances in Water Resources, 135 (2020), 103474, <https://doi.org/10.1016/j.advwatres.2019.103474>.
- [52] G. WELLEIN, T. ZEISER, G. HAGER, AND S. DONATH, *On the single processor performance of simple lattice Boltzmann kernels*, Comput. & Fluids, 35 (2006), pp. 910–919, <https://doi.org/10.1016/j.compfluid.2005.02.008>.
- [53] M. WITTMANN, T. ZEISER, G. HAGER, AND G. WELLEIN, *Comparison of different propagation steps for lattice Boltzmann methods*, Comput. Math. Appl., 65 (2013), pp. 924–935, <https://doi.org/10.1016/j.camwa.2012.05.002>.
- [54] J. ZHOU, *A lattice Boltzmann model for the shallow water equations*, Computer Methods in Applied Mechanics and Engineering, 191 (2002), pp. 3527–3539, [https://doi.org/10.1016/s0045-7825\(02\)00291-8](https://doi.org/10.1016/s0045-7825(02)00291-8).