KINETICALLY CONSISTENT THERMAL LATTICE BOLTZMANN MODELS

by

Parthib R. Rao

B.Tech, National Institute of Technology, Calicut, India, 2004

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2015

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Parthib R. Rao

It was defended on

March, 27th 2015

and approved by

Laura A. Schaefer, Ph.D., Professor, Department of Mechanical Engineering and Materials Science, University of Pittsburgh

Minking K. Chyu, Ph.D., Associate Dean for International Initiatives, University of Pittsburgh

Paolo Zunino, Ph.D., Assistant Professor, Department of Mechanical Engineering and Materials Science, University of Pittsburgh

Anirban Jana, Ph.D., Senior Scientific Specialist, Pittsburgh Supercomputing Center

Dissertation Director: Laura A. Schaefer, Ph.D., Professor, Department of Mechanical Engineering and Materials Science, University of Pittsburgh

Copyright © by Parthib R. Rao

2015

KINETICALLY CONSISTENT THERMAL LATTICE BOLTZMANN MODELS

Parthib R. Rao, PhD

University of Pittsburgh, 2015

The lattice Boltzmann (LB) method has developed into a numerically robust and efficient technique for simulating a wide variety of complex fluid flows. Unlike conventional CFD methods, the LB method is based on microscopic models and mesoscopic kinetic equations in which the collective long-term behavior of pseudo-particles is used to simulate the hydrodynamic limit of a system. Due to its kinetic basis, the LB method is particularly useful in applications involving interfacial dynamics and complex boundaries, such as multiphase or multicomponent flows. However, most of the LB models, both single and multiphase, do not satisfy the energy conservation principle, thus limiting their ability to provide quantitatively accurate predictions for cases with substantial heat transfer rates. To address this issue, this dissertation focuses on developing kinetically consistent and energy conserving LB models for single phase flows, in particular.

Firstly, we present a mathematical formulation of the LB method based on the concept of projection of the distributions onto a Hermite-polynomial basis and their systematic truncation. This formulation is shown to be capable of approximating the near incompressible, weakly compressible, and fully compressible (thermal) limits of the continuous Boltzmann equation, thus obviating the previous low-Mach number assumption. Physically it means that this formulation allows a kinetically-accurate description of flows involving large heat transfer rates. The various higher-order discrete-velocity sets (lattices) that follow from this formulation are also compiled. The resulting higher-order thermal model is validated for benchmark thermal flows, such as Rayleigh-Benard convection and thermal Couette flow, in an off-lattice framework. Our tests indicate that the D2Q39-based thermal models are capable of modeling incompressible and weakly compressible thermal flows accurately. In the validation process, through a finite-difference-type boundary treatment, we also extend the applicability of higher-order la ttices to flow-domains with solid boundaries, which was previously restricted.

Secondly, we present various off-lattice time-marching schemes for solving the discrete Boltzmann equation. Specifically, the various temporal schemes are analyzed with respect to their numerical stability as a function of the maximum allowable time-step . We show that the characteristics-based temporal schemes offer the best numerical stability among all other comparable schemes. Due to this enhanced numerical stability, we show that the usual restriction no longer applies, enabling larger time-steps, and thereby reducing the computational run-time. The off-lattice scheme were also successfully extended to higher-order LB models.

Finally, we present the algorithm and single-core optimization techniques for a off-lattice, higher-order LB code. Using simple cache optimization techniques and a proper choice of the data-structure, we obtain a 5-7X improvement in performance compared to an unoptimized code. Thereafter, the optimized code is parallelized using OpenMP. Scalability tests indicate a parallel efficiency of 80% on shared-memory systems with up to 50 cores (strong scaling). An analysis of the higher-order LB models also show that they are *less memory-bound* if the off-lattice temporal schemes are used, thus making them more scalable compared to the stream-collide type scheme.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

This dissertation primarily deals with a mathematical description of singlephase fluid flows. The choice of the description depends on the level of abstraction used to to describe or model a fluid and its dynamics. A fluid can be modeled using three broad levels (scales) of abstraction: the microscale, mesoscale, and macroscale. The choice of a particular abstraction ultimately depends upon the flow physics of interest, length and time-scales driving the flow, and increasingly, on the computational costs of the investigation. At each level of abstraction, the governing equations describing the physics are different.

As described in the sections below, the Newton-Hamilton equation governs the microscale, the Boltzmann equation governs the mesoscale, and the Navier-Stokes (N-S) equation governs the macroscale description. An approximate quantitative metric for the demarcation of the levels is the Knudsen number (Kn), which is the ratio of the molecular mean free path and the macroscale characteristic length scale of the flow. Fig. 1 describes the levels of abstraction based on Kn. Therefore, this chapter will first discuss the implications of each level of abstraction, before providing an overview of one approach (the lattice Boltzmann method) that can be used to overcome some of the computational complexities. The shortcomings of the lattice Boltzmann method will also be discussed, especially for thermal flows, as well as a brief summary of the improvements to the method proposed in this dissertation.

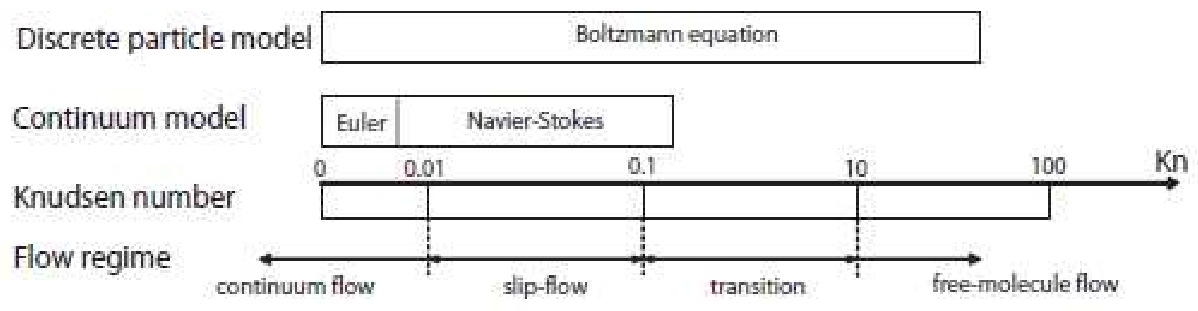


Figure 1: Classification of flow regimes based on the Knudsen number, Kn. The lower and upper limits for Kn number correspond to the macroscale (hydrodynamic regime) and microscale (free molecular flow regime), respectively. Adapted from [18]

1.1 LEVELS OF ABSTRACTIONS FOR FLUID FLOW

1.1.1 Microscale Description

In the microscale description, the motion of N number of particles (molecules or atoms) in an ensemble is modeled, such that their position and momentum can be obtained. The dynamics of the particles are governed by the Newton's laws of motion, which form a set of N, non-linear ordinary differential equations, given as:

$$m_i \frac{d\boldsymbol{x}_i^2}{dt^2} = \boldsymbol{F}_i = f_{ij} + \boldsymbol{G}_i \tag{1.1}$$

where $\boldsymbol{x}_i$ are a particle's position vectors, N is typically of the order of the Avagadro's number $N_A \sim 6 \times 10^{23}$, f_{ij} is the force exerted upon the i^{th} molecule by the j^{th} molecule, and $\boldsymbol{G}$ is an external body force, such as gravity [18]. The pairwise intermolecular forces, f_{ij}, driving the system can be expressed in terms of the gradient of a potential function V:

$$f_{ij} = -\nabla_i \sum_j V(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{1.2}$$

A popular model of V, for example, is the Lennard-Jones $12-6$ potential. Integration of the above equations of motion for the $N-$particle system yields the position and velocity of each molecule in time. This detailed microscale information can then be related to their macroscale counterpart through ensemble averaging, using laws of statistical mechanics. This methodology is broadly termed as Molecular Dynamics (MD) [93]. However, to describe even a small volume of fluid behavior, this procedure requires a very large number of particles ($> 10^{23}$), and complex interactions between the particles. Computationally, this implies tracking at least $6N$ variables (three positions and three velocities). An alternate description is to consider the Hamiltonian, $\mathscr{H}$, of the $N-$particle system, where $\mathscr{H}$ is the total energy of the system, including the kinetic and the potential energy due to molecular interactions. Given a definition of the Hamiltonian, the time evolution of $N-$particle system is governed by Hamilton's equation:

$$\frac{\partial \boldsymbol{p}_i}{\partial t} = -\frac{\partial \mathscr{H}}{\partial \boldsymbol{q}_i} \quad \frac{\partial \boldsymbol{q}_i}{\partial t} = \frac{\partial \mathscr{H}}{\partial \boldsymbol{p}_i} \tag{1.3}$$

with $(\boldsymbol{q}, \boldsymbol{p})$ being the generalized co-ordinates. However, with either formulations (Newtonian or Hamiltonian), due to the large number of required particles even for a small volume of fluid, the computational run-time is prohibitively expensive even with state of the art supercomputers and software techniques. Molecular dynamics is therefore usually used in the more fundamental investigations, such as computational chemistry, biology, material science, etc., where properties of molecules and their interactions (resulting in various physical phenomena) are more pertinent.

1.1.2 Mesoscale Description

1.1.2.1 Liouville Equation In the mesocale description of fluids, the $N-$particle system and their motion are *coarse-grained* using the concept of probability distribution functions (pdf). That is, instead of individual molecular trajectories, the $N-$particle mesoscale or kinetic level deals with distribution functions, $f_N(\boldsymbol{x}_1, \boldsymbol{v}_1 ... \boldsymbol{x}_N, \boldsymbol{v}_N, t)$, which are smooth *fields*, that describe the *joint probability* of finding molecule 1 at position $\boldsymbol{x}_1$ with speed $\boldsymbol{v}_1$, and molecule 2 at position $\boldsymbol{x}_2$ with speed $\boldsymbol{v}_2$, and so on, up to molecule N around position $\boldsymbol{x}_N$ with speed $\boldsymbol{v}_N$, all at the same time t. Trajectories are here replaced by the notion of a fluid in a phase-space $(\boldsymbol{v}, \boldsymbol{x}, t)$ obeying a $6N$-dimensional equation, known as the Liouville equation [61, 53]:

$$\frac{\partial f_N}{\partial t} + \sum_{i=1}^{N} \boldsymbol{v}_i \cdot \frac{\partial f_N}{\partial \boldsymbol{x}_i} + \boldsymbol{a}_i \cdot \frac{\partial f_N}{\partial \boldsymbol{v}_i} = 0, \tag{1.4}$$

where $\boldsymbol{a} = \boldsymbol{F}_i/m_i$ are the particle accelerations. The Liouville equation, valid for both equilibrium and non-equilibrium systems, is considered as bridge from classical mechanics (of particles) to statistical physics. However, although we have simplified the description of the N-body system by considering the probability distribution over N-particle phase space, this equation is still of limited practical use, since f_N is a still a continuum $6N-$ dimensional field.

1.1.2.2 Boltzmann Equation We can further simplify the description by simply integrating f_N over unwanted coordinates, by defining low-order *reduced particle distribution functions* (PDFs), which are defined as $f_M \equiv f_{12...M<N} = \int f_{12...N} dz_{M+1}...dz_N$, where $dz_k \equiv d\boldsymbol{x}_k d\boldsymbol{v}_k$, for $k = M+1, ...N$. This results in a chain of equations, called as the BBGKY hierarchy in the literature, since it was deduced independently by the researchers Bogoliubov, Born and Green, Kirkwood and Yvon:

$$\frac{\partial f_M}{\partial t} + \sum_{i=1}^{M} \boldsymbol{v}_i \cdot \frac{\partial f_M}{\partial \boldsymbol{x}_i} + \boldsymbol{a}_i \cdot \frac{\partial f_M}{\partial \boldsymbol{v}_i} = C_M \tag{1.5}$$

Notice that the right hand side now gathers all the effects of intermolecular interactions. A key feature of this hierarchy is that the one-particle PDF f_1 equation contains the two-particle PDF f_2, the two-particle PDF equation contains the three-particle PDF f_3, and so on. It is therefore clear that, numerically, the BBGKY hierarchy is as difficult to solve as the original Liouville equation due to its fully-coupled nature. Fortunately, most practical macroscale observables, such as density, pressure, temperature, and energy, only depend on 1 or 2-body distributions, and therefore, only the lowest order elements in the BBGKY hierarchy are of practical interest. The Boltzmann equation [43, 45], discussed in detail in Chap. 2, is a result of such an approximation. Essentially, in the Boltzmann equation, the N-body distribution f_N is approximated by a one-body pdf, or a single-body distribution function (SPDF) $f(\boldsymbol{x}, \boldsymbol{v}, t) \propto f_1(\boldsymbol{x}_1, \boldsymbol{v}_1, t)$:

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \nabla_{\boldsymbol{x}} f + \boldsymbol{g} \cdot \nabla_{\boldsymbol{v}} f = C(f, f) \tag{1.6}$$

Ludwig Boltzmann, however, derived the equation that bears his name from heuristic arguments, and not from the BBGKY hierarchy itself. Derivation of the Boltzmann equation from the BBGKY hierarchy can be found in the classic texts [53] and [61], a more recent one in [65].

The Boltzmann-like equations are fundamental to the notion of *quasi-particles*. By shifting the focus from actual particles (real atoms or molecules) to quasi-particles, the Boltzmann equation goes far beyond the original framework from which it was derived (i.e., rarefied gas dynamics), and is employed in a huge variety of fields in statistical mechanics, including neutron and radiation transport, electron transport in semiconductors, and many others. For hydrodynamics, a great deal of information can be extracted from the Boltzmann equation even without solving this equation fully. Mesoscale methods such as the Lattice Boltzmann method aim to do just that. The key observation is that even though the traditional kinetic theory of gases was originally derived for weakly-interacting (dilute) molecules, kinetic-like equations can be applied whenever strong interactions between elementary degrees of freedom (particles) can be cast in the form of weak interactions between collective degrees of freedom (quasi-particles) [114].

1.1.3 Macroscale Description

In the macroscale level of abstraction, the fluid is treated as a *continuum* regardless of its molecular structure. The continuum assumption implies that the hydrodynamic variables, such as density, (ρ), temperature, (T), and velocity $\boldsymbol{u} = (u_x, u_y, u_z)$, as well as their gradients, are continuous functions of space and time co-ordinates. It is also assumed that the spatial and temporal gradients of the hydrodynamic variables are small enough; i.e., the gradients are sufficiently smooth functions. At this level, the dynamics of a fluid can be described on the basis of conservation laws (mass, momentum and energy), which are a set of coupled non-linear partial differential equations [13, 66]:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) &= 0 \\
\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) &= -\nabla p + \nabla \cdot \boldsymbol{\tau} \\
\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} E) &= \nabla \cdot \boldsymbol{q} - p \nabla \cdot \boldsymbol{u} + \tau : \nabla \boldsymbol{u}
\end{aligned} \tag{1.7}$$

where p is the pressure, $\boldsymbol{q}$ is the heat flux and $\boldsymbol{\tau}$ is the deviatoric stress-tensor or shear stress. These equations are not closed since there are no first-principle relations for $\boldsymbol{\tau}$ and $\boldsymbol{q}$. The above equations can be qualitatively described as the statistically averaged equations of individual motions of fluid particles, with the critical property of satisfying the conservation laws exactly. Additionally, assuming a linear constitutive relationship between viscous stresses and strain rate (Newtonian fluid), and between heat flux and temperature gradient (Fourier's law), we obtain the widely used Navier-Stokes equations (N-S) for fluid flow [1]. The N-S equations describe a wide variety of single-phase flow regimes - incompressible and compressible flows, laminar and turbulent flows, viscous and inviscid flows, etc. It is important, however, to recognize the range of applicability, and therefore the limitations of the N-S equations. Due to their coarse-grained nature, the N-S equations surely cannot describe the macroscale phenomena caused or influenced by the omitted physical mechanisms, such as the phase transition (two-phase flows), the non-Newtonian stress-strain relationships

[1]In the literature, some authors refer only to the momentum equation as the Navier-Stokes equation, while others refer the entire set of conservation equations (mass, momentum, and energy) as the Navier-Stokes equations.

(soft-matter flows of such as polymers, bio-fluids, etc.), flows at the scale of the mean free path of particles (micro-scale flows), and other non-continuum effects.

In spite of the intuitive simplicity of the N-S equation, solving them in reality is a numerically very challenging. In fact, the existence and smoothness of N-S solutions is one of the six remaining Clay Millennium Prize problems in classical (non-quantum) physics [39]. From a computational perspective, much effort is invested in developing numerical approaches to solving this equation, rather than adding more complexity through additional physical phenomena, except for, perhaps, the field of turbulence modeling. The general procedure to solve N-S equations is to discretize the governing equations in the computational domain by different methods, such as FD, FV, or FE, and then the corresponding algebraic equations are solved iteratively to obtain macroscopic variables at each mesh point in a particular numerical grid [60].

Having discussed the levels of abstractions for fluid flows, in conclusion, we can infer that a series of systematic assumptions, approximations, and averaging lead us from the Newton-Hamilton equations for a $N-$particle system, through Liouville and Boltzmann equation for the spdf, to the continuum-based Navier-Stokes/Euler equations. Fig. 2 shows a pictorial view of this hierarchy [112].

1.1.4 Why Mesoscale Methods?

Continuum-assumption based modeling is sufficient for many fluid flow problems; however, many macroscale fluid phenomena have significant effects from the underlying microscopic world. Surface tension is a prime example: cohesive (attractive) forces between molecules are responsible for phase separation on the macroscopic scale and the concomitant interfacial dynamics. However, modeling such a phenomena using continuum-based methods leads to difficulties, since the governing equations are on the macroscale, whereas the fundamental mechanism giving rise the interfacial dynamics is on the microscale. Gaseous flow in microchannels, especially in the presence of wall boundaries (large Kn flows) is another example, where continuum-based methods are insufficient. Mesoscale modeling techniques are useful in such contexts, where we need to simulate macroscale flows, which also exhibit

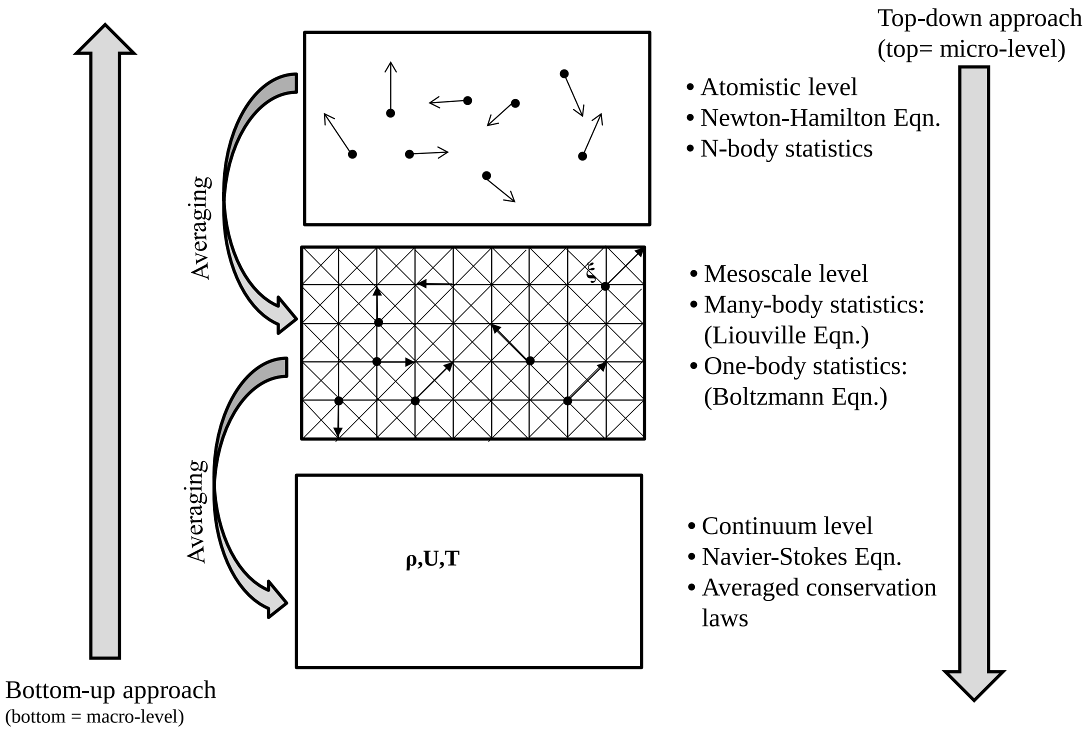


Figure 2: Hierarchy of fluid dynamic description

significant underlying microscale phenomena. Mesoscale methods provide the linkage to the microscale and macroscale worlds, through the fields of statistical mechanics and kinetic theory. Many such mesoscale methods have been developed, some of which solve the Boltzmann equation directly, such as the lattice Boltzmann (LB) method, the discrete velocity method (DVM), the gas-kinetic scheme (GKS), etc.; and some of which simulate psuedo-particles directly, such as the Lattice Gas Cellular Automata (LGCA), Dissipative Particle Dynamics (DPD), and Direct Simulation Monte Carlo (DSMC) method. All of these techniques do not track individual particles as in MD, but a cluster of pseudo-particles, statistically represented by their PDFs.

1.2 LATTICE BOLTZMANN METHOD

Among the mesoscale methods, the LB method has increasingly been adopted as an alternative numerical scheme for the simulation of a wide variety of fluid flows-not just singlephase flows. [4, 27, 112, 128]. The theoretical foundation for using simplified kinetic models to simulate hydrodynamic systems (or other complex systems) is based upon the observation that in nature macroscale phenomena, such as hydrodynamics, are rather insensitive to the underlying details of the microscale dynamics. In hydrodynamic systems, details of microscopic dynamics can affect the numerical values of the transport coefficients, but not the overall form of the Navier–Stokes equations.

The LB method, is therefore, a minimal form of the discrete Boltzmann equation, that primarily reproduces Navier-Stokes level hydrodynamics in the limit of small Kn. It is minimal, because only a handful of discrete values $\boldsymbol{v} \rightarrow \boldsymbol{v}_i$ are retained from velocity space, that are required to preserve conservation laws and to recover the correct continuum space-time *symmetries* (Galilean, translational and rotational invariance). This makes the LB method an indirect N-S solver. Additionally, the LB method can also accommodate inter-particle forces that describe many complex flow phenomena, without having to solve the complicated kinetic equations or their molecular dynamics. Thus, the LB method is suited for problems involving interfacial dynamics, particulate/suspension flows, multiphase flows, Kn-transition flows, porous media flows, and binary and ternary complex flows, etc. It can be argued that the versatile nature of the LB method is an asset, but at the same time, also a potential source of confusion regarding its applicability [112].

1.2.1 A Brief History

Although the history of the LB method is documented in several places, a brief discussion on it is useful here, because many of the new model developments in state-of-the-art LB advances still have their basis in the older formulations and models, upon which they are built. Consistent with most numerical techniques, the LB method has gone through an evolutionary process over the past three decades, the highlights of which are described below:

LGCA-based formulation

The LB method originated from its predecessor - the lattice gas cellular automata (LGCA), and in particular the FHP model [40], quite independent of the kinetic theory. The fundamental idea behind LGCA is that microscopic interactions of pseudo-particles on a microscopic lattice can lead to averaged macroscopic equations. For the interaction, which consists of collision and propagation (or streaming) of particles with lattice velocities, lattice (discrete) symmetry plays a key role for conserving mass and momentum, as well as ensuring the angular momentum conservation. However, the FHP model had several shortcomings, such as statistical noise, lack of Galilean invariance, low Reynolds number, difficult extension in three-dimensions, etc. The LB method originated from the efforts to address these shortcomings of LGCA. Through a series of landmark advancements, such as replacement of Boolean variables by real-valued variables, linearization of collision operator, etc., the field of the lattice Boltzmann-BGK (LBGK) model was developed, which addressed the shortcomings of LGCA successfully [14, 58, 59, 63, 77]. It is noted that the original LBM formulations of Qian *et al.* [90], and Chen *et al.* [29] were limited to isothermal low-Mach number (near-incompressible) flows, and did not provide a systematic way of deriving non-isothermal lattice Boltzmann models that also required conservation of energy. The earliest LB models, derived from LGCA, are sometimes termed as the first-generation LB models.

Continuous Boltzmann-BGK based formulation

The LGCA-based formulation did not have a formal connection to the Boltzmann equation or the kinetic theory of gases. The link to LGCA was broken in the seminal papers by Abe [2] and He and Luo [55], where the LB method was derived from the direct phase-space discretization of the (continuous) Boltzmann-BGK equation. That is, the LB method was proven to be a special finite-difference form of the continuous Boltzmann-BGK equation. Although the resulting method retained the structural form as the previous LGCA-based formulation, it was mathematically rigorous, (partially) consistent with kinetic theory, and had better stability and accuracy. More importantly, this formulation enabled large-scale extension of the LB method to practical engineering flows problems through adaptive grids, non-Cartesian grids, etc. However, just like the LGCA-based formulation, due to the small Mach number (near-incompressibility) assumption implicit in the derivation [55], this formulation is restricted primarily to isothermal, low-speed flows. The LB models based on He and Luo's formulation are termed as the second-generation LB models.

Grad's Hermite-expansion based formulation

Inspired by Harold Grad's original work on the kinetic theory of rarefied gases [45], Shan and He [104, 105] proposed a new theoretical formulation for discretizing the Boltzmann-BGK equation in phase-space, which links the lattice Boltzmann method to the Grad 13-moment system. The idea of this formulation is to project the distribution functions on Hermite polynomials, and the subsequent Gauss–Hermite quadrature to discretize the continuous microscopic velocity space. Truncation of the Hermite expansion of the distribution functions at a particular order can be done *a priori*, depending upon the flow physics to be modeled. This *kinetically-consistent* formulation is general in nature and can recover the previous LB models as special or limiting cases. It also places the LB method on a more sound and systematic mathematical basis. This formulation can construct models that can not only simulate the incompressible limit of the Navier–Stokes equations as previous models, but also isothermal compressible fluids, and theoretically, also model thermal flows or flows that require energy conservations. Similarly, two other LB formulations, namely the multiple-relaxation time models (MRT) and entropic LB models (ELB) were also developed with different collision models to address the shortcomings of the BGK approximation, such as fixed Prandtl number (addressed by the MRT models), stability at high Reynolds number flow and thermohydrodynamics (addressed by the entropic LB models), etc. The Hermite-based LB formulation, MRT models [37], and the entropic LB models [8, 31] are all, in general, called the third-generation LB models.

1.3 MOTIVATION

Over the past twenty years, the LB method has undergone a rapid evolution in terms of physical models, numerics, computer implementations, and engineering-scale applications. Despite this progress, there are still several deficiencies and areas of potential progress. One of the most important voids in the the LB field is the lack of a *kinetically-consistent* thermohydrodynamic model or a thermal LB model. By thermal model we mean a model that guarantees not only momentum conservation, but also *energy conservation*. For example, flows with significant heat transfer, flows without/beyond the Boussinesq approximation, etc. The most widely used LB model, across a wide spectrum of simple and complex flows, is based on the so-called D2Q9 velocity lattice, which is fundamentally based on an isothermal assumption, and therefore energy conservation cannot be strictly enforced. Thus, although this lattice configuration has been proven successful for a large number of low-speed isothermal fluid flows, extending this lattice configuration for simulating flows with large heat transfer has not been nearly as successful. Several attempts have been made to develop thermal LB models, a brief review of which is given in Sec. 5.2. However, many of these are phenomenological models, and may not be kinetically consistent with respect to energy and its conservation, and therefore limited in their applicability. A consistent kinetic scheme for thermohydrodynamics requires that pressure, temperature and density are related to each other, for example through equations of state, the definition of speed of sound, etc., throughout the entire evolution process.

The inability to model energy conservation by the D2Q9 lattice is manifested in the flow simulations based upon it. For example, although many of the current multiphase models based on D2Q9 describe the multiphase flow dynamics qualitatively, numerically stable and quantitatively accurate models are still lacking. The problem is even more severe when a phase change takes place at the interface, for example, during droplet evaporation or vapor-bubble growth. Heuristic thermal models for single-phase flows, such as the passive-scalar model, therefore, cannot be accurately extended to multiphase flows. Energy conserving models are also necessary for singlephase compressible flows, aeroacoustics simulations, microscale flows, etc. Thus, given the wide potential of the LB method in modeling various fluid flows, it is essential that an accurate and stable underlying thermal model is developed.

In spite of the previous limited success and the theoretical challenges in developing a consistent thermal LB model, the third generation LB models, namely the entropic LB and the Hermite polynomial based LB formulations, offer promise. In this dissertation we will focus on the Hermite polynomial based LB formulation. Specifically, in this method, through a mathematically rigorous procedure, the LB method is derived as a moment solution of the continuous Boltzmann-BGK equation using the Hermite polynomials as the expansion basis. The equilibrium distribution in the discrete model is obtained by truncating the Hermite expansion of the Maxwellian distribution. Importantly, the choice of the order of truncation is done on *a priori* basis. The resulting kinetic equations thus guarantee the correct hydrodynamic equations as long as sufficient number of moments (flow variables) are preserved by the truncation, and accurately represented by the discrete velocities. Through this systematic formulation, the N-S energy equation can be approximately recovered up to a small error in the thermal diffusivity if the third moments of the Maxwellian are preserved. It is exactly recovered if the fourth order moments are preserved. This way, thermal LB models can be constructed consistently without using the previous small-Mach number assumption.

1.4 OBJECTIVES

Although the theory of the Hermite-based LB formulation has been established recently, several implementation aspects remain to be thoroughly investigated. One of the main challenges is the nature of the *higher-order lattices* which are required for thermal models under the new formulation. Unlike the popular D2Q9 lattice which has only one speed in each coordinate direction, the lattices required for a thermal model have multiple speeds in each direction (extended neighborhood). Due to this feature, higher-order lattices are challenging from an implementation point of view, especially in the presence of wall boundaries, since the commonly used bounce-back scheme cannot be extended easily. In fact, it has been observed that a particular extension of the bounceback scheme for the higher-order lattices results in severe inaccuracies [97]. Therefore, alternate schemes to solve the kinetic Boltzmann-BGK equation in the presence of wall boundaries are required. Off-lattice temporal evolution schemes-an Eulerian analogue of the Lagrangian stream-collide scheme- may provide such an alternative. These temporal schemes are also commonly referred to as off-lattice Boltzmann schemes.

In summary, then, the main focus of this work is the detailed, *implementation aspects* of the Hermite-based LB formulation for thermal flows, especially in the off-lattice Boltzmann framework. Specifically the broad objectives of this work are two-fold:

1. To develop, analyze, and implement off-lattice Boltzmann schemes in presence of wall boundaries, and analyze its numerical stability and accuracy; and
2. To extend the off-lattice Boltzmann scheme to solve the Boltzmann-BGK equation with higher-order lattices (D2Q37, for example) for singlephase thermal flows.

1.5 ORGANIZATION

This dissertation is organized in two parts. Part I, consisting of Chaps. 2 and 3 (along with Appendices. A and B), lays the mathematical foundations of the new LB formulation; while Part II, consisting of Chaps. 4, 5, and 6 presents the implementation aspects of the models developed in Part I. Specifically, in Chap. 2, we present a brief background on the Boltzmann equation and its connection to hydrodynamics, and then, a systematic derivation of the hydrodynamic equations from the Boltzmann-BGK equation, based on the method of projection of the distributions on to a Hermite polynomial basis. This procedure permits a *a priori* construction of the discrete thermal LB models directly from the continuous Boltzmann-BGK equation. Chap. 3 discusses the velocity discretization procedure, based on Gauss-Hermite quadrature, that is necessary to capture the moments of the truncated distribution, especially those relevant to thermal flows. Chap. 4 discusses various alternate space-time discretization schemes for the discrete Boltzmann equations, and their numerical stability and accuracy. In Chap. 5, the models developed in Part I are validated on benchmark thermal flows, especially in an off-lattice framework. We then, in Chap. 6, discuss the computer implementation, along with optimization and parallelization techniques that can be used for simulations with the higher-order models. Finally, in Chap. 7, the work is summarized, conclusion are provided, and avenues for future work are identified.

2.0 THE BOLTZMANN EQUATION FOR HYDRODYNAMICS

Within the context of this dissertation, it is not practical to write a complete, self-contained account of the evolution of the Boltzmann equation. For that, an interested reader can refer to the classic texts by Cercignani [21], Harris [53] or Huang [61]. Only the essential concepts from the kinetic theory of gases, which have considerable base in the derivation of the numerical models are presented in Sec. 2.1. Next, in Sec. 2.2, the reasoning is shown that leads from the Boltzmann equation to the macroscopic conservation equations governing fluid flow. Within that context, a detailed mathematical derivation of the Euler and Navier-Stokes-like equations is presented in Sec. 2.3, based on the projection of the Boltzmann distribution on the Hermite polynomial bases - a method first formulated by Grad in 1949 [45], and recently extended by Shan *et al.* [105]. This is done with an overarching goal of obtaining conservation laws suitable for *thermal* flows, directly from the Boltzmann equation.

2.1 THE BOLTZMANN EQUATION

Ludwig Boltzmann laid the foundations for kinetic theory - a special case of statical mechanics. In essence, he postulated that the knowledge of the position and velocity of individual particles is not important, but rather their *distribution* in the velocity and configuration space, i.e., what percentage of molecules at a particular location have velocities within a certain range, and at a certain time. Accordingly, the single-particle distribution function (SPDF), $f(\boldsymbol{\xi}, \boldsymbol{x}, t)$, gives the probability of finding a particular particle with a given position, $\boldsymbol{x}$, and velocity, $\boldsymbol{\xi}$, at a particular time t. For example, for a gas with N number of particles, the number of particles having velocities in the $\boldsymbol{x}$ direction between ξ_x and $\xi_x + d\xi_x$

is $Nf(\xi_x)$. Thus, $f(\boldsymbol{\xi}, \boldsymbol{x}, t)d^3\boldsymbol{x}d^3\boldsymbol{\xi}$ gives the total number of particles, at time t, located in a control volume $d^3\boldsymbol{x}$ around position $\boldsymbol{x}$, and which have a velocity located in the volume $d^3\boldsymbol{\xi}$ around $\boldsymbol{\xi}$. The evolution of the SPDF, a fundamental variable in kinetic theory, is described by the Boltzmann equation:

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} f + \boldsymbol{g} \cdot \nabla_{\boldsymbol{\xi}} f = Q(f, f), \tag{2.1}$$

where the left hand side represents free streaming of the particles in the phase space, the collision term, $Q(f, f)$, represents the time rate of change in f due to inter-particle collisions, and $\boldsymbol{g}(\boldsymbol{x}, t)$ is the acceleration due to the force field, such that $\boldsymbol{F}(\boldsymbol{\xi}) = -\boldsymbol{g} \cdot \nabla_{\boldsymbol{\xi}} f$. The gradient operators in the configuration and velocity spaces are denoted by $\nabla_{\boldsymbol{x}}$ and $\nabla_{\boldsymbol{\xi}}$, respectively. Hereafter, we will use the term distributions to imply the single-particle distribution function.

The Boltzmann equation in itself can be derived from the BBGKY hierarchy using assumptions known as the Boltzmann-Grad limit (BGL). The physical and mathematical implications of the Boltzmann-Grad limit are provided in the classic works [21, 43, 45, 65], and a more intuitive description in [22]. A few of the most significant assumptions are:

1. Density of the gas is low enough so that only binary collisions take place between the constituent particles under a short-range two-body potential;

2. Collisions are localized in physical space;

3. The effect of external forces on the particles during the mean collision time is negligible in comparison with the interacting molecular forces;

4. Since the probability of multi-particle configurations is separated into a probability of single particle configurations, the velocities before and after the collision do not correlate with each other. This assumption is also called the molecular chaos assumption: $f_{12}(\boldsymbol{x}_1, \boldsymbol{\xi}_1, \boldsymbol{x}_2, \boldsymbol{\xi}_2, t) = f_1(\boldsymbol{x}_1, \boldsymbol{\xi}_1, t) f_2(\boldsymbol{x}_2, \boldsymbol{\xi}_2, t)$. This also means that the period between two collisions is much longer than the duration of the collision itself.

2.1.1 Link to Hydrodynamic Variables

The distribution function enables us to calculate the macroscopic flow variables involved in the hydrodynamic description of the fluid. In general, the hydrodynamic variables can be expressed as the first few velocity moments of the distribution function. Moments are easily defined up to an arbitrary order, but it is the first few moments (≤ 3) that have a physical meaning.

To repeat, the quantity $fd^3\boldsymbol{\xi}d^3\boldsymbol{x}$ gives the average number of particles, N, contained in a volume element $d^3\boldsymbol{x}$ about $\boldsymbol{x}$, and a velocity-space element $d^3\boldsymbol{\xi}$ about $\boldsymbol{\xi}$ at time t. So, when this distribution is integrated in phase space between $\boldsymbol{\xi}$ and $\boldsymbol{\xi} + d\boldsymbol{\xi}$, we lose this detailed information (coarse-graining), and we get statistically averaged values, which are the macroscopic hydrodynamic variables of interest. Specifically, the number density n, density ρ, velocity $\boldsymbol{u}$, and the internal energy per unit mass, ϵ, are given as:

$$\begin{aligned} \rho(\boldsymbol{x},t) \equiv mn(\boldsymbol{x},t) &= m\int fd\boldsymbol{\xi}, \\ \rho\boldsymbol{u}(\boldsymbol{x},t) &= m\int \boldsymbol{\xi}d\boldsymbol{\xi}, \\ \rho\epsilon(\boldsymbol{x},t) &= \frac{1}{2}m\int |\boldsymbol{\xi}-\boldsymbol{u}|^2 d\boldsymbol{\xi} \end{aligned} \tag{2.2}$$

where m is the molecular mass of the gas particle, and $f \equiv f(\boldsymbol{\xi},\boldsymbol{x},t)$ for brevity. Density is referred as the zeroth moment, momentum as the first, and internal energy as the second moment of the distribution function. These are conventional hydrodynamic variables that correspond to conservation of mass, momentum and energy. The higher-order ($n \geq 2$) moments of the distribution function are described more conveniently on the basis of the so-called peculiar or intrinsic velocity, $\boldsymbol{c} \equiv \boldsymbol{\xi} - \boldsymbol{u}(\boldsymbol{x},t)$. The peculiar velocity gives the particle speed as measured by an observer moving with the fluid at the local bulk flow velocity $\boldsymbol{u}$, i.e., an stationary observer.

Some other hydrodynamic quantities of interest are the pressure p and heat flux $\boldsymbol{q}$. They are related to the second $\boldsymbol{P}$ and third moments $\boldsymbol{Q}$, respectively, which are defined as:

$$\boldsymbol{P} = m\int(\boldsymbol{\xi}-\boldsymbol{u})(\boldsymbol{\xi}-\boldsymbol{u})d\boldsymbol{\xi} = m\int \boldsymbol{cc}fd\boldsymbol{c}, \tag{2.3}$$

$$\boldsymbol{Q} = m\int(\boldsymbol{\xi}-\boldsymbol{u})(\boldsymbol{\xi}-\boldsymbol{u})(\boldsymbol{\xi}-\boldsymbol{u})d\boldsymbol{\xi} = m\int \boldsymbol{ccc}fd\boldsymbol{c}. \tag{2.4}$$

The elements of the second-order tensor $P_{\alpha\beta}$ represent the the rate of momentum transfer of the α component in the β direction. Under the assumption of negligible long-range intermolecular interactions, which is consistent with the Boltzmann-Grad limit, the hydrodynamic pressure p can be found as the average of the diagonal components of $P_{\alpha\beta}$:

$$p = \frac{P_{\alpha\alpha}}{D} = \frac{2\rho\epsilon}{D} \tag{2.5}$$

Further, if we assume a monatomic gas under consideration, we can relate the internal energy of the gas to its temperature, θ, using the the equipartition theorem:

$$\rho\epsilon = \frac{D}{2}\frac{\rho k_B}{m}\theta \quad \text{or} \quad \theta = \frac{2m\epsilon}{Dk_B} \tag{2.6}$$

where D is the physical dimensions of space, and k_B is the Boltzmann constant. From Eqs. (2.5) and (2.6), we obtain an equation of state relating pressure, temperature and density:

$$p = nk_B\theta \tag{2.7}$$

Eq. (2.7) is same as the ideal-gas equation of state, which is consistent with our previous assumptions of the gas being monatomic. Heat or internal energy flux through the fluid due to molecular motion, $\boldsymbol{q}$, which is one of the component of the total energy flux through the fluid, is by definition the contraction of the third-order moment $\boldsymbol{Q}$:

$$\boldsymbol{q} = \int f|\boldsymbol{\xi} - \boldsymbol{u}|^2(\boldsymbol{\xi} - \boldsymbol{u})d\boldsymbol{\xi} = \frac{1}{2}\int \boldsymbol{c}\boldsymbol{c}^2 d\boldsymbol{c} \tag{2.8}$$

2.1.2 Boltzmann-BGK equation

2.1.2.1 Collision Models Due to the complexity of the collision integral and the large phase-space dimensions, a direct solution of the full Boltzmann equation with the non-linear collision is difficult to solve both analytically and numerically. Hence, simpler model expressions for the collision operator are employed depending upon the physics to be modeled. From a continuum-based flow modeling perspective, the idea behind collision models (assumptions) is that the large amount of small-scale details of binary particle interactions does not significantly influence macroscopic hydrodynamic quantities [21, 128].

In order to simply the collision integral $Q(f, f)$ to a more tractable integral $\Omega(f)$, i.e, $Q(f, f) \to \Omega(f)$, a potential collision model has to obey two inviolable rules. The first one is that the collision model $\Omega(f)$ conserves all the collision invariants, $\psi_k(\boldsymbol{\xi})$, such that:

$$\int \psi_k(\boldsymbol{\xi})\Omega(f) = 0 \tag{2.9}$$

where the five elementary collision invariants defined as $\psi_0 = 1$, $\psi_1, \psi_2, \psi_3 = \boldsymbol{\xi}$, and $\psi_4 = \boldsymbol{\xi}^2$ are related to mass, momentum, and kinetic energy. A corollary of the requirement of conservation of collisional invariants is that a collision model should conserve mass, momentum and energy. The second rule is that a collision model should have the tendency of leading the distributions towards a *local* equilibrium distribution function through the H theorem [128].

Local Equilibrium: The concept of local equilibrium is critical to the Boltzmann equation's ability to model Navier-Stokes flows. Local equilibrium is that special *positive* value of the distribution function, denoted by $f^{(0)}$, such that the collision integral vanishes:

$$Q(f^{(0)}, f^{(0)}) = 0 \tag{2.10}$$

Without going into the details of the derivation, it can be shown the Maxwell-Boltzmann distribution (Maxwellian), given below in a reference frame moving with the fluid, is such a special distribution function:

$$f^{(0)}(\boldsymbol{\xi}, \boldsymbol{x}, t) = \rho\Big(\frac{m}{2\pi k_B\theta}\Big)^{D/2} \exp\Big[-\frac{m(\boldsymbol{\xi}-\boldsymbol{u})^2}{2k_B\theta}\Big] \tag{2.11}$$

The condition given by Eq. (2.11) represents a local or metastable equilibrium. In other words, if the system is in local equilibrium, the distribution function is Maxwellian.

It is important to distinguish between the concepts of global and local equilibrium. A system is in global equilibrium when the macroscale velocity and internal energy (temperature) are constant throughout the physical flow domain. This happens on a much larger time-scales, $T \ggg \tau$. On the other hand, at time-scales $\mathcal{O}(\tau)$, the system reaches equilibrium locally, such that the right hand term of Boltzmann equation goes to zero, but not necessarily the left hand side. So, on time-scales $\mathcal{O}(\tau)$, it is still possible to have spatial or time gradients of the distributions, and therefore of the macroscale quantities.

2.1.2.2 BGK Collision Model All of the above collision constraints are fulfilled by the so-called Bhatnagar-Gross-Krook (BGK) approximation, also known as the Single Relaxation Time (SRT) model [16]. The BGK approximation-the most widely used collision approximation for modeling hydrodynamic flows-is a milestone development in the LB method [58] and, was the key to the development of LBM as a tool to model Navier-Stokes flows. The BGK approximation linearizes the collision integral by assuming the collision process changes f by an amount proportional to the departure of f from the local Maxwellian distribution function $f^{(0)}$. The BGK model is given as:

$$\Omega(f)_{BGK} = -\frac{1}{\tau}(f(\boldsymbol{\xi}, \boldsymbol{x}, t) - f^{(0)}(\boldsymbol{\xi}, \boldsymbol{x}, t)) \tag{2.12}$$

The coefficient τ, is the *characteristic time between collisions*, also called the *relaxation time*. On the microscale, τ influences the details of the collision process, while at the macroscale level, it influences the magnitude of the transport coefficients ν, α,through the the Chapman-Enskog expansion procedure, as explained in Sec. 2.3. That is, τ is a free parameter which can be tuned to match the transport properties of the fluid being modeled. However, with just a single relaxation time, the BGK approximation implies that mass, momentum, and energy all relax at the same rate, which is true only for ideal gases. In physical terms, this approximation also implies that momentum diffusivity (ν), and thermal diffusivity (α) are equal, or $Pr = 1$.

It is mentioned in passing that the BGK approximation is, in fact, a special case of the general quasi-linear collision operator, A_{ij}, such that $\Omega(f) = A_{ij}(f - f^{(0)})$. A_{ij} is also called as the scattering matrix, whose eigenvalues and eigenvectors control the fluid flow through relaxation of macroscopic quantities, as the mass, the momentum, the viscosity and the energy [63, 59]. With the BGK approximation, it is assumed that the whole scattering matrix is replaced by only one leading non-zero eigenvalue of A_{ij} that controls the viscosity, i.e. $A_{ij} \to -\omega\delta_{ij}$ [112].

Incorporating the BGK approximation, Eq. (2.1) can be re-written as:

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} f + \mathbf{g} \cdot \nabla_{\boldsymbol{\xi}} f = -\frac{1}{\tau}(f - f^{(0)}), \tag{2.13}$$

This equation is called as the *Boltzmann-BGK equation.* Although the BGK collision appears linearized, it is highly non-linear, since $f^{(0)}$ is a complicated function of the hydrodynamic variables which are themselves moments of f in the velocity phase.

2.1.2.3 Non-dimensionalisation The Boltzmann-BGK equation has to be recast in a non-dimensional form so that it is amenable to general computations. The choice of the characteristic speed to be chosen is critical since its has significant implication of the subsequent discretization process. The characteristic speed is chosen to be $c_s \equiv \sqrt{k_B\theta_0/m_0}$, where θ_0 and m_0 are the characteristic temperature and molecular mass of a single-component fluid, respectively. c_s can be identified as the speed of sound corresponding to θ_0[1]. The characteristic length scale l_0 and time scale t_0 are chosen such that $c_s = l_0 t_0$. With these characteristic variables, the other variables in Eq. (2.13) can be non-dimensionalized as described in Tab. 1:

Omitting the caret notation for convenience, the non-dimensional Boltzmann-BGK retains its original form, except that now all the variables are non-dimensional:

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} f + \mathbf{g} \cdot \nabla_{\boldsymbol{\xi}} f = -\frac{1}{\tau}(f - f^{(0)}), \tag{2.14}$$

[1]The notation for characteristic speed is purposely chosen to be c_s and not c_0 so as to maintain consistency with literature

Table 1: Non-dimensional variables in the Boltzmann-BGK equation.

Variable	Non-dimensional form	Variable	Non-dimensional form
PDF	$\hat{f} = \frac{f}{\rho_0}$	Macroscale velocity	$\hat{\boldsymbol{u}}_\alpha = \frac{\boldsymbol{u}_\alpha}{c_s}$
Microscale velocity	$\hat{\boldsymbol{\xi}} = \frac{\boldsymbol{\xi}}{c_s}$,	Body force	$\hat{\boldsymbol{g}}_\alpha = \frac{\boldsymbol{g}_\alpha l_0}{c_s^2}$
Length	$\hat{x}_\alpha = \frac{x_\alpha}{l_0}$	Time	$\hat{t} = \frac{t c_s}{l_0}$
Density	$\hat{\rho} = \frac{\rho}{\rho_0}$	Relaxation time	$\hat{\tau} = \frac{c_s \tau}{l_0} = \frac{\tau}{t_o}$
Spatial derivative	$\frac{\partial}{\partial x} = \frac{\partial}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} = \frac{1}{l_0} \frac{\partial}{\partial \hat{x}}$	Temporal derivative	$\frac{\partial}{\partial t} = \frac{\partial}{\partial \hat{t}} \frac{\partial \hat{t}}{\partial t} = \frac{c_s}{l_0} \frac{\partial}{\partial \hat{t}}$

where the non-dimensional local equilibrium function, $f^{(0)}$, becomes:

$$f^{(0)} = \frac{\rho}{(2\pi\theta)^{D/2}} \exp\left[-\frac{\boldsymbol{c}^2}{2\theta} \right], \tag{2.15}$$

and the equation of state is:

$$p = \rho\theta \tag{2.16}$$

for isothermal flows ($\theta = 1$), and the non-dimensional speed of sound, $c_0 = 1$. Note that here the non-dimensional relaxation time $\tau = c_s\tau/l_0 = \lambda/l_0$, where λ is the mean free path of the particles, and $Kn = \lambda/l_0$. As a result $\tau = \mathcal{O}(Kn)$. The Boltzmann-BGK equation forms the basis for the majority of research in the lattice Boltzmann field.

2.1.3 Boltzmann-BGK to Conservation Laws

We are now interested in the continuum or conservation laws that result from the limiting behavior of the Boltzmann-BGK equation. From Eq. (2.2), we can see that the hydrodynamic variables, ρ, u, ϵ are the low-order moments of the distribution function. Therefore, the hydrodynamic equations can be obtained by taking various moments of the Boltzmann-BGK equation. This procedure is available in the classic texts [61, 53], and hence not detailed here. In brief, the procedure consists of taking moments or multiplying Eq. (2.13) with $1, \boldsymbol{\xi}, \frac{1}{2}\xi^2$, integrating the resulting equations over the velocity space, and using Eqs. (2.2) and (2.9). This procedure leads to the following conservation laws:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$$

$$\frac{\partial(\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) + \nabla \cdot \boldsymbol{P} = \rho \boldsymbol{g} \tag{2.17}$$

$$\frac{\partial(\rho\epsilon + \frac{1}{2}\rho \boldsymbol{u}^2)}{\partial t} + \nabla \cdot \left[\boldsymbol{u} \left(\rho\epsilon + \frac{1}{2}\rho \boldsymbol{u}^2 \right) + \boldsymbol{P} \cdot \boldsymbol{u} + \boldsymbol{q} \right] = 0$$

which are the generic hydrodynamic equations for conservation of mass, momentum and energy.

From a mesoscale point of view, there are 13 independent variables (moments) in the flow system. Namely, one moment for the density ρ, three for the velocity u_α, six for the stress tensor $P_{\alpha\beta}$, and three for the energy flux q_α. So, thirteen equations are needed to close this system. However, Eq. (2.17) represents five equations in total. Clearly, the conservation equations are not closed because of the presence of the higher-order moments of the stress tensor and energy flux. Therefore, to write a closed set of equations, one has to approximate the stress tensor and the energy flux in terms of the lower-order, conserved moments.

Two different approaches have been developed to handle the closure problem. One is the well known method of moments of Grad [45], also known as the Grad's 13 moment equations, and the other is the popular Chapman-Enskog multiscale expansion [24]. In the former approach, Grad stated that the above five equations are not sufficient to describe all the properties of the fluid, and hence another set of equations were proposed for stress tensor and energy flux, by projecting the distribution function on Hermite polynomials. While in the latter approach, according to Chapman and Cowling, the distribution function, as well as temporal and spatial variables, are expanded asymptotically in different powers of Kn, so that the system is separated to multiple length and time scales. By investigating the projection of the distributions on lattice symmetries (rooted in lattice gas automata) in different scales, approximations can be made for the stress tensor and energy flux using the lower-order moments and their derivatives with respect to time and space, and thus finally the hydrodynamic equations of Euler and Navier-Stokes can be derived. However, the Chapman-Enskog expansion procedure becomes difficult for systems requiring moments higher than second-order, or in other words, the expansion can be easily applied to obtain the isothermal Navier-Stokes, but obtaining thermal (compressible) Navier-Stokes and beyond becomes difficult.

Comparatively, Grad's representation approach of projecting the distribution function on Hermite polynomials and truncating it at a certain order is relatively easy without loss of the accuracy up to the truncated order. Unfortunately, the Grad 13-moment equations are just as difficult to solve as the Chapman-Enskog expansion for higher moments. Nevertheless, as put forth in [105], the Chapman–Enskog expansion as a procedure can be useful for providing a theoretical measure in determining the order of truncation in the Hermite series for a given desired accuracy requirement. In other words, we can use the Chapman–Enskog expansion to infer a *sufficient condition* for the terms retained in the truncated Hermite basis that is required for describing flows of certain Knudsen number orders. We will follow this idea, the details of which are provided in the subsequent sections.

2.2 HERMITE POLYNOMIALS-BASED LB FORMULATION

The following theoretical formulation of the LB method is based on the projection of distribution function onto a Hermite polynomial bases, originally proposed by Harold Grad in 1949 [45, 46], and developed further recently by Shan, Chen, and co-workers [80, 104, 105, 103, 101]. Specifically, the solutions to the Boltzmann-BGK equation are sought by projecting the distribution function, f, onto a sub-space spanned by the Hermite polynomials in velocity space, $\boldsymbol{H}(\boldsymbol{\xi})$. The reason Hermite polynomials are chosen as the expansion basis rather than any other orthogonal function is that the expansion coefficients of the projection correspond exactly to the velocity moment (macroscopic flow variables) up to the chosen degree [105]. In brief, the entire procedure involves the following three broad steps:

1. Projection of the distribution function and equilibrium distribution function onto a Hilbert sub-space via their expansion as orthogonal Hermite polynomials in the velocity space (Sec. 2.2.1);
2. Truncation of the distributions up to a arbitrarily chosen degree, N, which depends on the velocity moments to be accurately captured for the flow physics under consideration (Sec. 2.3.1 and Chap. 3); and
3. Application of a Gauss-Hermite quadrature for the truncated distributions, yielding the discrete velocity set (lattice), followed by the space-time discretization (Chap. 3).

The necessary properties of Hermite polynomials required for the following presentation is compiled are Appx. A.

2.2.1 Projection on Hermite bases

The distribution function expanded on the dimensionless Hermite polynomial bases reads:

$$f(\boldsymbol{x}, \boldsymbol{\xi}, t) = \omega(\boldsymbol{\xi}) \sum_{i=0}^{\infty} \frac{1}{n!} \boldsymbol{a}^{(n)}(\boldsymbol{x}, t) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}), \tag{2.18}$$

where $\omega(\boldsymbol{\xi})$ is a weight function, and $\boldsymbol{a}^{(n)}$ and $\boldsymbol{H}^{(n)}$ are the expansion coefficients and Hermite polynomials, respectively [45, 65]. Both are tensors of rank n. The weight function is:

$$\omega(\boldsymbol{\xi}) = \frac{1}{(2\pi)^{D/2}} \exp\big[-\frac{\xi^2}{2}\big] \tag{2.19}$$

The dimensionless expansion coefficients $\boldsymbol{a}^{(n)}(\boldsymbol{x}, t)$ are associated with hydrodynamic quantities, which are defined below:

$$\boldsymbol{a}^{(n)}(\boldsymbol{x}, t) = \int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\boldsymbol{H}^{(n)}(\boldsymbol{\xi})d\boldsymbol{\xi} \tag{2.20}$$

where the first five expansion coefficient tensors are explicitly given as:

$$\begin{aligned} \text{Density}: \ \boldsymbol{a}^{(0)} = &\ \textstyle\int f d\boldsymbol{\xi} &= \rho, \\ \text{Momentum}: \ \boldsymbol{a}^{(1)} = &\ \textstyle\int \boldsymbol{\xi} f d\boldsymbol{\xi} &= \rho\boldsymbol{u}, \end{aligned}$$

$$\begin{aligned} \text{Energy}: \ \boldsymbol{a}^{(2)} &= \int f(\boldsymbol{\xi}^2 - \boldsymbol{\delta})d\boldsymbol{\xi} = \boldsymbol{P} + \rho(\boldsymbol{u}^2 - \boldsymbol{\delta}), \\ \text{Energy Flux}: \ \boldsymbol{a}^{(3)} &= \int f(\boldsymbol{\xi}^3 - \boldsymbol{\xi})d\boldsymbol{\xi} = \boldsymbol{Q} + \boldsymbol{u}\boldsymbol{a}^{(2)} - (D-1)\rho\boldsymbol{uuu}, \\ \boldsymbol{a}^{(4)} &= \boldsymbol{R} + 4\boldsymbol{uQ} + 6\boldsymbol{uuP} - 6\boldsymbol{P\delta} - 6\rho\boldsymbol{uu} + 3\rho, \end{aligned} \tag{2.21}$$

and $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the third and fourth moments, given by:

$$\boldsymbol{Q} = \int \boldsymbol{ccc} f d\boldsymbol{c}, \qquad \boldsymbol{R} = \int \boldsymbol{cccc} f d\boldsymbol{c} \tag{2.22}$$

The energy flux tensor is related to heat flux by contraction:

$$q_\alpha = Q_{\alpha\beta\beta} \tag{2.23}$$

Here, D is the space dimension, and $\boldsymbol{\delta}$ is the Kronecker delta function, also sometimes termed as identity matrix $\boldsymbol{I}$, in n-dimensions depending on the rank of the tensor $\boldsymbol{a}^{(n)}$, i.e. $(I_n)_{ij} = \delta_{ij}$. Some of the tensor notations followed here are the same as those adopted by Shan *et al.* [105] and Grad [44]. For example, the product of two tensors, such that $\boldsymbol{ua}^{(2)}$ means the sum of all possible permutations of the tensor product, i.e. $\boldsymbol{ua}^{(2)} = u_i a^{(2)}_{jk} + u_i a^{(2)}_{jk} + u_i a^{(2)}_{jk}$.

However, in departure with [105], where $\boldsymbol{uu}$ is denoted as $\boldsymbol{u}^2$ and $\boldsymbol{uuu} \equiv \boldsymbol{u}^3$, etc., we follow the full notation, or $\boldsymbol{uu}$, $\boldsymbol{uuu}$, for clarity [2].

By rearranging Eq. (2.21), we can observe that the five fundamental thermohydrodynamic variables, ρ, $\boldsymbol{u}$ and θ, and the momentum flux tensor $\boldsymbol{P}$ (or its traceless part, the stress tensor, $\boldsymbol{\sigma}$) are *completely* determined by the first three Hermite expansion coefficients ($\boldsymbol{a}^{(0)}$, $\boldsymbol{a}^{(1)}$, $\boldsymbol{a}^{(2)}$) *alone*, whereas the third moment, i.e. the heat flux $\boldsymbol{q}$, is completely determined by the fourth coefficient $\boldsymbol{a}^{(3)}$.

In a similar fashion, we can project the equilibrium distribution function (Eq. (2.15)) to get:

$$f^{(0)}(\boldsymbol{x}, \boldsymbol{\xi}, t) = \omega(\boldsymbol{\xi}) \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}_0^{(n)}(\boldsymbol{x}, t) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}) \tag{2.24}$$

where the first five equilibrium expansion coefficient tensor, $\boldsymbol{a}_0^{(n)}$, are:

$$\begin{aligned}
\boldsymbol{a}_0^{(0)} &= \rho \\
\boldsymbol{a}_0^{(1)} &= \rho \boldsymbol{u} \\
\boldsymbol{a}_0^{(2)} &= \rho \boldsymbol{uu} + \rho(\theta - 1)\boldsymbol{\delta} \\
\boldsymbol{a}_0^{(3)} &= \rho \boldsymbol{uuu} + \rho \boldsymbol{u}(\theta - 1)\boldsymbol{\delta} \\
\boldsymbol{a}_0^{(4)} &= \rho \boldsymbol{uuuu} + \rho \boldsymbol{uu}(\theta - 1)\boldsymbol{\delta} + \rho(\theta - 1)^2 \boldsymbol{\delta}^2
\end{aligned} \tag{2.25}$$

With the moments defined as above, we can now project the Boltzmann-BGK equation on the Hermite polynomial bases. To repeat, the Boltzmann-BGK equation is:

$$\left(\frac{\partial}{\partial t} + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} + \boldsymbol{g} \cdot \nabla_{\boldsymbol{\xi}} \right) f(\boldsymbol{x}, \boldsymbol{\xi}, t) = -\frac{1}{\tau}(f - f^0), \tag{2.26}$$

Therefore, the projection of Eq. (2.26) on the Hermite-polynomials yields:

$$\int \left(\frac{\partial}{\partial t} + (\boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} + \mathbf{g} \cdot \nabla_{\boldsymbol{\xi}} \right) f(\boldsymbol{x}, \boldsymbol{\xi}, t) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}) = -\int \frac{1}{\tau}(f - f^0) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}) d\boldsymbol{\xi} \tag{2.27}$$

[2]· $\boldsymbol{u}$ is a vector and $\boldsymbol{a}^{(2)}$ is second-order tensor. So, for example in 2D, where we have $\{i, j, k\} = \{x, y\}$, $\boldsymbol{ua}^{(2)}$ can be expanded as $\boldsymbol{ua}^{(2)} = a_{xx}(u_x + u_y) + 2a_{xy}(u_x + u_y) + a_{yy}(u_x + u_y)$.
· Note that $\boldsymbol{u}^2 \equiv \boldsymbol{uu}$ is a second-order tensor, $u = \sqrt{u_x^2 + u_y^2 + u_z^2}$ is the L-2 norm (magnitude) of $\boldsymbol{u}$, and u^2 is the power (second) of the norm of the vector $\boldsymbol{u}$.
· Also, the fourth-order tensor $\boldsymbol{\delta}^2$ is defined by: $\boldsymbol{\delta}^2 \equiv \boldsymbol{\delta\delta} = \delta_{ij}\delta_{kl} + \delta_{ik}\delta_{jl} + \delta_{il}\delta_{jl}$. See [44], [35] and Appx. A for more details.

For further analysis, consider the left hand side:

$$\begin{aligned} &\left(\frac{\partial}{\partial t} + \boldsymbol{\xi}\cdot\nabla_{\boldsymbol{x}} + \mathbf{g}\cdot\nabla_{\boldsymbol{\xi}}\right) f\boldsymbol{H}^{(n)}(\boldsymbol{\xi}) \\ = \frac{\partial}{\partial t}\left(\int \underbrace{f\boldsymbol{H}^{(n)}(\boldsymbol{\xi})d\boldsymbol{\xi}}_{Eqn.\,2.20}\right) &+ \int \nabla_{\boldsymbol{x}}\cdot\left(\underbrace{\boldsymbol{\xi}\boldsymbol{H}^{(n)}(\boldsymbol{\xi})d\boldsymbol{\xi}}_{Appx.\,A}\right) + \int \underbrace{\nabla_{\boldsymbol{\xi}}\cdot\big(\boldsymbol{g}(\boldsymbol{x},t)f\big)\boldsymbol{H}^{(n)}(\boldsymbol{\xi})d\boldsymbol{\xi}}_{Green's\,identities} \\ &= \frac{\partial(\boldsymbol{a}^n)}{\partial t} + \nabla_{\boldsymbol{x}}\cdot\left(\int\big(\boldsymbol{H}^{n+1}(\boldsymbol{\xi})f + n\boldsymbol{H}^{(n-1)}(\boldsymbol{\xi})\boldsymbol{\delta}f\big)d\boldsymbol{\xi}\right) \\ &+ \int\big(\underbrace{\boldsymbol{g}(\boldsymbol{x},t)f\big)\boldsymbol{H}^{(n)}(\boldsymbol{\xi})\cdot\boldsymbol{n}d\psi}_{=0} - \int\underbrace{\big(\boldsymbol{g}(\boldsymbol{x},t)f\big)\cdot\nabla_{\boldsymbol{\xi}}\boldsymbol{H}^n(\boldsymbol{\xi})d\boldsymbol{\xi}}_{Appx.\,A}\end{aligned}$$

We have used the Green's theorem, and relations and properties of Hermite polynomials from Appx. A, to obtain simplifications for the left hand side.

The right hand side of Eq. (2.27) yields:

$$\int \underbrace{-\frac{1}{\tau}(f - f^{(0)})\boldsymbol{H}^{(n)}(\boldsymbol{\xi})d\boldsymbol{\xi}}_{Eqn.\,2.20} = -\frac{1}{\tau}\left(\boldsymbol{a}^{(n)} - \boldsymbol{a}_0^{(n)}\right) \tag{2.28}$$

Combining the left and right hand sides, we obtain the Hermite-projected Boltzmann-BGK:

$$\boxed{\frac{\partial\boldsymbol{a}^{(n)}}{\partial t} + \nabla_{\boldsymbol{x}}\cdot(\boldsymbol{a}^{(n+1)}) + n\nabla_{\boldsymbol{x}}\cdot(\boldsymbol{a}^{(n-1)}\boldsymbol{\delta}) - n\boldsymbol{g}\cdot(a^{n-1}\boldsymbol{\delta}) = -\frac{1}{\tau}\left(\boldsymbol{a}^{(n)} - \boldsymbol{a}_0^{(n)}\right)} \tag{2.29}$$

2.3 CHAPMAN-ENSKOG ACCURACY ESTIMATE

Up to this point the procedure above is, in essence, similar to the one outlined by Grad. However, as mentioned before, instead of using the 13-moment equation for closing the equation, we want general accuracy estimates for capturing physical effects in terms of the *deviation of the distributions from equilibrium* [105]. By these estimates, we can still remain in the kinetic space with f as the primary variable, instead of ρ, u_α, θ, $P_{\alpha\beta}$, and q_α, as in the 13-moment equations. Using Chapman-Enskog expansion to analyze the effect of the loss of information due to the truncation at a certain order, The estimation is primarily done through procedure of the Chapman-Enskog expansion. The Chapman-Enskog expansion can be interpreted as an asymptotic expansion in the power of an expansion parameter, ϵ, which provides a quantitative measure for the deviation of the distribution from its local equilibrium, $f^{(0)}$:

$$f = f^{(0)} + \epsilon f^{(1)} + \epsilon^2 f^{(2)} + \dots \tag{2.30}$$

The expansion parameter is of the same order of magnitude as the Knudsen number, $\epsilon \sim Kn$. The expansion can be viewed as an infinite sum of terms which mimics the shape of the real function. The main idea behind truncation is that, by taking a leading part of it, is possible to find an estimate of the function treating the residual terms as negligible.

For macroscale hydrodynamic flows, where time-scales are slow (compared to τ), and spatial variations are large (compared to the mean free path), Kn is very small. Typical values are $\tau \approx 10^{-11}$ sec and $\lambda \approx 10^{-7}$ cm for H_2. Hydrodynamic flows, governed by Euler or the Navier-Stokes equations, are therefore valid only in the limit of small Kn. Consequently, a Chapman-Enskog expansion procedure of the Boltzmann-BGK equation results in a *hierarchy of equations,* corresponding to different levels of expansion in Eq. (2.30). That is, a zeroth order approximation in the expansion parameter yields the Euler equation, the first order approximation yields the Navier-Stokes, and so on. In the following section, we use this principle on the projected Boltzmann-BGK equation (Eq. (2.29), instead of the original equation, in order to obtain Euler and Navier-Stokes level approximations.

2.3.1 Zeroth-Order Approximation

In an equilibrium state, the distribution function is given as Maxwell-Boltzmann distribution, Eq. (2.15). Therefore, by setting $f = f^{(0)}$, it can be shown the projected Boltzmann-BGK equation (Eq. (2.29)) leads to the Euler equations for fluid flow. Towards that end, we repeat the same procedure as detailed in Sec. 2.2.1, except that now we substitute $f^{(0)}$ instead of the f, which results in an equation similar to Eq. (2.29):

$$\frac{\partial \boldsymbol{a}_0^{(n)}}{\partial t} + \nabla_{\boldsymbol{x}} \cdot (\boldsymbol{a}_0^{(n+1)}) + n\nabla_{\boldsymbol{x}} \cdot (\boldsymbol{a}_0^{(n-1)}\boldsymbol{\delta}) - n\boldsymbol{g} \cdot (\boldsymbol{a}_0^{n-1}\boldsymbol{\delta}) = -\frac{1}{\tau}\left(\boldsymbol{a}_0^{(n)} - \boldsymbol{a}_0^{(n)}\right) \tag{2.31}$$

Using the definitions of equilibrium expansion tensors (Eq. (2.25)), we can write the explicit equations for $n = 0, 1, 2$, which are:

$$\begin{aligned}
n = 0 \Rightarrow \qquad & \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0 \\
n = 1 \Rightarrow \qquad & \frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) + \nabla \cdot) \rho \theta \boldsymbol{\delta} = \rho \boldsymbol{g} \\
n = 2 \Rightarrow \qquad & \frac{\partial \rho \epsilon}{\partial t} - \boldsymbol{u} \cdot \nabla (\rho \theta) + \frac{D+2}{2} \nabla \cdot (\rho \boldsymbol{u} \theta) = 0
\end{aligned} \tag{2.32}$$

where the equation corresponding to $n = 2$ is obtained by taking a term-by-term trace of Eq. (2.31) and the subsequent division by two. Eq. (2.32) is recognized as the Euler equation for fluid flow. Therefore, we can conclude that the kinetic equation for Hermite-projected $f^{(0)}$ results in the Euler equations. Also, notice the absence of transport coefficients, which implies that the viscosity and thermal conductivity are both zero for this ideal, imaginary fluid. Most fluids, however, have non-zero transport coefficients, which from a kinetic point of view, corresponds to the non-equilibrium state, i.e. $f \neq f^{(0)}$. In the following section, we will derive the compressible Navier-Stokes equations, using approximations for this non-equilibrium state, via the Chapman-Enskog expansion.

2.3.2 First-Order Approximation

A gas is not in equilibrium when the distribution is different from the Maxwell-Boltzmann distribution, $f \neq f^{(0)}$ [61]. This might arise due to non-uniform density, velocity or temperature. Since we are interested in continuum flows, for which $Kn \ll 1$ (c.f. Fig. 1 and Eq. (2.30)), we will assume $f^{(1)} \ll f^{(0)}$, where $f^{(1)} = \mathcal{O}(Kn)$. Consequently, the full distribution can be reduced to just two parts, an equilibrium part $f^{(0)}$ as given by Equation 2.15, and an off-equilibrium or non-equilibrium part, $f^{(1)}$, such that:

$$f = f^{(0)} + f^{(1)} \tag{2.33}$$

In light of Eq. (2.33), the Boltzmann-BGK equation is:

$$\left(\frac{\partial}{\partial t} + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} + \mathbf{g} \cdot \nabla_{\boldsymbol{\xi}} \right) (f^{(0)} + f^{(1)}) \approx -\frac{1}{\tau} f^{(1)}(\boldsymbol{x}, \boldsymbol{\xi}, t), \tag{2.34}$$

Further, with $Kn \ll 1$, the above Equation can be approximated by neglecting the $f^{(1)}$ on the left hand side:

$$\left(\frac{\partial}{\partial t} + \boldsymbol{\xi} \cdot \nabla_{\boldsymbol{x}} + \mathbf{g} \cdot \nabla_{\boldsymbol{\xi}}\right) f^{(0)} \approx -\frac{1}{\tau} f^{(1)}(\boldsymbol{x}, \boldsymbol{\xi}, t), \tag{2.35}$$

Now, using a similar procedure as detailed in Sec. 2.2.1, we project Eq. (2.35) on to the Hermite bases, following which we get an equation for the evolution of the expansion coefficients, similar to (2.29):

$$\frac{\partial}{\partial t}\left(\boldsymbol{a}_0^{(n)}\right) + \nabla_{\boldsymbol{x}} \cdot \left(\boldsymbol{a}_0^{(n+1)}\right) + n\nabla_{\boldsymbol{x}} \cdot \left(\boldsymbol{a}_0^{(n-1)}\boldsymbol{\delta}\right) - n\boldsymbol{g} \cdot \left(\boldsymbol{a}_0^{(n-1)}\boldsymbol{\delta}\right) \approx -\frac{1}{\tau}\boldsymbol{a}_1^{(n)} \tag{2.36}$$

where the formal definitions of $f^{(1)}$, and the non-equilibrium expansion coefficient tensor, $\boldsymbol{a}_1^{(n)}$, are given as:

$$f^{(1)}(\boldsymbol{x}, \boldsymbol{\xi}, t) = \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{H}^{(n)}(\boldsymbol{\xi}) \boldsymbol{a}_1^{(n)}(\boldsymbol{x}, t) \tag{2.37}$$

$$\boldsymbol{a}_1^{(n)}(\boldsymbol{x}, t) = \int f^{(1)}(\boldsymbol{x}, \boldsymbol{\xi}, t) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}) \tag{2.38}$$

It is evident from Eq. (2.37) that to obtain $f^{(1)}$, we need the non-equilibrium coefficient $\boldsymbol{a}_1^{(n)}$. Towards that end, Eq. (2.36), which relates the equilibrium and non-equilibrium coefficients, can be used.

2.3.3 Compressible Navier-Stokes

From Eq. (2.25), the first five equilibrium coefficients, $(\boldsymbol{a}_0^{(0)} \ldots \boldsymbol{a}_0^{(4)})$, imply that the equilibrium coefficients up to fourth order depend on the conserved quantities: density ρ, momentum $\boldsymbol{j} \equiv \rho\boldsymbol{u}$, and energy $E \equiv \rho\epsilon$, but *not* explicitly on time. For the first term on the left hand side of Eq. (2.36), which involves the temporal derivative, we then can apply the chain rule as:

$$\frac{\partial}{\partial t}\left(\boldsymbol{a}_0^{(n)}\right) = \frac{\partial}{\partial \rho}\left(\boldsymbol{a}_0^{(n)}\right)\underbrace{\frac{\partial \rho}{\partial t}}_{Eqn.\,2.32} + \frac{\partial}{\partial \boldsymbol{j}}\left(\boldsymbol{a}_0^{(n)}\right)\cdot\underbrace{\frac{\partial(\rho\boldsymbol{u})}{\partial t}}_{Eqn.\,2.32} + \frac{\partial}{\partial E}\left(\boldsymbol{a}_0^{(n)}\right)\underbrace{\frac{\partial(\rho\epsilon)}{\partial t}}_{Eqn.\,2.32} \tag{2.39}$$

which can be simplified term-by-term using the Euler's equation for mass, momentum, and energy, respectively as:

$$\begin{aligned}\frac{\partial}{\partial t}\left(\boldsymbol{a}_0^{(n)}\right) &= -\frac{\partial}{\partial \rho}\left(\boldsymbol{a}_0^{(n)}\right)\nabla\cdot(\rho\boldsymbol{u}) + \frac{\partial}{\partial \boldsymbol{j}}\left(\boldsymbol{a}_0^{(n)}\right)\cdot\left(\rho\boldsymbol{g} - \nabla\cdot(\rho\boldsymbol{u}\boldsymbol{u}) - \frac{2}{D}\nabla\cdot(\rho\epsilon\delta)\right) \\ &+ \frac{\partial}{\partial E}\left(\boldsymbol{a}_0^{(n)}\right)\left(\frac{D+2}{2}\rho g\cdot\boldsymbol{u} + \frac{2}{D}u\cdot\nabla(\rho\epsilon) - \frac{D+2}{D}\nabla\cdot(\rho\epsilon\boldsymbol{u})\right).\end{aligned} \tag{2.40}$$

From the definition of the equilibrium coefficients (Eq. (2.25)), we can obtain expressions for the partial derivatives of the first four $\boldsymbol{a}_0^{(n)}$, with respect to density, momentum and energy, i.e. for $n = 0, 1, 2, 3$, we have [84]:

$$\frac{\partial \boldsymbol{a}_0^{(0)}}{\partial \rho} = 1,\quad \frac{\partial \boldsymbol{a}_0^{(0)}}{\partial \boldsymbol{j}} = 0,\quad \frac{\partial \boldsymbol{a}_0^{(0)}}{\partial E} = 1$$

$$\frac{\partial \boldsymbol{a}_0^{(1)}}{\partial \rho} = 0,\quad \frac{\partial \boldsymbol{a}_0^{(1)}}{\partial \boldsymbol{j}} = 1,\quad \frac{\partial \boldsymbol{a}_0^{(1)}}{\partial E} = 0$$

$$\frac{\partial \boldsymbol{a}_0^{(2)}}{\partial \rho} = -\boldsymbol{u}\boldsymbol{u} - \boldsymbol{\delta},\quad \frac{\partial \boldsymbol{a}_0^{(2)}}{\partial \boldsymbol{j}} = 2\boldsymbol{u}\boldsymbol{\delta},\quad \frac{\partial \boldsymbol{a}_0^{(2)}}{\partial E} = \frac{2}{D}\boldsymbol{\delta} \tag{2.41}$$

$$\frac{\partial \boldsymbol{a}_0^{(3)}}{\partial \rho} = -2\boldsymbol{u}\boldsymbol{u}\boldsymbol{u} - 3\theta\boldsymbol{u}\boldsymbol{\delta},\quad \frac{\partial \boldsymbol{a}_0^{(3)}}{\partial \boldsymbol{j}} = 3\boldsymbol{u}\boldsymbol{u}\boldsymbol{\delta} + 3(\theta-1)\boldsymbol{\delta},\quad \frac{\partial \boldsymbol{a}_0^{(3)}}{\partial E} = 3\frac{2}{D}\boldsymbol{u}\boldsymbol{\delta}$$

Combining Eqs. (2.36), (2.41), and (2.25), we obtain the explicit expression for first *four non-equilibrium distribution coefficient tensors* as:

$$\begin{aligned}
\boldsymbol{a}_1^{(0)} &= 0, \\
\boldsymbol{a}_1^{(1)} &= 0, \\
\boldsymbol{a}_1^{(2)} &= \tau\rho\theta\Big(2\nabla\boldsymbol{u} - \frac{2}{D}(\nabla\cdot\boldsymbol{u})\boldsymbol{\delta}\Big), \\
\boldsymbol{a}_1^{(3)} &= \tau\rho\theta\Big(3\boldsymbol{u}\big(2\nabla\boldsymbol{u} - \frac{2}{D}(\nabla\cdot\boldsymbol{u})\boldsymbol{\delta}\big) + 3\nabla\theta\boldsymbol{\delta}\Big)
\end{aligned} \tag{2.42}$$

In explicit component notation, $\boldsymbol{a}_1^{(2)}$ and $\boldsymbol{a}_1^{(3)}$ are:

$$a^{(2)}_{1_{\alpha\beta}} = -\tau\rho\theta\Gamma_{\alpha\beta} \tag{2.43}$$

$$a^{(3)}_{1_{\alpha\beta}} = -\tau\rho\theta\Bigg(\Gamma_{\alpha\beta}u_\gamma + \Gamma_{\alpha\gamma}u_\beta + \Gamma_{\beta\gamma}u_\alpha + \big(\delta_{\alpha\beta}\frac{\partial\theta}{\partial x_\gamma} + \delta_{\alpha\gamma}\frac{\partial\theta}{\partial x_\beta} + \delta_{\beta\gamma}\frac{\partial\theta}{\partial x_\alpha}\big)\Bigg) \tag{2.44}$$

where Γ is a tensor similar to the strain-rate tensor [66, 124]:

$$\boldsymbol{\Gamma} \equiv \Gamma_{\alpha\beta} = \frac{\partial u_\beta}{\partial x_\alpha} + \frac{\partial u_\alpha}{\partial x_\beta} - \frac{2}{D}\frac{\partial u_\gamma\delta_{\alpha\beta}}{\partial x_\gamma} \tag{2.45}$$

With the knowledge of the non-equilibrium coefficients (Eq. (2.42)), we can therefore reconstruct the off-equilibrium distribution function:

$$\begin{aligned}
f^{(1)} &= \sum_{n=0}^{3}\frac{1}{n!}\boldsymbol{H}^{(n)}(\boldsymbol{\xi})\boldsymbol{a}_1^{(n)}(\boldsymbol{x},t) \\
&= \Big(\frac{1}{2}\boldsymbol{\mathcal{H}}^{(2)}\boldsymbol{a}_1^{(2)} + \frac{1}{6}\boldsymbol{\mathcal{H}}^{(3)}\boldsymbol{a}_1^{(3)}\Big)
\end{aligned} \tag{2.46}$$

Finally, in order to close the conservations laws (Eq. (2.17)), we need expressions for $\boldsymbol{P}$ and $\boldsymbol{q}$. Given the knowledge of equilibrium and non-equilibrium coefficients, we can develop approximations in terms of the conserved quantities:

$$\begin{aligned}
\boldsymbol{P} &= \int \boldsymbol{cc}f d\boldsymbol{\xi} = \int(\boldsymbol{\xi}-\boldsymbol{u})(\boldsymbol{\xi}-\boldsymbol{u})\left(f^{(0)}+f^{(1)}\right) \\
&= \underbrace{\rho T\boldsymbol{\delta}}_{Eqn.\,2.25} + \int(\boldsymbol{\xi\xi} - \boldsymbol{\xi u} - \boldsymbol{u\xi} + \boldsymbol{uu})f^{(1)}d\boldsymbol{\xi} \\
&= \rho T\boldsymbol{\delta} - \underbrace{\tau\rho\theta\boldsymbol{\Gamma}}_{Eqn.\,2.46}
\end{aligned} \tag{2.47}$$

and, by the definition of heat flux, we have:

$$\begin{aligned}
\boldsymbol{q} &= \frac{1}{2}\int \boldsymbol{c}\boldsymbol{c}^2 f d\boldsymbol{c} = \frac{1}{2}\int (\boldsymbol{\xi}-\boldsymbol{u})(\boldsymbol{\xi}-\boldsymbol{u})^2 \left(f^{(0)}+f^{(1)}\right) d\boldsymbol{\xi} \\
&= \underbrace{0}_{Eqn.\,2.25} + \frac{1}{2}\int (\boldsymbol{\xi}-\boldsymbol{u})(\boldsymbol{\xi}-\boldsymbol{u})^2 f^{(1)} d\boldsymbol{\xi} \qquad (2.48) \\
&= \underbrace{-\tau\rho\theta\left(\frac{D+2}{D}\right)\nabla\theta}_{Eqn.\,2.46}.
\end{aligned}$$

Finally, substituting the expressions for $\boldsymbol{P}$ and $\boldsymbol{q}$ in the conservation laws, and applying some algebraic manipulation, we have:

$$\begin{aligned}
\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}) &= 0, \\
\frac{\partial(\rho\boldsymbol{u})}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}\boldsymbol{u} + p\boldsymbol{\delta} - \mu\boldsymbol{\Gamma}) &= \rho\boldsymbol{g}, \qquad (2.49) \\
\rho\frac{\mathrm{d}\epsilon}{\mathrm{d}t} + (p\boldsymbol{\delta} - \mu\boldsymbol{\Gamma}):(\nabla\boldsymbol{u}) - \frac{D+2}{2}\nabla\cdot(k\nabla\theta) &= 0
\end{aligned}$$

The above equation can be recognized as the *compressible Navier–Stokes* equations, with the following transport coefficients:

$$\begin{aligned}
p &= \rho\theta \\
\mu = k &= \rho\theta\tau \qquad (2.50)
\end{aligned}$$

This rigorous, but straightforward, derivation shows that the Hermite-basis projected Boltzmann BGK equation, in the limit of small Kn, leads to the compressible Navier–Stokes equations. In the equations derived above, the dynamic viscosity and the thermal conductivity are equal (Eq. (2.50)). This results from the BGK collision model, where it is assumed that all moments of the distribution relax at the same rate. A consequence of this, relevant especially for thermal flows, is that it predicts a Prandtl number of one rather than a value close to the 2/3 appropriate for monoatomic gas. A multiple-relaxation time model, developed in the Hermite-LB framework, has been proposed by Shan and Chen [103] which may overcome this constraint. Moreover, the bulk viscosity, λ, in Eq. (2.49) is also forced to be zero. The bulk viscosity does not affect a truly incompressible fluid, but it does affect certain phenomena occurring in nearly incompressible fluids, e.g., sound absorption in liquids. It also cannot be generally neglected for fully compressible flows, for e.g., sound wave absorption attenuation, and shock waves.

2.4 DISCUSSION

In summary, we have shown that, to obtain the non-equilibrium expansion coefficients up to the third order, only the fourth order and the lower orders of equilibrium coefficients required. That is, to obtain $\boldsymbol{a}_1^{(3)}$, we need $\boldsymbol{a}_0^{(4)}$, $\boldsymbol{a}_0^{(3)}$, $\boldsymbol{a}_0^{(2)}$, $\boldsymbol{a}_0^{(1)}$ and $\boldsymbol{a}_0^{(0)}$ [80]. In general, if n^{th} non-equilibrium coefficient, we need the $(n+1)^{th}$ equilibrium coefficient for Navier-Stokes level physics, and therefore the Maxwell-Boltzmann equilibrium distribution has to be truncated at the $(n+1)^{th}$ order. Physically this means that if the *heat flux, $\boldsymbol{q}$, and energy dynamics have to be recovered, as required by a thermal model, then the equilibrium distribution has to be expanded up to fourth order*, since $\boldsymbol{q}$ is the third moment.

On the other hand, if we are only interested in the density and momentum and its flux, rather than energy or higher moments, momentum flux, $\boldsymbol{P}$, is completely determined by $\boldsymbol{a}_1^{(2)}$, which in turn depends only on the equilibrium coefficients up to the third-order i.e. $\boldsymbol{a}_0^{(3)}$, $\boldsymbol{a}_0^{(2)}$, $\boldsymbol{a}_0^{(1)}$, $\boldsymbol{a}_0^{(0)}$, since $\boldsymbol{P}$ is the second-moment. This approximation leads to an isothermal model, i.e. $(\theta = 1)$, since energy dynamics is not being captured. Most importantly, this procedure can be generalized systematically, which shows up to which order the equilibrium distribution has to be expanded, in order to capture the right macroscopic equations. In fact, using this theoretical procedure, it is claimed that physics even beyond the Navier-Stokes level (Burnett-level) can be obtained [80, 105, 129].

2.5 SUMMARY

In this chapter, we presented the derivation of compressible Navier-Stokes equation from the continuous Boltzmann-BGK equation. We started by taking moments of the continuous BGK equation leading to the conservation equations of mass, momentum, and energy, but with an unknown pressure tensor and energy flux. In order to close the system (i.e., find constitutive equations for the pressure tensor and the energy flux), we followed the idea of Grad, by projecting the Boltzmann-BGK equation on the Hermite basis, and truncating the equilibrium distribution, *a priori,* at the fourth order. Then, using the Chapman–Enskog type accuracy estimation, we derived constitutive equations relating the pressure tensor to velocity gradients, and the heat flux to temperature gradients. Furthermore, we derived relations for the viscosity and thermal conductivity, in terms of the relaxation time.

The preceding equations are all still in continuous space. Obviously for computer implementation, the independent variables in the kinetic space, i.e., $\boldsymbol{\xi}$, $\boldsymbol{x}$, and t, have to be discretized. The key question concerning the discretization process is how accurately the macroscopic thermo-hydrodynamics of the continuum kinetic system can be reproduced by the much simplified (discretized) systems. That is, the discrete model should faithfully reproduce the continuous behavior, without any artifacts. The discretization of velocity and space-time, as required for the thermal model, is covered in the next chapter.

3.0 VELOCITY AND SPACE-TIME DISCRETIZATION

3.1 INTRODUCTION

In the previous chapter, we concluded that the compressible Navier-Stokes-like equations can be derived using a Hermite series expansion of the continuous Boltzmann-BGK equation, in a manner similar to the Grad 13-moment equation. Still, in order for us to use the kinetic (Boltzmann) equation to simulate flows of practical interest, the previously presented theoretical work has two important remaining tasks: (i) the choice of the order of truncation of the distributions; and (ii) discretization of the velocity ($\boldsymbol{\xi}$) and space-time ($\boldsymbol{x}, t$). A fully discrete kinetic, scheme suitable for fluid-flow computations, should have the following three essential components:

1. A *discrete equilibrium distribution* that is formulated to recover a specific thermohydrodynamic regime, such as the Euler, (isothermal) incompressible Navier-Stokes, (thermal) compressible Navier-Stokes equations. The discrete functional form of the equilibrium distribution depends on its order of truncation. In the previous chapter, we showed that by truncating the equilibrium distribution *a priori* at the 4^{th} order, we recover the compressible Navier-Stokes-like equation. The choice of 4^{th} order was deliberate so as to obtain a thermal model. In this chapter, we discuss some additional aspects of the truncation which are relevant for the velocity discretization. This is discussed in Sec. 3.2.

2. A *discrete velocity set* that represents the entire continuous velocity space, $\boldsymbol{\xi}$, by some discrete velocities $\boldsymbol{\xi}_i$, $i = 0, \ldots b$. The flow variables are (velocity) moments of the distributions, and hence, the macroscopic behavior of the kinetic equation must be preserved

by the discretization of the velocity space. The set of discrete velocities are termed as a lattice, a legacy of LGCA. In the discussion hereinafter, we shall use the terms lattice and discrete velocities indistinguishably. Velocity discretization is discussed in Sec. 3.3 and Appx. B.

3. *A evolution or kinetic equation* for the distributions: The evolution equation can be the widely used lattice Boltzmann equation (LBE), or other temporal schemes based on a finite difference method, finite-element method, volumetric formulations, etc. This is discussed in Sec. 3.4.

The three components listed above are collectively called a *lattice Boltzmann model*. In fact, all LB models have the above three components, although the particular form and details of each component may vary.

3.2 TRUNCATION OF THE HERMITE-EXPANDED DISTRIBUTIONS

In Sec. 2.3.3, we truncated the Hermite expansion of the distributions at the fourth-order and obtained the macroscopic behavior of the truncated Boltzmann-BGK equation. However, the choice of fourth-order was not elucidated. In this section, we provide the necessary background and justification behind that choice and generalization of the choice of the truncation order.

Consider $f^{(0,N)}(\boldsymbol{x},\boldsymbol{\xi},t)$ to be the truncated approximation of the full equilibrium distribution, $f^{(0)}(\boldsymbol{x},\boldsymbol{\xi},t)$, which is defined by:

$$f^{(0,N)}(\boldsymbol{x},\boldsymbol{\xi},t) = \omega(\boldsymbol{\xi}) \sum_{n=0}^{N} \frac{1}{n!} \boldsymbol{a}_0^{(n)}(\boldsymbol{x},t) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}), \tag{3.1}$$

where N is the order of truncation. Here, the question to which we seek an answer is: *at what order should the equilibrium distribution be truncated*? The answer lies in the Chapman-Enskog accuracy estimation procedure outlined in Chap. 2. For the Navier-Stokes level approximation, we observed that moments up to one order higher are needed of the equilibrium distribution, so that the transport of conserved moments is recovered. Specifically, we showed that, in order to recover the heat flux and energy dynamics, as required by a thermal model, the equilibrium distribution has to expanded up to 4^{th} order, since heat flux is the 3^{rd} moment. That is, *for a thermal LB model,* $N = 4$. On the other hand, if we are interested only in an isothermal case, to recover the required pressure tensor and momentum dynamics, the equilibrium distribution function has to be expanded only to the 3^{rd} order, since momentum is the 2^{nd} moment. Hence, for an accurate, *isothermal LB model,* $N = 3$. It is useful to point out that the most widely used LB model is based on only a second-order expansion of the equilibrium distribution ($N = 2$) around the local velocity (or Mach number) $\boldsymbol{u}^2$, instead of a third-order expansion, which results in a well-known error of $\mathcal{O}(\boldsymbol{u}^3)$ and non-Galilean invariant behavior [80, 90]. More on this in Sec. 3.3.2.

It is clear that once the order of truncation is chosen to be N, *the higher-order* ($f^{(N+1)}$) t*runcated terms should not affect the lower-order terms and their moments.* The mutual orthogonality of the Hermite polynomials is useful in this regard. Due to this property, the leading moments of a distribution function up to N^{th} order are preserved by truncations of the higher-order terms ($> N + 1$) in its Hermite expansion. Therefore, a distribution function can be approximated by its projection onto a Hilbert subspace spanned by the first N Hermite polynomials, *without changing the first N moments,* i.e., up to the N^{th} order, $f^{(N)}(\boldsymbol{x}, \boldsymbol{\xi}, t)$ has exactly the same moments as the full $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$:

$$f^{(0)}(\boldsymbol{x}, \boldsymbol{\xi}, t) \approx f^{(0,N)}(\boldsymbol{x}, \boldsymbol{\xi}, t) = \omega(\boldsymbol{\xi}) \sum_{n=0}^{N} \frac{1}{n!} \boldsymbol{a}^{(n)}(\boldsymbol{x}, t) \boldsymbol{H}^{(n)}(\boldsymbol{\xi}). \tag{3.2}$$

Now, given $f^{(0,N)}$, the next question is, *how do we obtain the discrete velocity moments* of the truncated distributions required for computing the hydrodynamic variables? The answer to this lies in the velocity-space discretization process outlined below. Of paramount importance in the discretization procedure is ensuring that the moments of $f^{(0,N)}$ in the discrete space are *exactly* the same as those in continuous space.

3.3 VELOCITY DISCRETIZATION

From the definition of expansion coefficients, we know that $\boldsymbol{a}_0^{(n)}$ does not depend on higher-order terms $\boldsymbol{a}_0^{(n+1)}, \boldsymbol{a}_0^{(n+2)} \dots$. Therefore, $\boldsymbol{a}_0^{(n)}$ in Eq. (3.2) can be evaluated exactly by $f^{(0,N)}$ for $n = 1 \dots N$, instead of the complete $f^{(0)}$, i.e.:

$$\boldsymbol{a}_0^{(n)}(\boldsymbol{x}, t) = \int f^{(0,N)}(\boldsymbol{x}, \boldsymbol{\xi}, t)\boldsymbol{H}^{(n)}(\boldsymbol{\xi})d\boldsymbol{\xi}. \tag{3.3}$$

Note that here $f^{(0,N)}(\boldsymbol{x}, \boldsymbol{\xi}, t)/\omega(\boldsymbol{\xi})$ and $\boldsymbol{H}^{(n)}(\boldsymbol{\xi})$ both are polynomials of the order no more than N. Therefore, we have a polynomial $p(\boldsymbol{\xi}, \boldsymbol{x}, t)$ in $\boldsymbol{\xi}$, of the order $2N$, such that:

$$\omega(\boldsymbol{\xi})p(\boldsymbol{\xi}, \boldsymbol{x}, t) = f^{(N)}(\boldsymbol{x}, \boldsymbol{\xi}, t)\boldsymbol{H}^{(n)}(\boldsymbol{\xi}). \tag{3.4}$$

Given this fact, $\boldsymbol{a}^{(n)}$ in Eq. (3.3) can be computed *exactly* using the Gauss-Hermite quadrature in multiple dimensions, and with precision of quadrature, $\mathcal{Q} \geq 2N$. That is:

$$\begin{aligned} \boldsymbol{a}_0^{(n)}(\boldsymbol{x}, t) &= \int \omega(\boldsymbol{\xi})p(\boldsymbol{x}, \boldsymbol{\xi}, t) = \sum_{i=0}^{q-1} \omega_i p(\boldsymbol{x}, \boldsymbol{\xi}_i, t) \\ &= \sum_{i=0}^{q-1} \frac{w_i}{\omega(\boldsymbol{\xi}_i)} f^{(0,N)}(\boldsymbol{x}, \boldsymbol{\xi}_i, t)\boldsymbol{H}^{(n)}(\boldsymbol{\xi}_i). \end{aligned} \tag{3.5}$$

$\mathcal{Q} \equiv \{(w_i, \boldsymbol{\xi}_i) : i = 0, \dots, q-1\}$ represents a Gauss-Hermite quadrature set with weights, w_i and abscissae $\boldsymbol{\xi}_i$. Therefore, the discrete distributions $\{f^{(N)}(\boldsymbol{x}, \boldsymbol{\xi}_i, t) : i = 0, ..., q-1\}$, *completely* determine $f^{(N)}$, and therefore its first N moments and vice versa. That is, $f^{(0,N)}$ has exactly the same velocity moments as the full $f^{(0)}(\boldsymbol{x}, \boldsymbol{\xi}, t)$ does. Appx. B lists the results of the Gauss-Hermite quadrature for $\mathcal{Q} = 5, 7$, and 9 for both two- and three dimensions.

3.3.1 On-lattice and Off-lattice Velocity Sets

Next, consider the following fully discretized (velocity and space-time), standard fully discretized form of the Boltzmann-BGK equation, termed as the lattice BGK equation:

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i, t + 1) - f_i(\boldsymbol{x}, t) = -\frac{1}{\tau}\left[f_i - f_i^{(0)}\right] \tag{3.6}$$

For this evolution equation to be used, the spatial grid should be invariant under the transform $\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{e}_i$. This means that, if $\boldsymbol{x}$ is a node on the grid, $\boldsymbol{x} + \boldsymbol{e}_i$ are also nodes on the grid. In the case of an extended neighborhood (more than one neighbor), also called higher-order lattices, $\boldsymbol{x} + 2\boldsymbol{e}_i$, $\boldsymbol{x} + 3\boldsymbol{e}_i$, etc., should also be nodes on the grid. For this to be realized, the higher-order speeds should be expressed as an integer multiple of the smallest, non-trivial constant (speed). Or, in other words, each Cartesian component of every discrete velocity must be an integer (in units of lattice spacing). Accordingly, a velocity-set that can be expressed as *integers* or *integer multiple* of a common (lattice) constant are called as *space-filling* or *on-(Cartesian) lattice velocity sets*. Otherwise, they are termed as *non-space-filling* or *off-lattice velocity sets*. Fig. 3 shows a pictorial view of this concept in one dimension. Besides being a defining feature of LBGK due to its particle analog (streaming and collision processes), Eq. (3.6) also has an important advantage of ensuring an *exact advection* (*zero diffusion)* of the distribution, regardless of the distance it travels. This compared to the fractional advection common in finite-difference or finite-volume schemes. The fact that exact advection also makes a simple and fully parallel algorithm is an added LBGK asset.

Coming back to the discretization process omitted in the presentation of Eq. (3.6), in the Hermite-based LB formulation, the *discrete velocities in* Eq. (3.6) are now *abscissas of a Gauss-Hermite quadrature, and not some predefined 'lattice' vectors.* However, as listed in Appx. B, the abscissas in many of the discrete velocity-sets cannot be expressed as integer multiples of a common constant c_l, therefore making them off-lattice. Since these off-lattice discrete velocities do not coincide with nodes of a regular lattice, they preclude the simple and exact discretization of LBGK. For example, the D2Q12 and D2Q16 lattices in two-dimensions, and D3Q27 in two-dimensions are off-lattice velocity sets. Consequently,

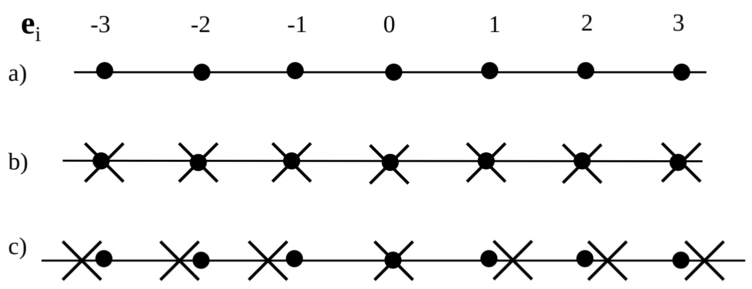


Figure 3: Schematic representation of space-filling and non-space-filling lattices in one dimension. (•) symbols represent nodes on a uniform Cartesian grid and × symbols represent discrete (non-dimensional) velocities. **a**. A uniform Cartesian grid (a lattice); **b.** A space-filling (on-lattice) velocity-set where all discrete velocities coincide with a lattice; **c.** A non-space-filling (off-lattice) velocity-set where discrete velocities *do not* coincide with a lattice.

Eq. (3.6) type evolution of distributions with off-lattice sets require additional numerical treatment, such as point-wise interpolation, which increases the computational cost, and also introduces extra numerical error.

To construct higher-order on-lattice discrete velocities, Shan *et al.* suggested a special quadrature technique in which abscissae are searched on the grid points of Cartesian coordinates, such that integer-valued discrete velocity values can be constructed [80, 103, 101]. The D2Q17, D2Q21, and D2Q37 velocity sets listed in Appx. B are examples of on-lattice sets. A similar method was employed by Phillipi *et al.* for thermal LB models, based on 'quadrature with prescribed abscissas' [86, 107, 108]. Some of the velocity sets proposed by them are the same as the one proposed by Shan *et al.*, for example D2Q17 and D2Q37. Furthermore, Chikatamarla *et al.* also developed higher-order integer-valued discrete velocity, such as D2Q16 and D2Q25 models, based on the entropic lattice Boltzmann formulation [32, 31].

Another important aspect of the higher-order velocity sets to be noted is their *extended neighborhood*. While in D2Q9, the distributions stream (advect) to their nearest neighbor, in the higher-order lattices, such as D2Q17 and D2Q37, the distributions have to stream to their next-nearest and next-next-nearest neighbors. This feature leads to a major difficulty in imposing boundary conditions through the simple bounce-back technique.

3.3.2 Galilean Invariance

Galilean invariance is a principle requires that the laws of physics (and hydrodynamics) are the same in all inertial frames of reference. In the LBGK context, obtaining Galilean invariance involves finding a discrete velocity set that makes *isotropic tensors*, up to a desired order. This is because, *the hydrodynamic variables (moments) are expressed as weighted sums of tensors constructed from discrete velocities, which are not always isotropic as their continuum counterparts*. Provided that the discrete moments agree with those of the continuous Maxwell distribution, the macroscopic hydrodynamics of the LBGK equation is the same as that of the continuum BGK equation, which is known to be fully Galilean invariant. The insufficient isotropy of certain moments has long been recognized as one of the reasons responsible for the deviations from the continuum hydrodynamics. For the Hermite-based LB formulation Nie, Shan and Chen [80, 26] showed that the sufficient conditions for discrete moments, and therefore, macroscopic hydrodynamics to be the same as the continuum system described by the Boltzmann-BGK system on a regular lattice, the following condition has to be met:

$$N \geq M \quad \text{and} \quad \mathcal{Q} \geq M + N \tag{3.7}$$

where M is the highest order of the moments that determines the hydrodynamic equations of interest, for example $M = 4$, if the energy equation at the Navier-Stokes level is sought, and $M = 3$ if the momentum equation is sought. N, as before, is the order of truncation of equilibrium distribution and $\mathcal{Q}$ is the order of precision of the quadrature.

Per rule (3.7), it is clear that, to obtain the fully Galilean invariant momentum equation (without any 'cubic' dependence of viscosity on velocity), $M = 3$, $N \geq 3$ and $\mathcal{Q} \geq 6$. The widely used D2Q9 belongs to the $\mathcal{Q} = 5$ quadrature rule, which results in the lack of Galilean invariance of certain moments. D2Q17 and D2Q21 are examples of the $\mathcal{Q} = 6$ rule. Similarly, to obtain the fully Galilean invariant energy equation (without a velocity-dependent thermal diffusivity), $M = 4$, $N \geq 4$ and $\mathcal{Q} \geq 8$. D2Q37 and D2Q121 are examples of the $\mathcal{Q} = 9$ quadrature rule.

It should be remarked that, usually, for the same degree of precision of the quadrature, the discrete velocities in a off-lattice velocity-set *are fewer* in number compared to those in a on-lattice set. For example, the off-lattice velocity set, D2Q16, has the same degree of precision as the on-lattice D2Q21 ($\mathcal{Q} = 7$), but obviously has fewer velocities. The disadvantage is that, unlike in D2Q21, the simple stream-collision type evolution cannot be used with D2Q16. The choice of a particular velocity set therefore involves a tradeoff between the (potential) computational savings obtained using D2Q16 versus the numerical-diffusion free results from D2Q21. Moreover, in general the difference in number of velocities in off- and on- lattices tends to increase as the precision of the quadrature increases. For off-lattice velocity sets, alternate time-evolution schemes for the distributions have to be used, which are called off-lattice Boltzmann schemes.

3.3.3 Rescaling

The derivations thus far, here and in Chap. 2, were based on the non-dimensional, lattice speed of sound, c_s, set to unity. The results of the quadrature presented in Appx. B are also based on $c_s = 1$. With this values of c_s, the results of the quadrature procedure reveals irrational numbers, for example, the roots in the D2Q9 quadrature sets are $\pm\sqrt{3}$. In the original formulation, and the results presented by Shan and co-workers elsewhere, c_s is maintained to be unity and the abscissas presented in their irrational form. However, the speed of sound, in fact, can be chosen arbitrarily per convenience.

Therefore, in order to avoid irrational number representation, we can rescale (normalize) the abscissae $\boldsymbol{\xi}_i$ with a different speed of sound, related to the respective lattice constant through $c_s \equiv 1/c_l$, such that:

$$\boldsymbol{e}_i = c_s \boldsymbol{\xi}_i \tag{3.8}$$

where $\boldsymbol{e}_i$ are *integer valued* discrete velocities (vectors), i.e., $\{\boldsymbol{e}_i\} = [-m, m]$, $m \in \mathbb{Z}$. Therefore, for D2Q9, the speed of sound is $c_s = 1/\sqrt{3}$, and for D2Q37 the speed of sound is $c_s = 1/1.1969$. Thus, c_s is nothing but a scaling factor used to obtain integer-valued discrete velocities from the abscissas. A rescaling of the abscissas leads to integer-valued discrete velocities suitable for the stream-collide scheme. However, rescaling of abscissas also requires a suitable rescaling of the expansion coefficients and equilibrium distribution function. Towards that end, the scaled equilibrium expansion coefficients are given by:

$$\begin{aligned}
\boldsymbol{a}_0^{(0)} &= \rho \\
\boldsymbol{a}_0^{(1)} &= \rho\boldsymbol{u} \\
\boldsymbol{a}_0^{(2)} &= \rho\boldsymbol{u}\boldsymbol{u} + c_s^2\rho(\theta-1)\boldsymbol{\delta} \\
\boldsymbol{a}_0^{(3)} &= \rho\boldsymbol{u}\boldsymbol{u}\boldsymbol{u} + 3c_s^2\rho(\theta-1)\boldsymbol{\delta}\boldsymbol{u} \\
\boldsymbol{a}_0^{(4)} &= \rho\boldsymbol{u}\boldsymbol{u}\boldsymbol{u}\boldsymbol{u} + 6c_s^2\rho(\theta-1)\boldsymbol{u}\boldsymbol{u}\boldsymbol{\delta} + 3c_s^4\rho(\theta-1)^2\boldsymbol{\delta}\boldsymbol{\delta}
\end{aligned} \tag{3.9}$$

For convenience, we can redefine $f_i(\boldsymbol{x},t) \equiv \frac{w_i}{\omega(\boldsymbol{\xi}_i)} f(\boldsymbol{x},\boldsymbol{\xi}_i,t)$ so that the rescaled N^{th}-order, truncated, equilibrium distribution function, $f_i^{(0,N)}$ can be written as:

$$f_i^{(0,N)} = w_i \sum_{n=0}^{4} \frac{1}{c_s^{2n} n!} \boldsymbol{a}_0^{(n)} \boldsymbol{H}^{(n)}(\boldsymbol{\xi}_i). \tag{3.10}$$

In explicit terms, the rescaled, 4^{th}-order truncated equilibrium distribution is:

$$\begin{aligned}
f_i^{(0,4)} = w_i\rho\Bigg(& 1 + \frac{\boldsymbol{\xi}_i\cdot\boldsymbol{u}}{c_s^2} + \frac{1}{2c_s^4}\left[(\boldsymbol{\xi}_i\cdot\boldsymbol{u})^2 - c_s^2\boldsymbol{u}^2 + c_s^2(\theta-1)(\boldsymbol{\xi}_i^2 - c_s^2 D)\right] \\
& + \frac{\boldsymbol{\xi}_i\cdot\boldsymbol{u}}{6c_s^6}\left[(\boldsymbol{\xi}_i\cdot\boldsymbol{u})^2 - 3c_s^2\boldsymbol{u}^2 + 3c_s^2(\theta-1)(\boldsymbol{\xi}_i^2 - c_s^2(D+2))\right] \\
& + \frac{1}{24c_s^8}\Bigg[(\boldsymbol{\xi}_i\cdot\boldsymbol{u})^4 - 6c_s^2\boldsymbol{u}(\boldsymbol{\xi}_i\cdot\boldsymbol{u})^2 + 3c_s^4\boldsymbol{u}^4 \\
& + 6c_s^2(\theta-1)\left((\boldsymbol{\xi}_i\cdot\boldsymbol{u})^2(\boldsymbol{\xi}_i^2 - c_s^2(D+4)) + c_s^2\boldsymbol{u}^2(c_s^2(D+2) - \boldsymbol{\xi}_i^2)\right) \\
& + 3c_s^4(\theta-1)^2\left(\boldsymbol{\xi}_i^4 - 2c_s^2(D+2)\boldsymbol{\xi}_i^2 + c_s^4 D(D+2)\right)\Bigg]\Bigg).
\end{aligned} \tag{3.11}$$

From Eq. (3.11), we observe that for $N = 2$, $\theta = 1$, (isothermal flow), and for the D2Q9 discrete velocity set ($c_l = \sqrt{3})$, we obtain the most widely used (second-order) functional form of the equilibrium distribution, which is given as:

$$f_i(\boldsymbol{x},t) = w_i\rho\left(1 + 3\boldsymbol{\xi}_i\cdot\boldsymbol{u} + \frac{9(\boldsymbol{\xi}_i\cdot\boldsymbol{u})^2}{2} - \frac{3\boldsymbol{u}\cdot\boldsymbol{u}}{2}\right) \tag{3.12}$$

Importantly, we can infer that this rigorously obtained result mimics the original somewhat ad hoc coefficient-matching procedure utilized in most LBM analysis.

Similarly, the fourth-order discretized force term is given as:

$$\begin{aligned}
\boldsymbol{F}_i &= \sum_{n=1}^{4-1} \frac{1}{n! c_s^{2n}} \boldsymbol{H}^{(n)} \boldsymbol{a}_0^{(n)} \\
&= w_i \rho \Bigg(\frac{\boldsymbol{\xi}_i \cdot \boldsymbol{g}}{c_s^2} + \frac{1}{2c_s^4} \Big[(\boldsymbol{\xi}_i \cdot \boldsymbol{u})(\boldsymbol{\xi}_i \cdot \boldsymbol{g}) - c_s^2 (\boldsymbol{g} \cdot \boldsymbol{u}) \Big] \\
&+ \frac{1}{6c_s^6} \Big[(\boldsymbol{\xi}_i \cdot \boldsymbol{g}) \big((\boldsymbol{\xi}_i \cdot \boldsymbol{u})^2 + c_s^2 \boldsymbol{\xi}_i^2 (\theta - 1) \big) \\
&- c_s^2 \Big[(\boldsymbol{\xi}_i \cdot \boldsymbol{u}) \big(2(\boldsymbol{g} \cdot \boldsymbol{u}) + \boldsymbol{u}^2 + c_s^2 D(\theta - 1) \big) + 2c_s^2 (\theta - 1)(\boldsymbol{\xi}_i \cdot \boldsymbol{g}) \Big] \Big] \Bigg).
\end{aligned} \tag{3.13}$$

It should be pointed out that, in actual simulations, the effects of a body force are seldom incorporated by computing $\boldsymbol{F}_i$ as written above. In fact, in the original formulation [105], the authors absorb the body force term $\boldsymbol{F}_i$ into the equilibrium distribution $f_i^{(0)}$ to form an effective equilibrium, such that $f_i^{eq} \equiv f_i^{(0)} + \tau F_i$. In addition to truncation of the above $\boldsymbol{F}_i$ to the first and second-order terms, velocity is also corrected (or shifted) in order to incorporate the effects of force. This method, just one of the many ways to incorporate forces in LB simulations, is called as the s*hifted-equilibrium* method. See [96, 97] for more details.

Finally, omitting the superscript N in the subsequent expressions, and incorporating $f_i^{eq} \equiv f_i^{(0)} + \tau F_i$, we now have the Boltzmann-BGK equation discretized in velocity space, for f^N :

$$\boxed{\frac{\partial f_i}{\partial t} + \boldsymbol{\xi}_i \cdot \nabla f_i = -\frac{1}{\tau}\big(f_i - f_i^{(eq)}\big)} \tag{3.14}$$

This discrete velocity form of the Boltzmann-BGK equation is called as the *discrete-velocity Boltzmann equation (DVBE)*. The different moments that we used to describe the hydrodynamic observables are no longer computed as integrals over an infinite velocity set, but as weighted sums over a finite number of discrete velocities:

$$\begin{aligned} \rho &= \sum_{i=0}^{q-1} f_i \\ \rho\boldsymbol{u} &= \sum_{i=0}^{q-1} f_i \boldsymbol{\xi}_i \\ \rho\epsilon &= \frac{1}{2}\sum_{i=0}^{q-1} f_i \boldsymbol{c}_i^2 \\ \boldsymbol{P} &= \sum_{i=0}^{q-1} f_i \boldsymbol{c}_i \boldsymbol{c}_i \\ \boldsymbol{q} &= \frac{1}{2}\sum_{i=0}^{q-1} f_i \boldsymbol{c}_i \boldsymbol{c}_i^2 \end{aligned} \tag{3.15}$$

where $\boldsymbol{c}_i(\boldsymbol{x}, t) = \boldsymbol{\xi}_i - \boldsymbol{u}(\boldsymbol{x}, t)$ is the discrete peculiar velocity. It must be noted that because of the rescaling of the Hermite basis we have the following expressions for the viscosity, thermal conductivity and the perfect gas law:

$$\begin{aligned} \mu &= c_s^2 \rho\theta\tau \\ k &= c_s^4 \rho\theta\tau \\ p &= c_s^2 \rho\theta \end{aligned} \tag{3.16}$$

3.4 SPACE-TIME DISCRETIZATION

Having discussed the velocity-discretization in detail, the next task in building a complete LB model is to discretize the configuration space (space-time). While there are several options for this, such as the volumetric formulation [25, 79], the finite-element formulation [70], etc., in this work we will focus on the finite-difference type of spatial discretization.

3.4.1 Method of Characteristics

The most widely used method of space-time discretization of the discrete velocity Boltzmann equation (Eq. (3.14)) is based on the method of characteristics. Essentially, for a first-order partial differential equation (PDE) like Eq. (3.14), the method of characteristics finds lines (called characteristic lines or characteristics) along which the PDE degenerates into an ordinary differential equation (ODE). Once these characteristics are found, the ODE can be solved along the characteristics in a much simpler way. To begin with, the Eq. (3.14) with forcing is:

$$\frac{\partial f_i}{\partial t} + \boldsymbol{\xi}_i \cdot \frac{\partial f_i}{\partial \boldsymbol{x}} = -\frac{1}{\tau}\big(f_i - f_i^{(0)}\big) + F_i \tag{3.17}$$

For clarity of presentation, we choose to use two notations: $f_i^{eq} = f_i^{(0)} + \tau F_i$ and $\Omega_i = -\frac{1}{\tau}(f_i - f_i^{(eq)})$. Therefore, Eq. (3.17) is:

$$\frac{\partial f_i}{\partial t} + \boldsymbol{\xi}_i \cdot \frac{\partial f_i}{\partial \boldsymbol{x}} = \Omega_i \tag{3.18}$$

First, we analyze the left hand side of Eq. (3.18). Towards that end, we apply the chain rule to the term $f_i\big(\boldsymbol{x}(s), t(s)\big)$, where s is a characteristic, and $\boldsymbol{x}(s), t(s)$, are still unknown. This leads to:

$$\frac{df_i\big(\boldsymbol{x}(s), t(s)\big)}{ds} = \frac{dt}{ds}\frac{\partial f_i}{\partial t} + \frac{d\boldsymbol{x}}{ds}\frac{\partial f_i}{\partial \boldsymbol{x}} \tag{3.19}$$

Comparing Eq. (3.19) with the left hand side of Eq. (3.18), we see that:

$$\begin{aligned} \frac{dt}{ds} &= 1 \\ \frac{dx}{ds} &= \boldsymbol{\xi}_i \end{aligned} \tag{3.20}$$

In view of Eq.(3.19), Eq. (3.18) can be written as:

$$\frac{df_i\big(\boldsymbol{x}(s), t(s)\big)}{ds} = \Omega_i. \tag{3.21}$$

It is evident from the form of Eq. (3.21) that the original PDE has now been converted into an ODE along the characteristics s. Integrating Eq. (3.20) we get:

$$\begin{aligned} t(s) &= t_0 + s - s_0 \\ \boldsymbol{x}(s) &= \boldsymbol{x}_0 + \boldsymbol{\xi}_i(s - s_0) \end{aligned} \tag{3.22}$$

Substituting Eq. (3.22) in Eq. (3.21), and integrating over the characteristics we get

$$\int_s \frac{df_i(\boldsymbol{x}_0 + \boldsymbol{\xi}_i(s - s_0), t_0 + s - s_0)}{ds} = \int_s \Omega_i \tag{3.23}$$

The left hand side of Eq. (3.23) is an integral of a differential with respect to the same variable s. The result is the argument of the integral itself, but evaluated at the boundaries of integration:

$$f_i(\boldsymbol{x}_0 + \boldsymbol{\xi}_i(s - s_0),\ t_0 + s - s_0) - f_i(\boldsymbol{x}_0, t) = \int_s \Omega_i. \tag{3.24}$$

From Eq. (3.22) we can set $s - s_0 = t - t_0 = \Delta t$, so that:

$$f_i(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t,\ t + \Delta t) - f(\boldsymbol{x}, t) = \int_t^{t+\Delta t} \Omega_i. \tag{3.25}$$

The evaluation of the collision integral can be generalized using the $\theta-$*method* used in the theory in ODE. That is, we can write:

$$\tilde{f}_i^{n+1} - \tilde{f}_i^n = \Delta t \Big[(1-\theta)\tilde{\Omega}_i^n + \theta \tilde{\Omega}_i^{n+1} \Big], \tag{3.26}$$

with $0 \leq \theta \leq 1$, and the superscript n denoting a time-level. The tilde indicates a quantity on the characteristic i.e. $\tilde{f}_i^{n+1} = f_i(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t, t_n + \Delta t)$ and $\tilde{f}_i^n = f_i(\boldsymbol{x}, t_n)$. For $\theta = \{0, \frac{1}{2}, 1\}$, we obtain an explicit $\mathcal{O}(\Delta t)$, an implicit $\mathcal{O}(\Delta t^2)$, and an implicit $\mathcal{O}(\Delta t)$ approximation of the collision term, respectively. The $\mathcal{O}(\Delta t)$ schemes are Euler-type schemes, and the $\mathcal{O}(\Delta t^2)$ is a Crank-Nicholson-type scheme.

3.4.2 Lattice Boltzmann Equation

The most straightforward method to evaluate the collision integral is to use an explicit Euler method where $\theta = 0$. This leads to:

$$f_i\big(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t, t + \Delta t\big) - f_i(\boldsymbol{x}, t) = -\frac{\Delta t}{\tau}\big(f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\big) \tag{3.27}$$

Eq. (3.27) can be identified as the *lattice-BGK equation* (LBGK) equation, or sometimes, just the lattice Boltzmann equation (LBE). It represents the simplest update-type scheme/rule possible for solving the Boltzmann equation. In practice, the computation consists of the following two steps:

1. **Collision or Relaxation Step:** At time instant t, a collision in $\boldsymbol{x}$ occurs. The collision term is evaluated to compute the new (temporary) distribution, f_i^c:

$$\text{Collision: } f_i^c(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\Big[f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\Big] \tag{3.28}$$

2. **Streaming Step:** Each collided distribution f_i^c is streamed along the characteristics depending upon its discrete velocity $\boldsymbol{\xi}_i$:

$$\text{Streaming: } f_i(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t, t + \Delta t) = f_i^c(\boldsymbol{x}, t) \tag{3.29}$$

The easiest way is to discretize the physical space, $\boldsymbol{x} \to \boldsymbol{x}_a$, is to have a regular lattice (uniform Cartesian mesh, in CFD terminology), such that for each $\boldsymbol{x}$, $\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t$ is also a node. This way, the information at all the grid nodes is automatically known at the next time step. Importantly, this means that the spatial step (grid-spacing) and temporal size are tightly coupled through the relation: $\Delta \boldsymbol{x} = \boldsymbol{\xi}_i \Delta t$. Furthermore, if we assume Δt and Δx as unity, as commonly done in LB simulations (lattice units), we get:

$$f_i\big(\boldsymbol{x} + \boldsymbol{\xi}_i, t + 1\big) = f_i(\boldsymbol{x}, t) - \frac{1}{\tau}\Big[f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\Big] \tag{3.30}$$

Eq. (3.30), along with the second-order form of the equilibrium distribution (Eq. (3.12)) and the D2Q9 discrete velocity set constitutes the most widely used lattice Boltzmann model. Traditionally, this equilibrium distribution was expanded as a Taylor series in terms of the local velocity, and truncated at the second order. Note that here $\tau = \mathcal{O}(1)$ and not $\mathcal{O}(Kn)$.

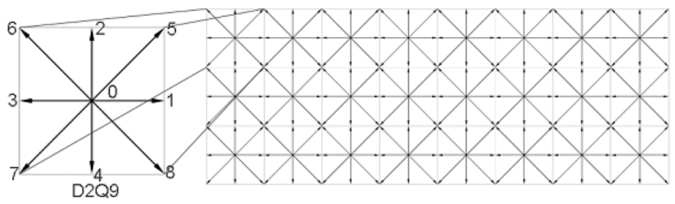


Figure 4: A uniform Cartesian grid (lattice) as used in LB simulations overlaid on the D2Q9 velocity set

This approach was derived from the lattice gas automata (LGA), the precursor to the LBM. The order of the of truncation was chosen *a posteriori* in order to recover the Navier-Stokes equations upon application of the coefficient-matching Chapman-Enskog multiscale analysis or an *inverse Chapman-Enskog expansion*. This model is suitable for simulating isothermal (near-incompressible or low-Mach number) flows, which are encountered extensively in engineering fluid flows. In fact, with suitable force models, they form the basis for even complex flows, such as multiphase, multicomponent flows. Numerical-diffusion-free advection, computational efficiency, simplicity of the algorithm, etc. are part of the appeal that makes the LB method a popular choice to simulate a variety of macroscopic flows. A thorough comparison between LBM and conventional PDE-based CFD is provided in Chapter 11 of [112].

3.5 SUMMARY

This chapter concludes the presentation on the theoretical background of the Hermite-based LB formulation. Here, we discussed a general approach to truncation of the equilibrium distribution depending upon the hydrodynamics regime needed to be recovered. Thereafter, we discussed how we can obtain the exact and Galilean-invariant discrete moments of the truncated distributions using the Gauss-Hermite quadrature. The resulting on-lattice and off-lattice discrete velocity sets were presented and discussed in detail. From a thermal LB model perspective, we saw that to recover a Galilean-invariant energy equation at the Navier-Stokes level, we need a fourth-order truncated equilibrium distribution, along with a ninth-order quadrature (D2Q37 or D3Q121). We also discretized the discrete velocity Boltzmann equation in space-time to obtain the popular LBGK equation. We consider alternative, more stable space-time discretizations in Chap. 4. Finally, we can conclude that the Hermite-based formulation of the lattice Boltzmann method not only offers another interpretation of the known lattice Boltzmann models, but also offers a systematic procedure for deriving higher-order models.

4.0 OFF-LATTICE TEMPORAL SCHEMES

4.1 INTRODUCTION

A defining feature of the LBGK scheme is the coupling between the velocity and space-time discretizations. That is, for a particular discrete-velocity set, $\boldsymbol{\xi}_i$, the coupling automatically fixes the temporal and spatial steps through the relation $\Delta\boldsymbol{x} = \boldsymbol{\xi}_i \Delta t$. The coupling is, in fact, a carryover from the earliest LB models, which were based on LGA. However, as shown in Chap. 3, and originally in [2, 55, 105], the LGCA link can be severed by noticing that LBGK is nothing but a special finite difference scheme of the discrete velocity Boltzmann equation on a regular uniform lattice, which also defines the associated discrete particle velocities. This implies that once we select a suitable set of discrete velocities, space may be discretized into any curvi-linear coordinates, independent of the discrete velocities. The discretization of space and time is thus a numerical requirement and, importantly, is not tied to the discretization of the velocity-space. This important idea opens up a gamut of spatial discretization schemes that can be applied to the discrete velocity Boltzmann equation, such as the finite element, finite volume, or the finite difference method.

As a consequence, a subset of the LB method, called the off-lattice Boltzmann (OLB) method, was developed where space and time are *independently* discretized, i.e., $\Delta\boldsymbol{x} \neq \boldsymbol{\xi}_i \Delta t$. Consequently, in the OLB method, we do not have the simplicity of a Lagrangian-type of evolution (streaming), rather the evolution of f_i takes place in an Eulerian sense. In that sense, any evolution equation for distributions that is *not* of the stream-collide type is an OLB scheme. The earliest OLB schemes were geared mainly towards extending the *geometric flexibility* of the LB method, which was previously limited, due to the requirement of a uniform Cartesian mesh. Several OLB schemes with different spatial discretization methods, such as finite-volume (FV), finite-element (FE), and finite-difference (FD), along with their variants, have been developed for use with body-fitted grids.

In addition to improving the geometric flexibility of the LB method, *OLB method can also be used to solve the DVBE with higher-order lattices*. As described in Chap. 2, higher-order lattices are sets of discrete velocities which are more suited to model more complex flows, such as thermal flows, micro-scale (high Knudsen number) flows, etc. In many of these velocity sets (*non-space-filling* or *off-lattice*), the discrete velocities cannot be expressed as an integer multiple of the smallest non-trivial speed. Since the regular stream-collide type of evolution scheme cannot be employed with such velocity sets, the OLB schemes provide a viable evolution scheme for the DVBE. Moreover, even if stream-collide can be used with higher-order on-lattice velocity sets, implementations in the presence of wall boundaries have been limited so far, due to difficulties in extending the bounce-back boundary treatment [97]. On the other hand, in OLB scheme, boundary conditions can be prescribed in simpler manner, similar to FD schemes. In short, OLB schemes enable both geometric flexibility and an ability to use off-lattice velocity sets.

In this chapter, we analyze the various off-lattice temporal schemes in detail. In particular, we focus on explicit and implicit models of the collision involved in time-integration schemes.

4.2 BACKGROUND

The basis for all OLB schemes is the discrete velocity Boltzmann equation, which is given as:

$$\frac{\partial f_i}{\partial t} + \boldsymbol{\xi_i} \cdot \nabla_{\boldsymbol{x}} f_i = -\frac{1}{\tau}(f_i - f_i^{eq}) \tag{4.1}$$

Since the discrete velocities $\boldsymbol{\xi}_i$ are constants, the DVBE can be considered as a system of linear, first-order, ordinary differential equations (ODEs) with a weak source (collision) term. This assumption is generally valid only if the gradients of the conserved quantities in the flow are not too high, i.e., the collision term is not highly non-linear. The number of ODEs in the system equals the number of discrete velocities; for example; nine equations in case of the D2Q9 lattice. Therefore, in principle, commonly used time marching schemes for ODEs, such as Euler, Runge-Kutta, etc. can be employed for temporal discretization of Eq. (4.1). A second-order Runge-Kutta (RK2) based OLB scheme for Eq. (4.1) can be written as:

$$\begin{aligned} f_i^{n+\frac{1}{2}} &= f_i^n - \frac{\Delta t}{2} R_i^n \\ f_i^{n+1} &= f_i^n - \Delta t R_i^{n+\frac{1}{2}} \end{aligned} \tag{4.2}$$

where:

$$R_i^n \equiv -\left(\boldsymbol{\xi}_{i\alpha} \frac{\partial f_i^n}{\partial x_\alpha}\right) - \frac{1}{\tau}(f_i^n - f_i^{eq,n}) \tag{4.3}$$

with the gradient term expanded in 2D Cartesian co-ordinates as:

$$\boldsymbol{\xi}_i \cdot \nabla f_i^n = \boldsymbol{\xi}_{i\alpha} \frac{\partial f_i^n}{\partial x_\alpha} = \xi_{ix} \frac{\partial f_i^n}{\partial x} + \xi_{iy} \frac{\partial f_i^n}{\partial y}. \tag{4.4}$$

Here $f_i^{n+\frac{1}{2}} \equiv f_i(\boldsymbol{x}, t_n + \frac{\Delta t}{2})$, $f_i^{n+1} \equiv f_i(\boldsymbol{x}, t_n + \Delta t)$, etc. Similar expressions can be written for the fourth-order RK scheme (RK4) [94]. In these schemes, the viscosity was related to the relaxation time through $\nu = c_s^2 \tau$.

Many of the earliest OLB schemes employed the forward Euler, RK2 or RK4 schemes, in combination with a variety of spatial discretization schemes. Using the RK-based time marching schemes, the geometric flexibility of the LB method was extended to non-Cartesian domains, non-uniform grids, body-fitted, stretched grids, etc. [19, 62, 94, 116]. In the case of FD spatial discretization, the spatial order-of-accuracy can also be increased arbitrarily, using higher-order Taylor approximations of the gradient terms. On the other hand, several discrete velocity models with non-space-filling velocity-sets also employed the RK-based schemes as the evolution equation [122].

Despite the geometric flexibility made possible by the RK-based schemes, the size of the Δt relative to τ has to be kept very small to maintain numerical stability. The constraint on Δt comes from an *explicit* approximation of the advection and collision terms of the Eq. (4.1). An explicit advection approximation imposes a stability criterion on the size of Δt through the *CFL* condition, $CFL = \boldsymbol{\xi}_i \Delta t / \Delta x < 1$. The *CFL* condition is well-understood to be a necessary condition for the stability of advection type of equation. However, the overall stability of the scheme is governed by the more restrictive condition on Δt due to an explicit approximation of collision, given by the approximate condition $\Delta t < \tau$ [120, **?**].

The small Δt requirement is particularly restrictive in the case of high Reynolds number (Re) flows where τ is very small. Moreover, in the LB method, the Mach number Ma in the simulations has to be kept small (generally less than 0.1) to limit the compressibility errors. Small values of Ma lead to a slower convergence rate, especially for steady-state flow problems [50, 118]. Thus, the combined effects of small Ma and Δt increase the overall computational cost of the RK-based OLB schemes. Therefore, OLB schemes that permit $\Delta t/\tau$ are needed to address this issue. The schemes presented in the next section aim to do that.

4.3 CHARACTERISTICS-BASED SCHEMES

To develop OLB alternatives to the RK-based methods, consider the $\theta-$*method* based time integrated form of the Eq. (4.1) along the *characteristics* (c.f. Eq. (3.26)). We rewrite that equation here:

$$\tilde{f}_i^{n+1} = \tilde{f}_i^n + \Delta t \left[(1-\theta)\tilde{\Omega}_i^n + \theta \tilde{\Omega}_i^{n+1} \right], \tag{4.5}$$

where $\Omega_i \equiv -\frac{1}{\tau}(f_i - f_i^{eq})$; a tilde indicates a term on the characteristic line, i.e., $\tilde{f}_i^{n+1} = f_i(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t, t + \Delta t)$, $\tilde{f}_i^n = f_i(\boldsymbol{x}, t)$; and $0 \leq \theta \leq 1$ [11, 71]. For $\theta = \{0, \frac{1}{2}, 1\}$, we obtain an explicit $\mathcal{O}(\Delta t)$, an implicit $\mathcal{O}(\Delta t^2)$, and an implicit $\mathcal{O}(\Delta t)$ approximations of the collision term, respectively. The $\mathcal{O}(\Delta t)$ schemes are Euler-type schemes, and the $\mathcal{O}(\Delta t^2)$ is a Crank-Nicholson-type or a trapezoidal scheme. The LBGK equation (c.f. Eq. (3.27)) was derived for the particular case $\theta = 0$.

4.3.1 Implicit-Collision Formulation

It is well-known in the ODE theory that an implicit approximation of the relaxation (collision) term is critical for numerical stability, especially for *stiff* equations (highly non-linear flows). So, we first develop semi-implicit formulation for the LBGK scheme, after which it is extended to the off-lattice case. Employing $\theta = \frac{1}{2}$, we obtain an implicit $\mathcal{O}(\Delta t^2)$ approximation of the collision term, whereby Eq.(4.5) becomes:

$$\tilde{f}_i^{n+1} = \tilde{f}_i^n + \frac{\Delta t}{2\tau}\big(\tilde{\Omega}_i^{n+1}\big) - \frac{\Delta t}{2\tau}\big(\tilde{\Omega}_i^n\big) + \mathcal{O}(\Delta t^2) \tag{4.6}$$

On inspection of Eq. (4.6) we can find that it is a non-linear equation. This is because, to evaluate f_i^{eq} in Ω_i at $n+1$, we need $\rho, \boldsymbol{u}, \theta$ values at time $n+1$, which are unknown since f_i^{n+1} is unknown. Of course we can solve the non-linear equation: $f_i^{eq,n+1} = \mathcal{G}\big(f_i^{n+1}\big)$, where $\mathcal{G}$ is a non-linear function. However, the iterative procedure for solving non-linear equation destroys the fully explicit and simple algorithm of the LB method, in addition to adding extra computational cost.

In order to avoid an iterative procedure to approximate an implicit-collision, an ingenious *variable transformation* is often employed in the LB method to mask the implicitness of the collision term, first proposed by He, Chen, and Doolen [54]. In the variable transformation technique, a new distribution g_i, is defined as:

$$g_i(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) - \Delta t \theta \Omega_i(f_i(\boldsymbol{x}, t)), \tag{4.7}$$

for a generic collision kernel $\Omega_i(f(\boldsymbol{x}, t))$ [11]. With the variable transformation, and for the specific case of $\theta = \frac{1}{2}$ and BGK kernel, Eq. (4.5) transforms to:

$$g_i\big(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t, t + \Delta t\big) - g_i(\boldsymbol{x}, t) = -\frac{\Delta t}{\lambda}\big(g_i(\boldsymbol{x}, t) - g_i^{eq}(\boldsymbol{x}, t)\big) \tag{4.8}$$

where we have expanded the terms with the tilde. Eq. (4.8) can be viewed as an *implicit LBGK* formulation, but which is computationally still fully explicit. The term λ is called the *modified relaxation time*, which is related to the original relaxation time, τ, by:

$$\lambda = \tau + \frac{\Delta t}{2}. \tag{4.9}$$

In the presence of external forces, it is important to observe that in the system represented by the new distribution g_i, the macro variables have to be adjusted:

$$\hat{\rho} = \sum_{i=0}^{q-1} g_i = \sum_{i=0}^{q-1} f_i = \rho \tag{4.10}$$

$$\hat{\rho}\hat{\boldsymbol{u}} = \sum_{i=0}^{q-1} \boldsymbol{\xi}_i g_i = \sum_{i=0}^{q-1} \boldsymbol{\xi}_i \left(f_i + \frac{\Delta t}{2\tau}(f_i - f_i^{(0)}) - \frac{1}{2}F_i \right) = \rho \boldsymbol{u} - \frac{\Delta t}{2}\rho \boldsymbol{g} \tag{4.11}$$

where $\hat{\rho}, \hat{\boldsymbol{u}}$ are the density and velocity based on the new distribution g_i. It should be noted that $\hat{\boldsymbol{u}}$ is *not* the 'physical velocity,' which in fact, is given by f_i. From Eqs. (4.10) and (4.11), the actual physical velocity $\boldsymbol{u}$ is computed as:

$$\boldsymbol{u} = \hat{\boldsymbol{u}} + \frac{\Delta t}{2}\boldsymbol{g}. \tag{4.12}$$

Importantly, the equilibrium distribution function, $f_i^{(0)}$ should be computed with $\boldsymbol{u}$ and *not* $\hat{\boldsymbol{u}}$ in order to get accurate results. Denoting $g_i^{(0)}(\rho, \boldsymbol{u}) \equiv f_i^{(0)}(\rho, \boldsymbol{u})$ for notational consistency, we finally obtain the implicit formulation, complete with the force term:

$$g_i\big(\boldsymbol{x} + \boldsymbol{\xi}_i\Delta t, t + \Delta t\big) = g_i(\boldsymbol{x}, t) - \frac{\Delta t}{\lambda}\Big(g_i(\boldsymbol{x}, t) - g_i^{(0)}(\boldsymbol{x}, t)\Big) + \Delta t\Big(1 - \frac{\Delta t}{2\lambda}\Big)\boldsymbol{F}_i(\rho, \boldsymbol{u}) \tag{4.13}$$

Denoting $\boldsymbol{G}_i \equiv (1 - \frac{\Delta t}{2\lambda})\boldsymbol{F}_i$, we obtain:

$$\boxed{g_i\big(\boldsymbol{x} + \boldsymbol{\xi}_i\Delta t, t + \Delta t\big) = g_i(\boldsymbol{x}, t) - \frac{\Delta t}{\lambda}\big(g_i(\boldsymbol{x}, t) - g_i^{(0)}(\boldsymbol{x}, t)\big) + \Delta t\boldsymbol{G}_i} \tag{4.14}$$

Due to the implicit approximation of the collision term through variable transformation, the $\Delta t < \tau$ constraint no longer applies. This observation is validated through extensive testing in the following sections. The only constraint on Δt is the *CFL* condition due to an explicit approximation of advection.

4.3.2 Off-lattice Boltzmann Schemes

In the LBGK scheme, with implicit or explicit collision approximation, the advection of the distributions is explicit and happens on the characteristic line in a Lagrangian fashion. In order to develop off-lattice schemes, we need to be able to set Δt independent of $\Delta\boldsymbol{x}$. Towards that consider the implicit formulation that was developed in the previous section, but now written in the tilde:

$$\tilde{g}_i^{n+1} = \tilde{g}_i^n - \frac{\Delta t}{\lambda}\Big(\tilde{g}_i^n - \tilde{g}_i^{eq,n}\Big) \tag{4.15}$$

Now, consider a characteristic line $\tilde{\boldsymbol{x}}(x_{ref}, t_{ref}, t)$ passing through the spatial point $\boldsymbol{x}_{ref} = \boldsymbol{x} + \boldsymbol{\xi}_i\Delta t$ at time $t_{ref} = t + \Delta t$. This choice of the reference point is known to enhance the the stability of the numerical scheme [130]. Another point on the same characteristic, but

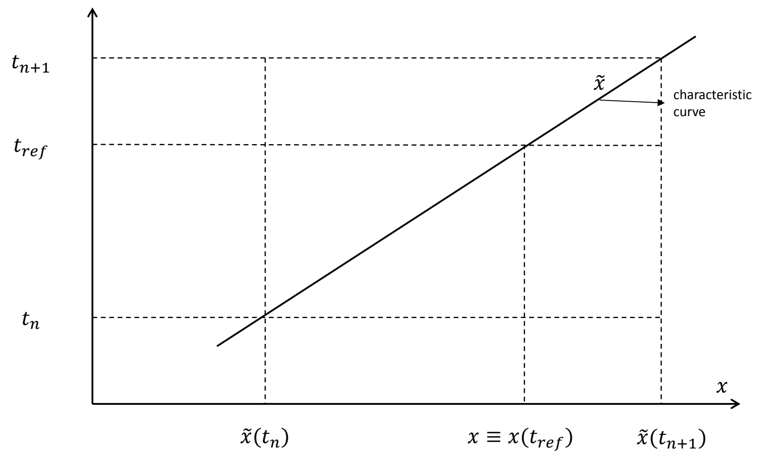


Figure 5: Reference points for discretization along characteristics. Adapted from [130]

located backwards in time with respect to $\tilde{\boldsymbol{x}}(t+\Delta t)$ can be expressed as (See Fig. 5 for illustration):

$$\tilde{\boldsymbol{x}}(t) = \tilde{\boldsymbol{x}}(t+\Delta t) - \Delta t \boldsymbol{\xi}_i. \tag{4.16}$$

We want to express $g_i(\tilde{\boldsymbol{x}}(t))$ in Eq. (4.15) in terms of $\boldsymbol{x}_{ref}$. For that, consider the Taylor-series expansion of $g_i(\tilde{\boldsymbol{x}}(t))$ around $\boldsymbol{\xi}_i \Delta t$ [11]:

$$\begin{aligned} g_i(\tilde{\boldsymbol{x}}(t), t) &= g_i(\tilde{\boldsymbol{x}}(t+\Delta t) - \boldsymbol{\xi}_i \Delta t, t) \\ &= g_i(\boldsymbol{x}(t+\Delta t), t) - \Delta t \xi_{i\alpha} \frac{\partial g_i(\boldsymbol{x}(t+\Delta t), t)}{\partial x_\alpha} + \frac{\Delta t^2}{2} \xi_{i\alpha} \xi_\beta \frac{\partial^2 g_i(\boldsymbol{x}(t+\Delta t), t)}{\partial x_\alpha \partial x_\beta} \\ &= g_i(\boldsymbol{x}_{ref}, t) - \Delta t \xi_{i\alpha} \frac{\partial g_i(\boldsymbol{x}_{ref}, t)}{\partial x_\alpha} + \frac{\Delta t^2}{2} \xi_{i\alpha} \xi_{i\beta} \frac{\partial^2 g_i(\boldsymbol{x}_{ref}, t)}{\partial x_\alpha \partial x_\beta} + \mathcal{O}(\Delta t^3). \end{aligned} \tag{4.17}$$

In the same fashion, we can Taylor expand $\tilde{g}_i^{eq,n} \equiv g_i^{eq}(\tilde{\boldsymbol{x}}(t), t)$. Here $\xi_{i\alpha}$ represents the components $\boldsymbol{\xi}_i$ along the Cartesian coordinates; for example, $\boldsymbol{\xi}_{i\alpha} = \{\xi_{ix}, \xi_{iy}, \xi_{iz}\}$ along the x, y, z coordinate system. Substituting the above equation in Eq. (4.15), and recognizing that $\tilde{g}_i^{n+1} \equiv g_i(\boldsymbol{x} + \boldsymbol{\xi}_i \Delta t, t + \Delta t) = g_i(\boldsymbol{x}_{ref}, t + \Delta t)$, we obtain:

$$\begin{aligned} g_i^{n+1} &= g_i^n - \Delta t \left[\xi_{i\alpha} \frac{\partial g_i^n}{\partial x_\alpha} + \frac{1}{\lambda}(g_i^n - g_i^{eq,n}) \right] + \frac{\Delta t^2}{2} \xi_{i\alpha} \frac{\partial}{\partial x_\alpha} \left[\xi_{i\beta} \frac{\partial g_i^n}{\partial x_\beta} + \frac{2}{\lambda}(g_i^n - g_i^{eq,n}) \right] \\ &\quad - \frac{\Delta t^3}{2\lambda} \xi_{i\alpha} \frac{\partial}{\partial x_\alpha} \left[\xi_{i\beta} \frac{\partial (g_i^n - g_i^{eq,n})}{\partial x_\beta} \right], \end{aligned} \tag{4.18}$$

where $g_i^n \equiv g_i(\boldsymbol{x}_{ref}, t)$ and $g_i^{n+1} \equiv g_i(\boldsymbol{x}_{ref}, t + \Delta t)$.

Eq. (4.18) represents a general off-lattice time-marching scheme applicable in three-dimensions. It is noted that, although the collision approximation in the scheme looks explicit, it is implicit nonetheless, and hence *unconditionally stable for $\theta \geq 1/2$, within the CFL limit for advection.* For comparison purposes, we will also describe some alternative off-lattice time-marching schemes that are available in the literature. These alternative schemes are useful for validation purposes of Eq. (4.18).

Lee and Lin proposed two off-lattice time-marching scheme, and used the FE method to discretize the spatial domain [71]. One of their proposed schemes, termed as AE/CE, approximated both advection and collision explicitly. The other proposed scheme, termed as AE/CI, approximated the advection explicitly, but collision implicitly, much like Eq. (4.18). However, they did not use the variable transformation technique to hide the implicitness, and hence an iterative predictor-corrector scheme was required, which increased the computational cost and complexity of the simulations. The AE/CE scheme is given by:

$$\begin{aligned} f_i^{n+1} &= f_i^n - \Delta t \left[\xi_{i\alpha} \frac{\partial f_i^n}{\partial x_\alpha} + \frac{1}{\lambda}(f_i^n - f_i^{eq,n}) \right] + \Delta t^2 \xi_{i\alpha} \frac{\partial}{\partial x_\alpha} \left[\frac{1}{2} \xi_{i\beta} \frac{\partial f_i^n}{\partial x_\beta} + \frac{1}{\lambda}(f_i^n - f_i^{eq,n}) \right] \\ &\quad - \frac{\Delta t^3}{2} \frac{1}{\lambda} \xi_{i\alpha} \frac{\partial}{\partial x_\alpha} \left[\xi_{i\beta} \frac{\partial}{\partial x_\beta}(f_i^n - f_i^{eq,n}) \right]. \end{aligned} \tag{4.19}$$

Guo and Zhao also used a variable transformation to obtain an implicit approximation of the collision [49], however their advection approximation was only $\mathcal{O}(\Delta t)$. Their scheme, termed as the GZ scheme in this work, can be written as:

$$g_i^{n+1} = f_i^n - \Delta t \xi_{i\alpha} \frac{\partial f_i^n}{\partial x_\alpha} - \frac{\Delta t}{2\tau}(f_i^n - f_i^{eq,n}). \tag{4.20}$$

Table 2: Summary of various off-lattice Boltzmann time-marching schemes. Here E indicates explicit, and I indicates implicit. Spatial order-of-accuracy for advection depends upon the spatial discretization method and schemes used, and hence is excluded.

Scheme	Advection	Collision	$\nu-\lambda$ relation
RK2	$\mathcal{O}(\Delta t^2)$ E	$\mathcal{O}(\Delta t^2)$ E	$\lambda=\nu/c_s^2$
AE/CE	$\mathcal{O}(\Delta t^2)$ E	$\mathcal{O}(\Delta t^2)$ E	$\lambda=\nu/c_s^2+0.5\Delta t$
AE/CI	$\mathcal{O}(\Delta t^2)$ E	$\mathcal{O}(\Delta t^2)$ I	$\lambda=\nu/c_s^2$
GZ	$\mathcal{O}(\Delta t)$ E	$\mathcal{O}(\Delta t^2)$ I	$\lambda=\nu/c_s^2$
Eq.(4.18) (BKG)	$\mathcal{O}(\Delta t^2)$ E	$\mathcal{O}(\Delta t^2)$ I	$\lambda=\nu/c_s^2+0.5\Delta t$

Finally, the various off-lattice time-marching schemes described above are summarized in Table 2.

4.3.3 Implementation of an OLB scheme

Eq. (4.18) represents a typical temporal evolution scheme for the distributions used in the OLB method. Clearly, the usual stream-collide scheme type of evolution on the standard LBM is replaced by a procedure that involves solving a set of ODEs at each node. For the scheme given by Eq.(4.18), the algorithm consists of the following steps:

1. Initialize g_i^0 according to prescribed initial conditions, i.e., set $g_i^0=g_i^{eq,0}$, where $g_i^{eq,0}=g_i^{eq}(\rho^0,\boldsymbol{u}^0)$. ρ^0 and $\boldsymbol{u}^0$ are the prescribed initial conditions.
2. Evaluate the gradient terms $\frac{\partial g_i^n}{\partial x}$, $\frac{\partial^2 g_i^n}{\partial x^2}$, etc. In the case of FD spatial discretization, with a central-differencing scheme, these quantities can be computed as:

$$\begin{aligned}\frac{\partial g_i^n}{\partial x} &= \frac{g_i(x+\Delta x,y,t_n)-g_i(\boldsymbol{x}-\Delta x,y,t_n)}{2\Delta x}+\mathcal{O}(\Delta x^2),\\ \frac{\partial^2 g_i^n}{\partial x^2} &= \frac{g_i(\boldsymbol{x}+\Delta\boldsymbol{x},y,t_n)-2g_i(x,y,t_n)+g_i(\boldsymbol{x}-\Delta x,y,t_n)}{\Delta x^2}+\mathcal{O}(\Delta x^2).\end{aligned}\quad(4.21)$$

Similar expressions can be written for the first-order derivatives in the y-direction, and the second-order mixed derivatives $\frac{\partial^2 g_i}{\partial x \partial y}$. Other schemes such as upwind-differencing can also be employed.

3. Evaluate g_i^{n+1} per Eq. (4.18).
4. Evaluate ρ^{n+1}, $\hat{\boldsymbol{u}}^{n+1}$, $\boldsymbol{u}^{n+1}$. Compute $g_i^{eq,n+1}$ based on $\boldsymbol{u}^{n+1}$.
5. If $\boldsymbol{u}^{n+1}$ has converged, then stop, if not, repeat steps 2-5.

4.4 BOUNDARY TREATMENT

In LB simulations, it can be argued that developing accurate and efficient boundary schemes is as important as developing accurate physics models, since they influence the stability and accuracy of the solutions. Normally in flow simulations, boundary conditions are prescribed on the primitive flow (macroscopic) variables, such as no-slip velocity wall or velocity inlet, pressure outlet, etc. In the LB method, however, the primitive variable of the simulation are the distributions. Although obtaining macroscopic variables from the distributions is trivial (through moment summations), the reverse is not so easy. This is because, the distributions are not unique for a particular prescribed value of macroscopic variables. Treatment of various boundary conditions is too vast a topic to be covered in this dissertation, so only a brief overview is provided. Moreover, since we are focusing on flow domains with straight walls/boundaries, only such boundary schemes are discussed. Two broad methods are used to deal with this issue:

1. *Heuristic or modified distributions*: In this method, *only the unknown distributions* at the boundaries are obtained using some (heuristic) physical rules, such as the bounceback rule, or the mass and momentum conservation principle. The most widely used scheme, viz., the full-way and half-way bounce-back rule, as well as the Inamuro scheme, Zou/He scheme, are some of the schemes which belong to this method. Such schemes, however, cannot be extended easily to velocity sets with larger neighborhoods, such as D2Q37. Specifying gradients of flow variables and handling moving boundaries are also difficult in this method.
2. *Reconstructed distributions:* In this method, *all the distributions (known and unknown)* at the boundaries are reconstructed based on the prescribed density (pressure), velocity, and temperature. Among the schemes of this methods are the finite-difference type LB boundary scheme [109, 28], Guo and Zhao's extrapolation scheme [51] and the regularized boundary [68]. These boundary schemes are found to be more stable for coarser meshes and more importantly, can be extended for off-lattice or non-D2Q9 cases. A thorough review of various boundary schemes for flat (straight) boundaries is provided in [69, 112], and for curved boundaries in [47].

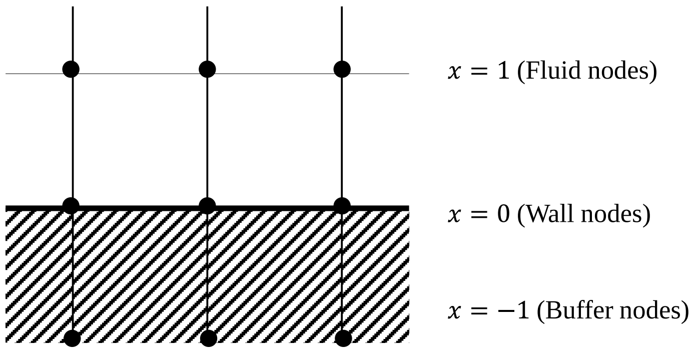


Figure 6: Schematic for extrapolation boundary scheme

In this work, for both isothermal and thermal flows, we employ the *extrapolation type boundary scheme*, first proposed by Chen *et al.* [28]. It is based on the fact that the LB method is a special finite-difference scheme for the discrete velocity Boltzmann equation. The implementation follows of the scheme is as follows: As shown in Fig. 6, a ghost or buffer layer of nodes is inserted beyond the physical wall boundary. The nodes of the physical wall are as considered fluid nodes, in the sense that, the regular collision (which involves computation of local equilibrium distributions, $f_i^{eq,n}$) and time-update (computation of g_i^{n+1}per (4.18)) are both performed on them. The equilibrium distributions on the wall nodes, however, are computed on the basis of the prescribed wall velocity and/or temperature. Finally, before g_i^{n+1} is computed as the wall boundary nodes, the distributions on the buffer nodes, are obtained using a *second-order extrapolation* of the distributions of the inner fluid nodes:

$$g_i(\boldsymbol{x_{-1}}) = 2g_i(\boldsymbol{x_0}) - g_i(\boldsymbol{x_1}) \tag{4.22}$$

where $-1, 0, 1$ are the labels for the nodes on the ghost, (wall) boundary, and the first fluid layer, respectively. This scheme, conceptually, is similar to the boundary schemes used in conventional finite-difference methods. It can be shown that this scheme is second-order, $\mathcal{O}(\Delta x^2)$, consistent with the order of the rest of simulation domain, and more importantly in our case, can be easily applied to higher-order lattices. Compared to a single layer of buffer nodes required in this scheme, attempts to develop bounce-back-type schemes for the

D2Q39 require three buffer nodes, resulting in a more complicated scheme [97]. In theory, the extrapolation scheme can also be extended to curvilinear coordinates through suitable coordinate transformation.

4.5 NUMERICAL VALIDATION

To test the numerical stability of the various schemes, as a function of the maximum stable $\Delta t/\tau$, several steady and unsteady, incompressible, two-dimensional flows were simulated, and their results are presented in this section. Since the focus of this work is on evaluation of the temporal schemes, the simpler finite-difference method is used to discretize the spatial domains. For uniformity, Eq. (4.21) is used to evaluate the derivatives in the gradient term for all the OLB schemes tested here. The central-difference scheme is chosen since they are less diffusive than other $\mathcal{O}(\Delta x^2)$ schemes. This minimizes the contribution of numerical diffusion to the overall stability of the scheme. The numerical stability of a scheme can be concluded by the maximum allowable Δt for a particular Re and grid size. In other words, for a particular flow problem, we fix the Re (fixed τ) and grid size (fixed Δx) and vary Δt, until the simulation becomes unstable.

The schemes were coded in C++ using the Armadillo linear algebra library [95], and parallelized for a shared-memory architecture using OpenMP. The detailed implementation and pseudo code is presented in Chap. 6.

4.5.1 Taylor Vortex Flow

To test the stability and accuracy of various schemes without the artifacts of a boundary treatment, we first simulate the Taylor-Green vortex flow. This flow represents the unsteady flow of a freely decaying two-dimensional vortex, and is often used to evaluate the effective viscosity and temporal and spatial accuracy of a scheme. Here, the flow is computed within a periodic square box defined as $-\pi \leq x, y \leq \pi$ with a uniform Cartesian mesh of size 100×100 on the domain. The analytical solutions of the flow-field are given by:

$$
\begin{aligned}
u(x,y,t) &= -u_{ref}\cos(k_1 x)\sin(k_2 y)\exp[-\nu(k_1^2+k_2^2)t], \\
v(x,y,t) &= u_{ref}\frac{k_1}{k_2}\sin(k_1 x)\cos(k_2 y)\exp[-\nu(k_1^2+k_2^2)t], \\
p(x,y,t) &= p_{ref} - \frac{u_0^2}{4}\left[\cos(2k_1 x) + \frac{k_1^2}{k_2^2}\cos(2k_2 y)\right]\exp[-2\nu(k_1^2+k_2^2)t].
\end{aligned}
\tag{4.23}
$$

In our simulations, to minimize compressibility effects, the Mach number is set as 0.01. The reference velocity is $u_{ref} = Ma \times c_s$, the reference density is $\rho_{ref} = 1$, the reference pressure is $p_{ref} = \rho_{ref}/c_s^2$, and the wave numbers k_1 and k_2 are constants set to 1.0 and 4.0, respectively. The non-equilibrium initialization scheme proposed by [109] is used to initialize the DFs, and periodic boundary conditions are applied in the x and y directions. The Re of the flow is 100.

Figs. 7 and 8 show the horizontal and vertical velocity profiles at different times as obtained using the BKG scheme. The velocity profiles are shown for $t/t_c = 0.5, 1$, and 2, where $t_c = \ln 2/\nu(k_1^2 + k_2^2)$ is the time at which the amplitude of the decay is halved. Importantly, these simulation have been obtained with $\Delta t = 10\tau$, where $\tau = \nu/c_s^2$. The corresponding CFL number is 0.7. This confirms the observations of high $\Delta t/\tau$ made in [11]. Moreover, the average relative error as defined by:

$$
E = \frac{\sum_y \left(|u_{num} - u_{exact}| + |v_{num} - v_{exact}|\right)}{\sum_{,y}\left(|u_{exact}| + |v_{exact}|\right)}
$$

is $< 1\%$. Here u_{num} is the simulated velocity, u_{exact} is the analytical velocity, and the summation is over vertical plane at $x = 0$. This result demonstrates that the BKG scheme successfully overcomes the $\Delta t < \tau$ restriction imposed by the RK-based OLB schemes. The only remaining restriction on Δt is the *local* CFL number which is due to an explicit advection approximation.

For comparison, we also simulated the Taylor-Green vortex flow using the GZ and the AE/CE schemes. The GZ scheme, although having treated the collision term implicitly, could not achieve higher $\Delta t/\tau$ values. In fact, in their simulations, $\Delta t/\tau \approx 1.6$ for a $Re \approx 63$ with a mixed (central-upwind) differencing scheme. Furthermore, for a purely central-difference

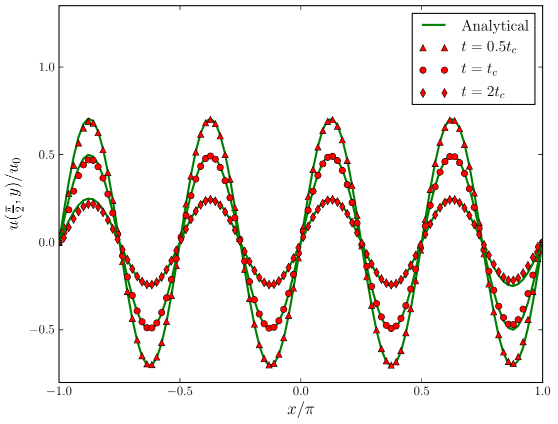


Figure 7: Horizontal velocity profile of Taylor-Vortex flow at various times. The numerical results have been obtained with a $\Delta t = 10\tau$ using the BKG scheme.

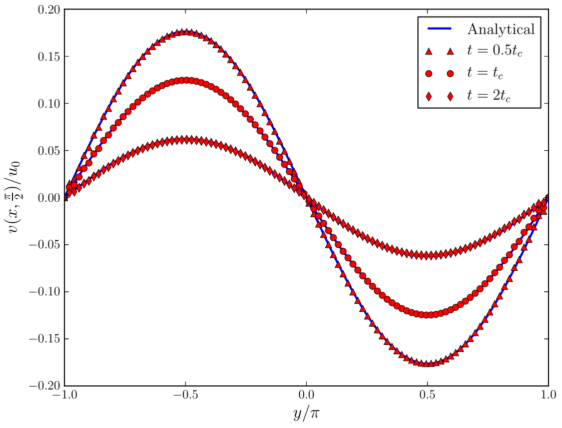


Figure 8: Vertical velocity profile of Taylor-Vortex flow at various times with $\Delta t = 10\tau$ using the BKG scheme.

scheme, i.e., without the additional marginal stability of upwind schemes, the authors showed analytically using the von-Neumann stability analysis, that the maximum $\Delta t/\tau$ that can be attained is less than 1.5, and that at low CFL numbers. We have observed that the AE/CE permits $\Delta t > \tau$, but stable simulations are obtained at much lower values of CFL number ($CFL < 0.2$) compared to the BKG scheme.

4.5.2 2-D Plane Poiseuille Flow

Another useful numerical test is Poiseuille flow. A plane Poiseuille flow describes the steady, laminar flow of an incompressible fluid in a rectangular channel driven by a pressure gradient. Assuming symmetry and incompressibility, it can be shown that for Poiseuille flow, the Navier-Stokes momentum equation reduces to:

$$\mu \frac{\partial^2 u}{\partial y^2} = \frac{\partial p}{\partial x} \tag{4.24}$$

which has a exact steady state solution for velocity given by:

$$\begin{aligned} u(y) &= 4u_{max}\frac{y}{H}\left(1-\frac{y}{H}\right) \quad \text{for } 0 \le y \le H \\ v(x,y) &= 0 \end{aligned} \tag{4.25}$$

where H is the channel height and u_{max} is the center-line velocity where the magnitude of velocity is the maximum. The Reynolds number of the flow is defined by $Re = u_{max}H/\nu$. In a LB simulation, imposing a pressure difference by specifying inlet and outlet densities increases the compressibility errors [112]. Therefore, the effect of Δp is imposed on the flow through an equivalent body-force F. This force has the same effect as having a pressure-gradient in the channel which produces the chosen u_{max}. This body force can be evaluated from $F = 8u_{max}\rho\nu/H^2$.

For the reference case, the BKG scheme is used to obtain the results presented here. In the simulation, we set $H = 1$ and $L = 1$, where L is the channel length. The domain is discretized into a uniform Cartesian mesh of size 100×100. The Mach number based on u_{max} is set to 0.1732, which corresponds to a u_{max} of 0.1. The velocity is initialized to zero everywhere and the average density is set to one. Since a steady-state flow is simulated, the equilibrium initial condition is applied to the distributions. The boundary scheme described in Sec. 4.4 is applied on the no-slip top and bottom walls, and a periodic boundary condition is applied at the inlet and outlets. Various Re flows are simulated by varying the viscosity but keeping the Ma constant. The simulations are run until a steady state criterion, defined as:

$$C = \frac{\sum_{i,j} |u_{i,j}^{n} - u_{i,j}^{n-50}|}{\sum_{i,j} |u_{i,j}^{n}|} < 10^{-6}$$

is attained. Here $u_{i,j}^{n} \equiv u(x_i, y_j, n\Delta t)$, where n is the time-step number, and the summation is over the entire flow field.

The simulated steady-state stream-wise velocity profile for $Re = 100$, along with the analytical solution is shown in Fig. 9. Clearly, the simulated and the analytical values are in excellent agreement with each other. The inset shows the velocity profile close to the wall, which is also in close agreement to analytical values. This serves as validation for the applicability and accuracy of the non-equilibrium boundary condition used in the simulation. It is worth noting that the velocity profile was generated on a 100×100 grid with a $\Delta t = 5\tau$. Even at such a high Δt, the average relative error, as defined earlier, is $\sim 2\%$.

For comparison, the plane Poiseuille flow is also simulated using the AE/CE scheme, and the GZ scheme. Fig. 10 shows the maximum allowable CFL number versus Re for the steady Poiseuille flow for the different schemes. This plot is obtained by fixing the grid size to 100×100 for each scheme, and then varying Δt. This is repeated for the various Re, as shown in the figure.

It can be seen that at low Re, both the AE/CE and BKG schemes have almost similar maximum CFL numbers. However, as Re increases, the maximum allowable CFL number is clearly much higher with the BKG scheme, as compared to the AE/CE scheme. This enhanced stability at higher Re results from the implicit treatment of the collision term, which becomes highly non-linear as Re increases, thus directly benefiting from the local implicit treatment. Importantly, the implicit treatment of collision, allows for $\Delta t > \tau$, thus resulting in large Δt and thereby reducing the computational effort.

4.5.3 Lid-Driven Cavity Flow

Isothermal, 2-D lid-driven cavity flow is commonly used as a benchmark case to test and evaluate numerical schemes for incompressible viscous flows. The flow domain consists of a square cavity with three stationary walls on the sides and bottom, and a top wall (lid) that moves with a uniform tangential velocity u_{lid}. Despite the simple geometry, this flow exhibits many complex flow patterns such as formation of vortices near corners due to singularities. In our context, the lid-driven cavity case is also useful for quantitatively evaluating the effects of collision approximations (explicit/implicit) on flows with moderately large non-linearities (high Re), and thus quantifying the overall stability of different schemes.

The computational domain consists of a square with height $L = H = 1$. The top wall moves with a constant velocity of $u_{lid} = 0.1$, and the Reynolds number of the flow is $Re = u_{lid}H/\nu$. The domain is discretized on a uniform mesh of size 257×257. The non-equilibrium extrapolation scheme is applied for all of the walls. The flow is initialized by

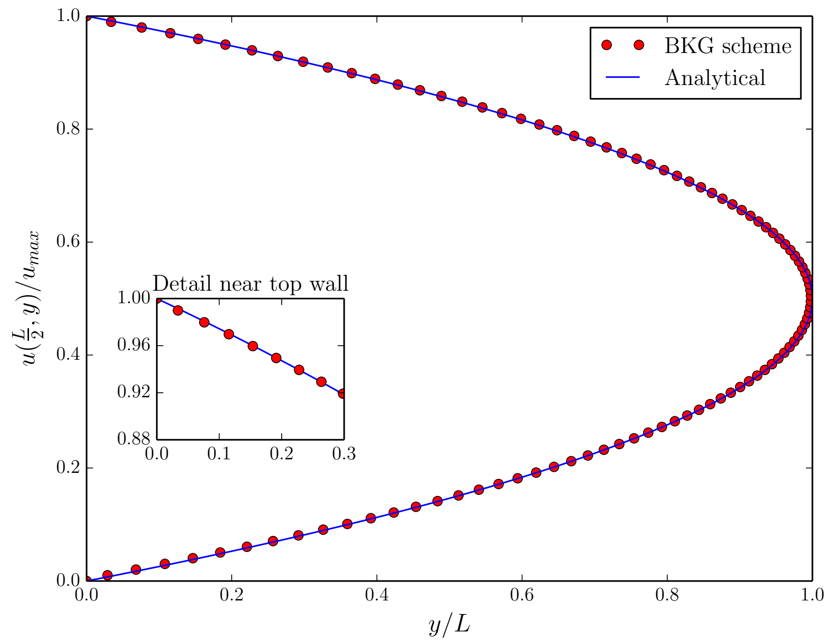


Figure 9: Steady-state horizontal velocity profile for Poiseuille flow at $Re = 100$. The results were obtained using $\Delta t = 5\tau$ on a 100×100 mesh using the BKG scheme.

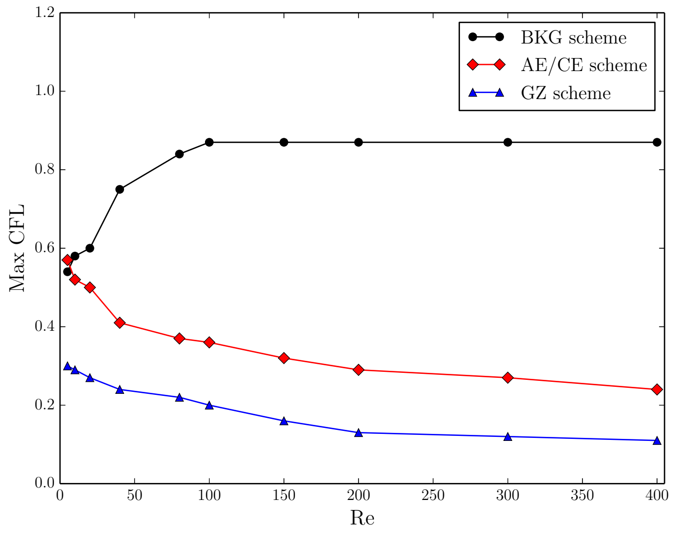


Figure 10: Maximum allowable CFL number vs. Re for steady Poiseuille flow.

setting $\rho = 1$ and $\boldsymbol{u} = 0$ in the entire flow. The flow is simulated using four schemes: BKG, AE/CE, GZ, and also a RK2-based scheme.

As before, for the reference case, we present the results of simulations using the BKG scheme. Steady-state velocity components along the horizontal and vertical centerlines at various Re are shown in Figs. 11 and 12. These results have been obtained with $\Delta t = 4\tau$ and $CFL = 0.5$. The results are compared with results from Ghia *et al.* who obtained their results with a 257×257 grid using the coupled strongly implicit multi-grid method, and a vorticity-stream-function formulation [41] . The velocity profiles change from curved at lower Re to linear for higher Re, which is consistent with the benchmark solutions. The near-linear velocity profiles at higher Re in the central core of the cavity indicates a region of uniform vorticity.

To assess and quantify the effects of the size of Δt on numerical stability, in Figs. 13-18, we plot the stability region for each scheme, as determined by two parameters: $\Delta t/\tau$ and $\Delta t/\Delta x$ (please note that the scales of the axes vary across the figures). Broadly speaking,

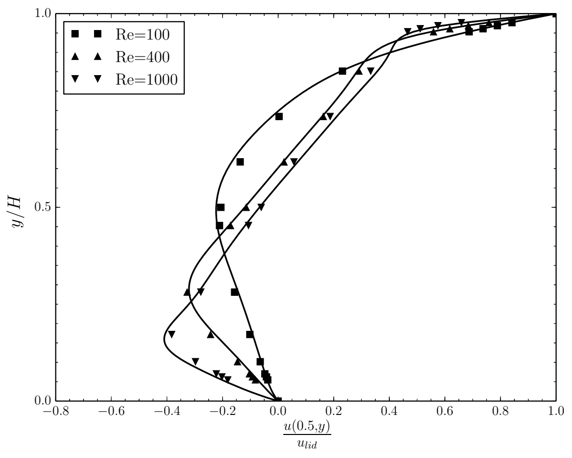


Figure 11: Centerline horizontal-velocity profile for a lid-driven cavity at various *Re*. The solid lines indicate the simulated values, while the markers (■, ▲, ▼) indicate the corresponding solution from Ghia et al. [41]

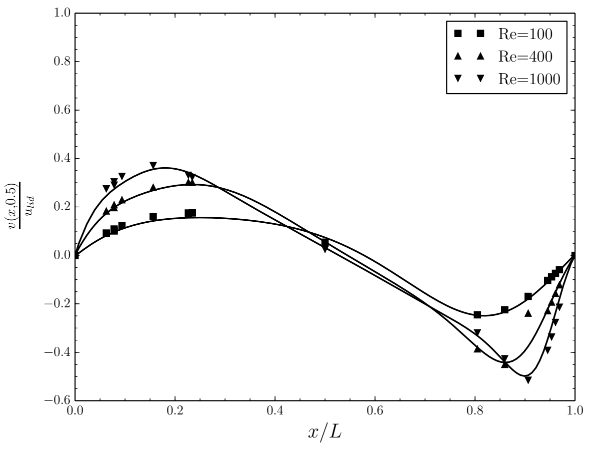


Figure 12: Centerline vertical-velocity profile for a lid-driven cavity at various *Re*. The solid lines indicate the simulated values, while the markers (■, ▲, ▼) indicate the corresponding solution from Ghia et al. [41]

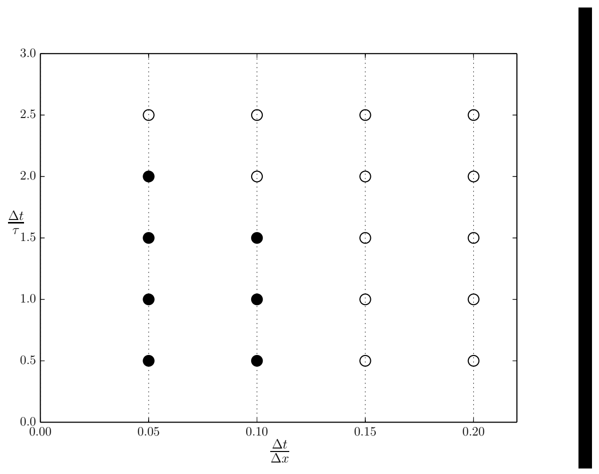


Figure 13: Stability region for the RK2 scheme at $Re = 100$.

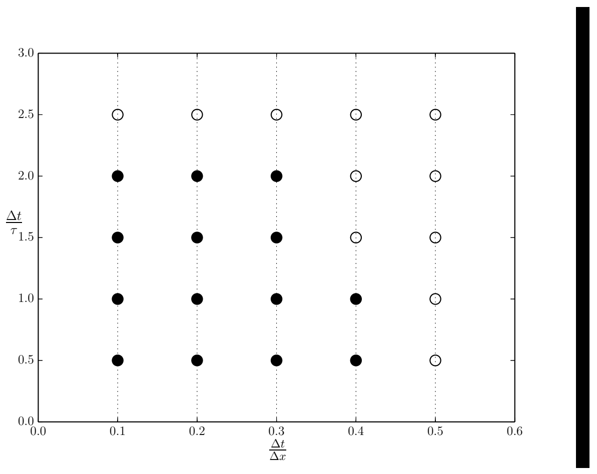


Figure 14: Stability region for the GZ scheme at $Re = 100$.

while $\Delta t/\tau$ represents the non-linear stability constraint imposed by collision, $\Delta t/\Delta x$ represent the linear-stability constraints of explicit advection. $\Delta t/\Delta x$ also represents the CFL number in the case of a D2Q9 lattice. Hence, a map determined by these two parameters should serve a guide to gauge the overall numerical stability of the schemes with respect to size of Δt. In all of the maps, $\bullet$ indicates a stable solution and $\circ$ indicates an unstable solution. Fig. 13 shows the stability region for the RK2-based OLB scheme with central differencing for $Re = 100$.

From Fig. 13, we can observe that the RK2-based scheme is unstable beyond $\Delta t/\tau > 2$, irrespective of the CFL number. Moreover, the stable solutions which are obtained at $\Delta t/\tau < 2$ are done so at very low CFL values. As described before, the small Δt requirement stems from the fact that as Re increases, the collision term becomes more non-linear and stiff, which requires an implicit approximation for numerical stability. Stability of RK2-based schemes can be increased slightly by adopting second-order upwind schemes; however, the marginal stability is added at the cost of increasing numerical diffusion. Similarly, multi-stage Runge-Kutta schemes such as RK4 can also increase stability. However, all RK-based OLB schemes involve multiple evaluation of the f_i^{eq} term for each advancement in Δt, the

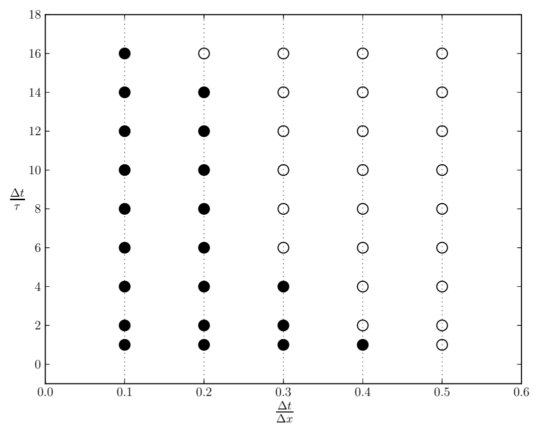


Figure 15: Stability region for the AE/CE scheme at $Re = 400$.

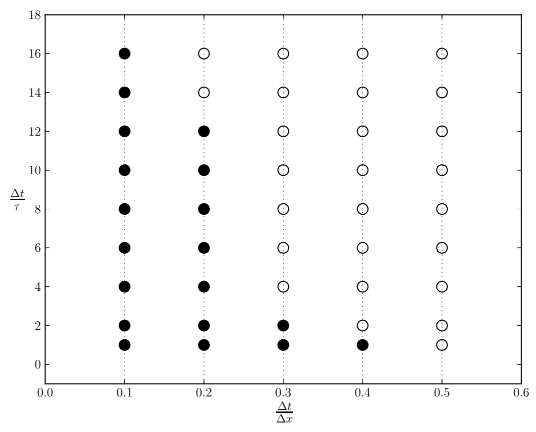


Figure 16: Stability region for the AE/CE scheme at $Re = 1000$.

number of evaluations depending upon the stages in the scheme. Since evaluation of f^{eq} is a computationally intensive step, RK-based schemes are also computationally inefficient. Therefore, we can conclude that RK2 based OLB schemes are not particularly suitable for flows with large non-linearities.

Fig. 14 shows the stability region for the GZ scheme for $Re = 100$. From the stability-region plot, we can again see that the scheme is unstable for all $\Delta t/\tau > 2$, irrespective of the CFL number. Thus, in spite of the implicit collision approximation, large Δt could not be used in the scheme. This observation is consistent with the stability analysis presented by the authors of the scheme.

The stability regions of the AE/CE scheme are shown in Figs. 15 and 16. From the figures, it is evident that the AE/CE scheme does allow $\Delta t/\tau > 2$, but does so only at lower CFL number ($CFL < 0.2$). As Re increases, however, the explicit collision approximation becomes inadequate, and hence the Δt has to be smaller.

Finally, Fig. 17 and 18 show the stability regions for the BKG scheme at $Re = 400$ and 1000. We can observe that the simulations are stable at $\Delta t/\tau$ as high as 30, and that at high CFL numbers. The trend also does not deteriorate with increasing Re in the range that

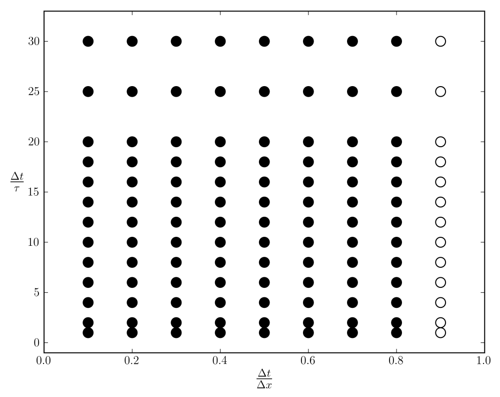


Figure 17: Stability region for the BKG scheme at $Re = 400$.

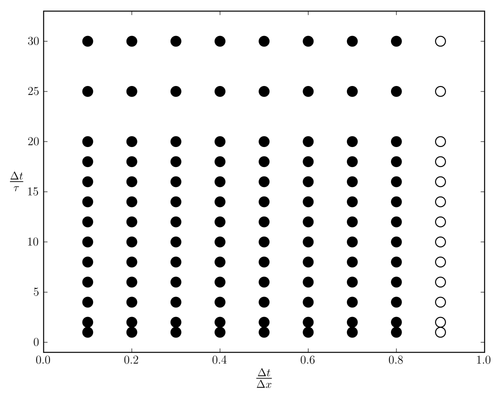


Figure 18: Stability region for the BKG scheme at $Re = 1000$.

we have tested. This demonstrates the unconditional collision stability of the BKG scheme, with Δt restricted only by the local CFL number due to explicit advection.

The BKG scheme is also computationally more *efficient* than the comparable AE/CI scheme. This is because whereas a predictor-corrector type of scheme is needed for the implicit collision approximation in the AE/CI scheme, a simple variable transformation given by Eq. (4.7) is required in the BKG scheme. Additionally, the BKG scheme, as with all advection-explicit OLB schemes, retains the data-locality feature of the LB method, and hence can be easily parallelized.

Finally, it is important to note that we have discussed only the numerical stability of different OLB schemes in terms of maximum allowable Δt. However, as for all explicit advection schemes, including the BKG scheme, much smaller values of $\Delta \boldsymbol{x}$ and Δt are required to get accurate solutions.

4.6 DISCUSSION

In this chapter, several explicit OLB schemes were analyzed and compared by implementing them for benchmark flow problems. The following conclusions can be drawn from the observations [92]:

1. The characteristics-based OLB schemes provide higher numerical stability compared to RK-based schemes.
2. In characteristics based-schemes, the $\Delta t < \tau$ constraint no longer applies, even at high Re.
3. The scheme proposed by Bardow *et al.* is stable over a much wider range of simulation parameters ($\Delta t/\tau$, $\Delta t/\Delta x$) compared to other similar characteristics-based OLB schemes for the problems tested in this work. The BKG scheme also retains the simple explicit form of the LB method, while providing unconditional collision-stability.

These conclusions indicate that the BKG scheme provides the most stable and efficient explicit time-marching scheme, which can be extended to flow problems with FV or FD discretization. This scheme, in theory, can also be adopted for thermal problems with both off- and on-lattice discrete-velocity sets [10]. There are, of course, disadvantages to using an off-lattice scheme like the BKG scheme instead of the stream-collide scheme. Some of them are:

1. The *exact* advection operation (stream-collide) is lost by using finite-difference schemes. This leads to addition of numerical diffusion to the solution and thus reduces the overall accuracy. This is a major disadvantage, especially for simulating turbulent flows.
2. The Lagrangian perspective of the LB method, where particles stream from one node to another and equilibrate (collide) at the nodes is lost. In off-lattice schemes, we no longer take a particle view. Instead, we have a set of ODEs (nine, in case of D2Q9) to be solved at each node. Thus, the algorithmic simplicity and ease of programming of the stream-collide-scheme is lost by using a off-lattice scheme.
3. Off-lattice are computationally more expensive than LBGK due the additional need to calculate the spatial gradients of distributions.

4. While LBGK has unconditional linear stability and conditional non-linear stability, off-lattice schemes have conditional linear and non-linear stability.

5.0 VALIDATION OF THERMAL MODEL

In Chap. 4, various alternate numerical schemes for the temporal evolution of the distributions were presented and analyzed. In this chapter, we return to the original topic of the development of thermal LB models. Previously, we showed that the new Hermite polynomial-based LB formulation presents a kinetically consistent way to achieve full energy conservation for both weakly and fully compressible flow cases. However, before we present the numerical validation for it, for the sake of completeness and proper context, a brief review and qualitative analysis of the existing thermal LB models is presented in Secs. 5.1 and 5.2. Thereafter, in Secs. 5.3 and 5.4, we present the numerical validation of the thermal LB models developed in Chaps. 2 and 3. More specifically, we combine the BKG-type off-lattice Boltzmann scheme and a higher-order lattice (D2Q37, in particular) to simulate benchmark thermal flows. The complexity is added incrementally as we move from the off-lattice, isothermal simulations with the D2Q9 lattice, as presented in Chap. 4, to an off-lattice, isothermal simulations with the D2Q37 lattice, presented in Sec. 5.3, finally to the off-lattice thermal simulation with the D2Q37 lattice in Sec. 5.4.

5.1 INTRODUCTION

Conventional LB models simulate incompressible flows by solving the (weakly) compressible, isothermal form of the Navier-Stokes equation at small Mach numbers. When the Mach number is small, temperature and density fluctuations are $\mathcal{O}(Ma^2)$, and hence the flow is approximately isothermal and incompressible. This modeling error associated with this technique is termed as the *compressibility error*. Although the conventional LB method has proven to be largely successful in solving flow problems involving many complex flow phenomena, most of the LB models are isothermal (athermal) in nature. Thermal LB (TLB) models, on the other hand, have not been as successful. By a thermal model, we mean a kinetically-consistent discrete model that solves for the true temperature evolution of a system along with ensuring full conservation of energy and a correct equation of state. In addition, the discrete model should all incorporate of the above without introducing any physical or numerical artifacts. Several existing TLB models are hampered by *numerical instabilities and limited parameter range*. In the following paragraphs, we present some plausible, qualitative reasons for the numerical instabilities that arise in thermal LB models.

It can be argued that a fluid flow is rarely isothermal in nature, even in the case of low-speed flows. Consider, for example, the case of a fluid flow between two adiabatic parallel plates. Due to the viscous conversion of mechanical energy into thermal energy, temperature will vary spatially; and this temperature variation attains a minimum at the channel symmetry axis, where the local speed is at a maximum. The temperature variation can be very small in many cases, but it increases with the local average speed and the fluid's Prandtl number. In the LB method, the temperature at a given node is related to the expected value, E, of the lattice-particle's kinetic energy; and the temperature variation is already captured, since E varies from node to node in accordance to the local macroscopic velocity. However, in the athermal LB model (D2Q9-based models), the spatial non-equilibrium arising from the thermal gradients is not captured *numerically*, in terms of heat (energy) flow, since athermal models are not formulated for describing this energy transfer. Thus, temperature gradients in athermal LB simulation cannot be avoided-they can only be kept small. When the gradients are not accounted for through heat transfer (ensuring energy conservation, in essence), they can, possibly, become sources of instabilities in the numerical scheme. It can be conjectured that the numerical stability issues in other simulations, such as multiphase flow, can be attributed to the lack of energy conservation in the athermal model.

On the other hand, Luo and his co-workers argued that the lack of energy conservation may not be the only fundamental cause of instabilities in thermal LB models [67]. They argued that the single-relaxation time (BGK) collision model may itself be the source of the instabilities [36], and proposed the multiple relaxation time (MRT) model to address them. The MRT model can be considered as an extension of the linearized collision operator discussed briefly in Chap. 2. However, the main aim of MRT models was to address stability issues when the local flow velocity increases, and not thermal LB in itself. Some authors have also argued that the cause of instability comes from a numerical analysis perspective by pointing towards the tight coupling between the velocity space and the physical space for the advection scheme leading to $CFL = \boldsymbol{\xi}_i \Delta t / \Delta x = 1$. From a more fundamental (statistical mechanics) point of view, the origin of the reduced stability of thermal lattice Boltzmann models was attributed to the lack of a global *H-theorem* [7, 117]. The H-theorem aims to links the reversible processes at the microscale to the irreversible processes at the macroscale. Although the validity of the H-theorem is still being debated, compliance with the H-theorem is often considered as a requirement for numerical stability in computational physics [113]. Thus, in spite of these works, to the best of the author's knowledge, a consensus still has not been reached on the fundamental cause of the numerical instabilities of the thermal LB models. In the absence of a general thermal LB model that models all types of flows (incompressible and compressible), several thermal LB models have been developed that are applicable for specific situations. An overview of them is presented below.

5.2 EXISTING THERMAL LB MODELS

This section describes, in brief, the previous attempts made at developing thermal lattice Boltzmann models. The thermal LB models that have been proposed can be broadly classified as: the passive-scalar (including a hybrid variation) model, the double-distribution, and the multispeed models[1]. Only, the essential features of these models are presented below. A detailed survey of the proposed thermal models can be found in [47] (N.B. Chapter 5).

[1]Thermal LB can also be classified into energy conserving and non-energy conserving models. While the double distribution and most multispeed thermal LB models are, in a limited sense, energy conserving, passive scalar is not.

5.2.1 Passive Scalar Model

The passive scalar (PS) model is the most popular and straightforward approach to implementing a thermal LB model. This method is based on an assumption that the macroscopic temperature field satisfies an evolution equation that is the same as that of a 'passive-scalar' In this context, a passive scalar implies a variable (scalar) that is just advected by the flow, but *does not* affect the flow (density and velocity) by itself, i.e., $\theta(\mathbf{x}, t) \leftarrow \boldsymbol{u}(\boldsymbol{x}, t)$. Moreover, by assumption, there are no sources or sinks for the temperature, and it has no effects on the fluid material properties. This is a reasonable assumption in the case of incompressible single-phase flows, where the viscous heat dissipation and the compression work are often negligible. Physically, the passive-scalar approximation is applicable not just for temperature, but for any conserved scalar, for example, it can be used for modeling concentration of trace species (pollutants, tracers, chemical species) etc. In fact, the passive-scalar originated from the more general multicomponent flow LB model[102].

In terms of the algorithm, two distributions are used in the passive scalar model: a density distribution, f_i, for density and momentum fields, which satisfy the Navier-Stokes equation; and a thermal distribution, h_i, for the temperature field which satisfies the following *advection-diffusion* type equation:

$$\frac{\partial T}{\partial t} + \boldsymbol{u} \cdot \nabla T = \alpha \nabla^2 T, \tag{5.1}$$

where α is the thermal diffusivity. The evolution of h_i is identical to that of f_i:

$$h_i(x + \Delta \boldsymbol{x}, t, t + \Delta t) = h_i(\boldsymbol{x}, t) - \frac{1}{\tau_T}(h_i(\boldsymbol{x}, t) - h_i^{eq}(\boldsymbol{x}, t), \tag{5.2}$$

where τ_T is a dimensionless thermal relaxation time corresponding to the rate at which the temperature dynamics approach their equilibrium h_i^{eq}. Numerically, Eq. (5.2) is solved on the same numerical grid as that of the density distribution. h_i^{eq} is the second-order thermal equilibrium distribution:

$$h_i^{eq} = w_i T \left[1 + \boldsymbol{\xi}_i \cdot \boldsymbol{u} + \frac{1}{2}(\boldsymbol{\xi}_i \cdot \boldsymbol{u})^2 - \frac{1}{2}\boldsymbol{u}^2\right]. \tag{5.3}$$

Analogous to density, the temperature at a node is the defined as the sum of all the thermal distributions at a node:

$$T = \sum_i h_i, \tag{5.4}$$

and the thermal diffusivity is related to the thermal relaxation time through (in lattice units):

$$\alpha = \frac{1}{3}(\tau_T - \frac{1}{2}). \tag{5.5}$$

It should be noted that for a numerically stable and accurate simulation using the passive scalar method, thermal diffusivity cannot be faster than the momentum diffusivity, ν, leading to the restriction on the Prandtl number that $Pr = \frac{\nu}{\alpha} \leq 1$. Since passive-scalar method was not formulated to ensure energy conservation, it serves as a good approximation only for thermal flows involving liquids and low-speed incompressible gas flows, where the Boussinesq approximation is valid (as discussed in Sec. 5.4.2).

5.2.2 Double Distribution Model

The work of He and Luo [55] showed that the isothermal lattice Boltzmann equation can be derived directly from the Boltzmann equation through a suitable phase and space-time discretization. Using the same principle, He *et al.* [54] proposed the *double-distribution function* model (DDF), or multi-lattice thermal model, where the thermal effects in a flow can be modeled by *independently* discretizing the continuous Boltzmann equation for the *internal energy distribution*, in addition to the density distribution. Consequently, there are *two* separate distributions and two separate evolution equations: one equation for the density distribution, which governs the evolution of the density and velocity fields; and the other for the internal energy distribution, which governs the evolution of the temperature field.

It has been shown that this model is simple and applicable to problems with different Prandtl numbers. More importantly, this model requires a low-order moment of the internal energy distribution, and its numerical stability is similar to that of the passive-scalar method. On the other hand, because the new scheme directly simulates the evolution of the internal energy, the viscous heat dissipation and compression work done by the pressure can be naturally incorporated, unlike the passive scalar model. Lattice velocities for this method are the same as in the D2Q9 model, and the both the equilibrium distributions are expanded to second-order in local flow velocity. A major shortcoming of this model, from a multiphase and fully compressible flows perspective, is that the equation of state is independent of the temperature. This deficiency means that, like the passive scalar model, the DDF model is suitable for thermal flows where Boussinesq approximations can be made or flows where Mach number is small.

5.2.3 Multispeed Model

Multispeed thermal models based on the LGA theory were built by an extension of the isothermal LB formulation of Qian *et al.* [90]. In general, multispeed models have more than nine velocities in the lattice in two dimensions, involve the equilibrium distribution expanded to the third order or more $\mathcal{O}(\boldsymbol{u}^3)$, and attempt to ensure to full energy conservation. The first multispeed thermal model was proposed by Alexander *et al.* [5], who extended the second-order equilibrium distribution of Qian *et al.* to a third-order model, along with a thirteen-velocity discrete lattice. Thereafter, McNamara and Alder found a set of thirteen and twenty-six moment restrictions that the equilibrium distribution must satisfy in order to retrieve the correct advection-diffusion macroscopic equations in two and three dimensions, respectively [75]. Some nonlinear artifacts (deviations) in the momentum and energy equations present in the model of Alexander *et al.* were observed by Chen *et al.* [30], who corrected them by introducing a fourth-order polynomial expansion of the equilibrium distribution by fitting adjustable parameters. They used combinations of square lattices for satisfying the restrictions imposed by the Chapman-Enskog analysis and found a sixteen velocity lattice in two dimensions and a lattice with forty-one velocities in three dimensions. McNamara *et al.* further attempted to improve the numerical stability of their thermal model [75] by adding numerical diffusion via a finite-difference based Lax-Wendroff type advection scheme, instead of the standard first-order upwind (streaming scheme) [76]. Vahala *et al.* proposed velocity lattices with better and higher order symmetry that do not coincide with a physical lattice (non-space filling lattices) [121].

Unlike the unconditionally stability of the isothermal LB method, all of the above thermal models have stability issues and a limited parameter range. The LGA-based thermal models were devised in a top-down fashion, where both the discrete velocities and functional form of the equilibrium distribution were set *a priori*, and coefficient matching was done through the Chapman-Enskog expansion to obtain the required thermohydrodynamics. The coefficient-matching procedure is somewhat empirical, and therefore does not have a fundamental mathematical or physical footing in kinetic theory. That is, there is *no formal link connecting the multispeed thermal LB models to the Boltzmann equation.*

It should be emphasized that some of the above methods (and their variations) present an accurate and efficient method to incorporate heat transfer and thermal effects for low-speed, moderate Rayleigh number, incompressible flows with the Boussinesq approximation, with varying Prandtl numbers. However, when these models are extended for use with multiphase or multicomponent flows, they become numerically unstable, or their applicability range is limited. Moreover, the temperature described in the above models does not coincide with *true temperature* in the sense of thermodynamics and statistical mechanics (kinetic consistency). Hence, as mentioned before in Chap. 1, we formulated the Hermite-based thermal LB model, which provides us a kinetically-consistent thermal LB model that can be extended to a wider parameter range and to other complex flows. The validation of the thermal models that were developed in Chaps. 2 and 3 is presented in the following sections.

5.3 OFF-LATTICE SCHEMES WITH HIGHER-ORDER LATTICES (ISOTHERMAL CASE)

As mentioned before, due to the ease with which wall-boundaries can be handled with finite-difference discretization, we employ the off-lattice temporal evolution of distributions to model thermal flows, instead of the standard stream-collide scheme. However, before extending the off-lattice scheme to thermal flows, we first validate the use of the BKG scheme, in particular, with the D2Q39 velocity-set, for isothermal flows. The numerical experiments involved in this validation are presented below.

The numerical testing and validation performed previously was done in order to compare and analyze the *numerical stability* of the various off-lattice schemes. The question of *numerical accuracy*, especially in conjunction with higher-order velocity-sets, was not addressed. The numerical accuracy of a simulation depends on the effective viscosity of the OLB scheme used. The effective (overall) viscosity can be considered as a combination of the actual physical viscosity and the artifact of numerical viscosity. While, the physical viscosity of the fluid is controlled by the relaxation time τ, through the relation $\nu = c_s^2\tau$, additional non-physical (numerical) viscosity is effected on the simulations due to the finite space-time discretization. Although, numerical viscosity marginally adds to the stability of the scheme, it nonetheless decreases the accuracy of the simulation. On the other hand, in the stream-collide type LB scheme, the numerical viscosity can be quantified *a priori* through the Chapman-Enskog procedure, and hence can compensated (corrected) by incorporating it in the viscosity-relaxation-time relationship. Due to this fact, the conventional LB method is said to have exact advection (diffusion and dispersion free). However, for the off-lattice temporal schemes and different spatial discretizations (FD or FV), it is difficult to obtain such compensating factors, beforehand. Therefore, it is important to *numerically* characterize and quantify the effects of finite discretization (through parameters) on the overall accuracy of the scheme.

In the following section, we investigate the effects of discretization parameters $(\Delta\boldsymbol{x}, \Delta t)$ of numerical accuracy resulting from using the BKG scheme along with the D2Q39 lattice. This investigation will allow us to find the correct $\nu - \tau - \lambda$ relationship which would be useful for the subsequent validation tests. It also provides us a useful test to implement the BKG scheme with the D2Q39 lattice (a first to the author's knowledge); although it has been used with a entropic lattice Boltzmann based higher-order velocity-set [10].

5.3.0.1 Decay of a shear-wave In order to check and quantify the effects of the discretization parameters ($\Delta\boldsymbol{x}$ and Δt) on the effective viscosity, we model the attenuation (decay) of a two dimensional shear-wave, whose wavelength equals the system size L. The shear-wave decay problem is often used to check the effective (overall) viscosity of a numerical scheme since exact closed-form solutions are available for both velocity and effective viscosity. As the initial conditions, the shear wave is initialized with the following velocity profile:

$$\begin{cases} u_x(x,y)|_{t=0} = & u_0 \sin(2\pi y) \\ u_y(x,y)|_{t=0} = & 0 \end{cases} \tag{5.6}$$

Assuming $u_x = u_x(y,t)$, and negligible pressure difference, the decay of the shear wave is governed by following simplified isothermal Navier-Stokes momentum equation:

$$\frac{\partial u_x}{\partial t} - \nu \frac{\partial^2 u_x}{\partial y^2} = 0 \tag{5.7}$$

where ν is the physical viscosity. Given the initial velocity condition per Eq. (5.6), an analytical solution of Eq. (5.7), at time t, can be calculated in a closed-form as:

$$u_x(y,t) = u_0 \exp(-k^2 \nu t) \sin(ky) \tag{5.8}$$

When the decay of the shear-wave is solved numerically on a grid, the effective value of the kinematic viscosity ν_e observed in the simulation results can be obtained from:

$$\nu_e = \frac{1}{k^2 t} \log \frac{\mathcal{A}(0)}{\mathcal{A}(t)}, \tag{5.9}$$

where $\mathcal{A}(t)$ is the Fourier coefficient given by [110]:

$$\mathcal{A}(t) = \int_0^L u_x(y,t) \sin(ky). \tag{5.10}$$

The objective of performing this investigation is to compute ν_e through simulations which can then be compared to the physical velocity $\nu = c_s^2 \tau$. This will provide us a measure of the difference between the actual and effective viscosity. From the simulation results, $\mathcal{A}(t)$ can be computed with the knowledge of the velocity profile $u_x(y,t)$, and therefore ν_e can be computed per Eq. (5.9)[2]. The apparent viscosity is a function of the numerical scheme (BKG scheme in this case) used to solve the shear-wave decay, and thus a function of Δt and $\Delta \boldsymbol{x}$. That is, for a finite $\Delta \boldsymbol{x}$ and Δt, ν_e has contribution from both the physical and numerical viscosity. In an ideal numerical simulation, of course, the apparent viscosity will be equal to the physical viscosity.

5.3.0.2 Simulation Setup We now test the accuracy of the off-lattice BKG scheme along with D2Q37 velocity set for the isothermal shear wave decay problem ($\theta = 1$) described above. The discrete velocities of D2Q37 are as listed in Appx. B and the discrete, fourth-order equilibrium is per Eq. (3.11). It should be noted that, in the BKG scheme, the relaxation time is related to the modified relaxation time through:

$$\lambda = \tau + \frac{\Delta t}{2}. \tag{5.11}$$

In the simulations, $u_0 = 0.01$, so that incompressible flow limit is ensured, and τ is kept constant at $\tau = 0.01$. The physical viscosity is calculated on the basis of τ and not λ. Also note that here we have taken $c_s = 1$, and therefore $\nu = c_s^2 \tau = 0.01$. The system size is assumed to $L = H = 1$, the magnitude of the wave-vector is $k = 2\pi/L$, and periodic boundary conditions are applied in both the x- and y- directions.

5.3.0.3 Results For the above mentioned flow parameters, Fig. 19 shows the time history of the decay of the shear wave, expressed through the velocity profile of $u_x(y,t)$. An exponential decay, as predicted by the analytical solution (Eq. (5.8)), can be observed. More importantly, this solution provides a validation for the use of an off-lattice (BKG) scheme with the D2Q37 lattice.

We now discuss the accuracy estimates for the BKG scheme. Fig. 20 plots the effective velocity, and the percentage error between the effective and actual viscosity, a for particular

[2]The matplotlib's (Python) `simps` functionality can be used to evaluate the integral in Eq. (5.10), given that $u_x(y,t)$ is available.

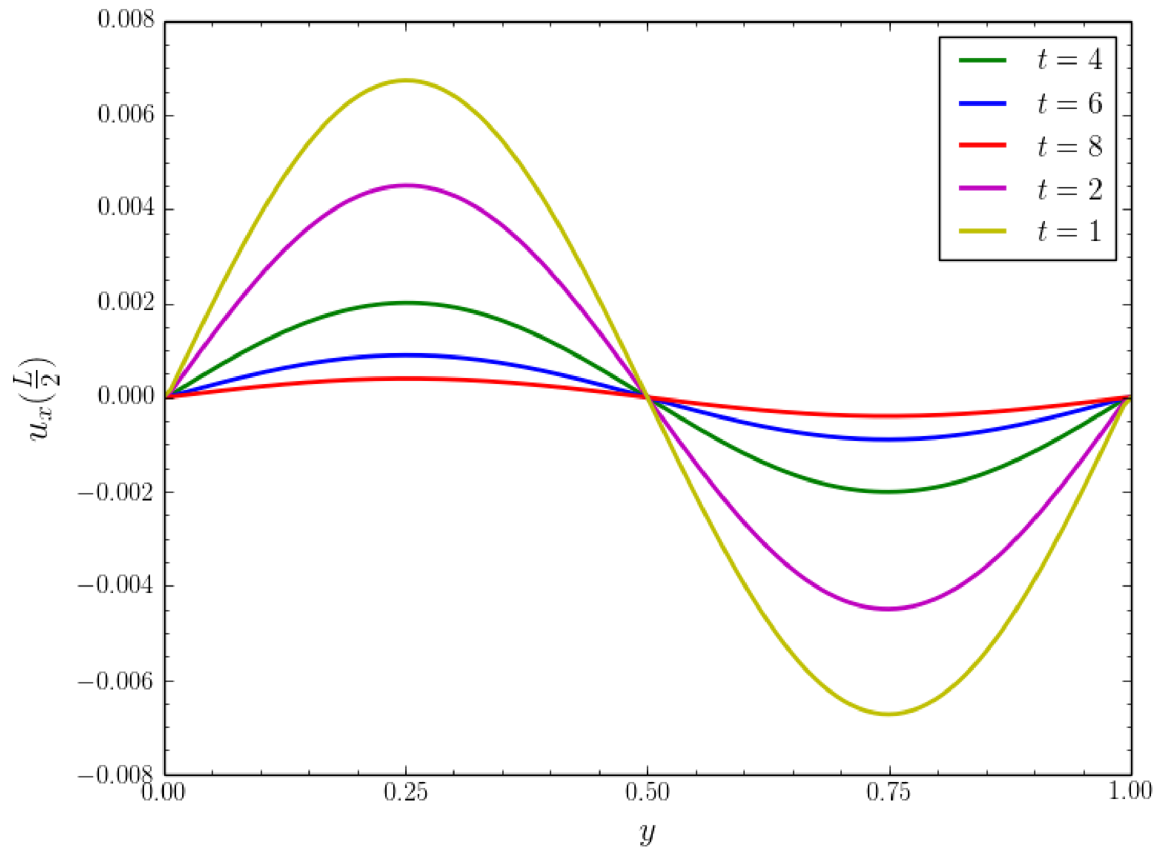


Figure 19: Time history of decay of shear wave. Results are obtained on a 100×100 grid with $\Delta t = \tau = 0.01$.

size of $\Delta \boldsymbol{x}$. We can observe that for a given grid size, reducing Δt decreases the difference between the two viscosities, thus enabling better accuracy in the simulations.

Similarly, Tab. 3 shows the percentage difference between the effective and physical viscosity for fixed size of Δt. Once again, we can infer that for a given Δt, reducing $\Delta \boldsymbol{x}$ gives more accurate simulations. These observations are consistent with explicit schemes for solving PDEs, where much smaller values of Δt and $\Delta \boldsymbol{x}$ (smaller than that required for stability) are required to obtain accurate solutions.

From the shear-wave decay problem we can make the following inferences:

1. Reynolds number of the flow should be calculated on the basis of the relaxation time, τ and *not* the modified relaxation time.
2. As with all explicit advection schemes, numerical diffusion (errors) tends to zero with both Δt, $\Delta \boldsymbol{x} \rightarrow 0$ as indicated in figures and tables. Per Eq. (5.11), it is also obvious that as $\Delta t \rightarrow 0$, the overall viscosity, $\nu_e = c_s^2 \lambda$ tends to the physical viscosity $\nu = c_s^2 \tau$. This also highlights the unavoidable consequence of OLB schemes, namely the numerical diffusion errors, which do not exist the stream-collide type schemes. Moreover, in any explicit OLB scheme, accurate solutions are obtained at very small CFL values (< 0.1), compared to the stream-collide type scheme, where $CFL = 1$, which results in longer computational run-time for the same sized mesh.

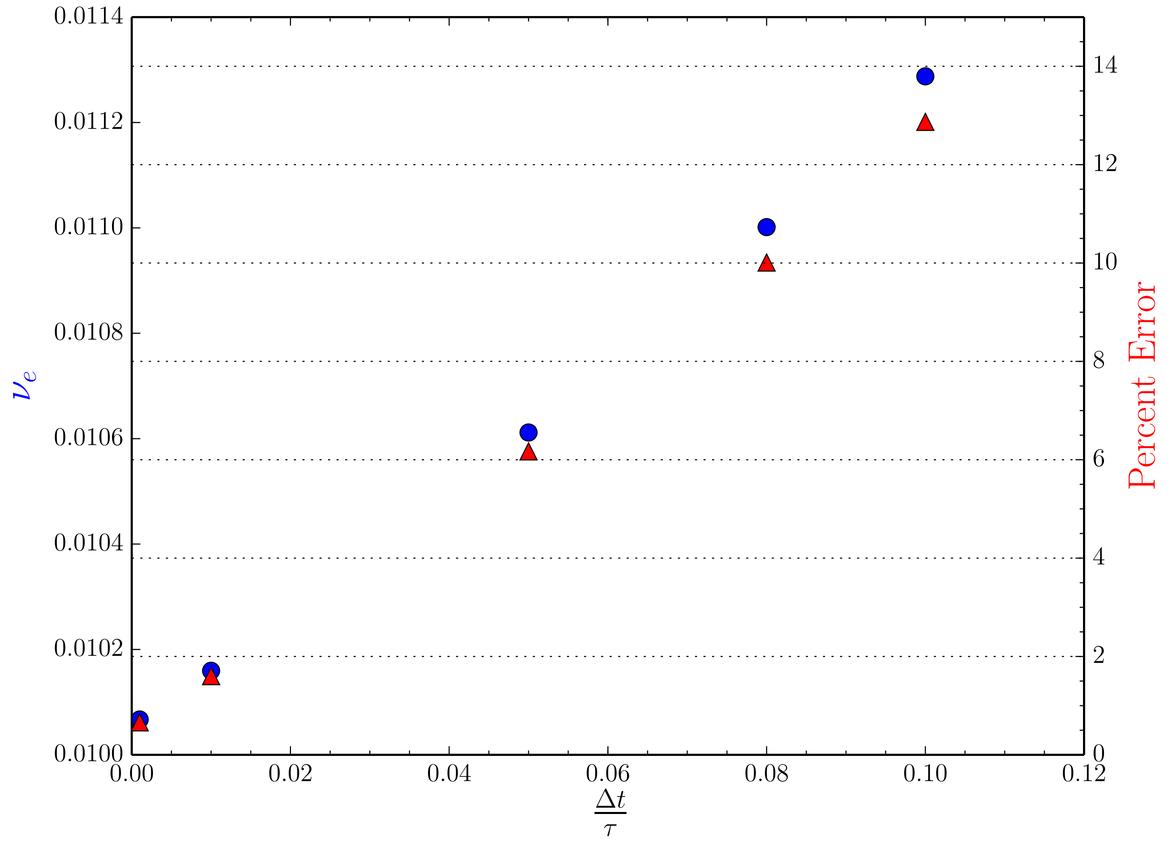


Figure 20: Values of effective viscosity, ν_e (blue dots), and percentage error between the effective and physical viscosities (red) as a function of time-step. The physical viscosity is $\nu = 0.01$ (constant).

Table 3: Percentage difference between effective and physical kinematic viscosity for various $\Delta \boldsymbol{x}$. Results are based on the D2Q37 velocity-set, $\tau = 0.01$ and $\Delta t = 0.01\tau$.

Δt	65×65	129×129	257×257
E_ν	14.5	4.6	0.04

5.4 VALIDATION OF THE THERMAL MODEL

5.4.1 Thermal Couette Flow

We now proceed towards validation of the thermal model developed in Chaps. 2 and 3. The first validation test we present is that of the thermal Couette flow. The (planar) thermal Couette flow represents the two dimensional flow of a fluid between two parallel flat plates kept at different temperatures. Fig. 21 provides a schematic of the flow. The thermal Couette flow not only provides a good benchmark test to validate the thermal model, but also the boundary scheme described in Sec. 4.4. If we assume no body forces (gravity), steady state conditions, constant material properties, no internal heat sources, and negligible pressure gradient along the channel, then the governing equation for the incompressible thermal Couette flow of an ideal gas can be written as:

$$\begin{cases} \mu \frac{\partial u_x^2}{\partial y^2} & = 0 \\ \frac{k}{\mu}\frac{\partial^2 T}{\partial y^2} + \frac{\partial}{\partial y}\left(u\frac{\partial u_x}{\partial y}\right) & = 0 \end{cases} \tag{5.12}$$

As shown in Fig. 21, if we apply the following boundary conditions:

$$\begin{cases} \boldsymbol{u}(\boldsymbol{x},t)|_{y=0} = 0 \\ u_x(\boldsymbol{x},t)|_{y=H} = U_t, u_x(\boldsymbol{x},t)|_{y=H} = 0 \\ \theta(\boldsymbol{x},t)|_{y=0} = \theta_b \\ \theta(\boldsymbol{x},t)|_{y=H} = \theta_t = \theta_b + \Delta\theta \end{cases} \tag{5.13}$$

the temperature profile along the height satisfies the following analytical solution of Eq. (5.12):

$$\frac{\theta - \theta_b}{\theta_t - \theta_b} = \frac{y}{H} + \frac{PrEc}{2}\frac{y}{H}\left(1 - \frac{y}{H}\right), \tag{5.14}$$

where θ_t and θ_b are the top and the bottom wall temperature, respectively, y is the distance from the bottom wall, H is the height of the channel, $Pr = \nu/\alpha$ is the Prandtl number, and $Ec = U_t^2/c_v(\theta_t - \theta_b)$ is the Eckert number. The combined effects of the Prandtl and Eckert number can be parameterized through the Brinkman number: $Br = PrEc$.

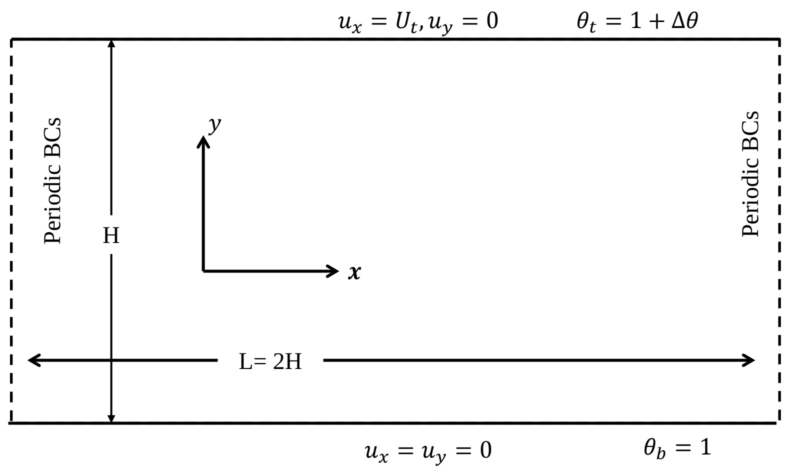


Figure 21: Schematic and simulation setup for thermal Couette flow

The accuracy of the thermal model is evaluated by carrying out simulations for the thermal Couette flow at different Brinkman numbers. The D2Q37 lattice, along with the fourth-order form of the equilibrium distribution function (Eq. (3.11)), is used for all of the cases. Since the BGK collision model is used, $Pr = 1$. For all the simulations, we use the following simulation parameters: $\theta_b = 1.0$, $\theta_t = 1.0001$, $H = 1$, and $\tau = 5 \times 10^{-3}$. The mesh size is 257×257 and the time step is kept small, $(\Delta t = 0.05\tau)$, in order to obtain accurate results. Keeping the temperature gradient and the above parameters fixed, the top boundary speed, U_t, is varied to obtain different Br simulations. Periodic boundary condition is enforced in the x-direction and the extrapolation boundary scheme is used to prescribe the temperature and no-slip velocity on the walls.

Fig. 22 shows the solution obtained using the D2Q39 lattice for various Brinkman number. Physically, increasing the Brinkman number increases the viscous dissipation in the fluid. We observe that the simulated temperature solution agrees with the predicted analytical one for the range of Brinkman numbers tested here. Even near the boundaries the profiles match closely, validating the boundary treatment used. It is clear that the D2Q39-based thermal model is capable of simulating viscous dissipation over this wide range.

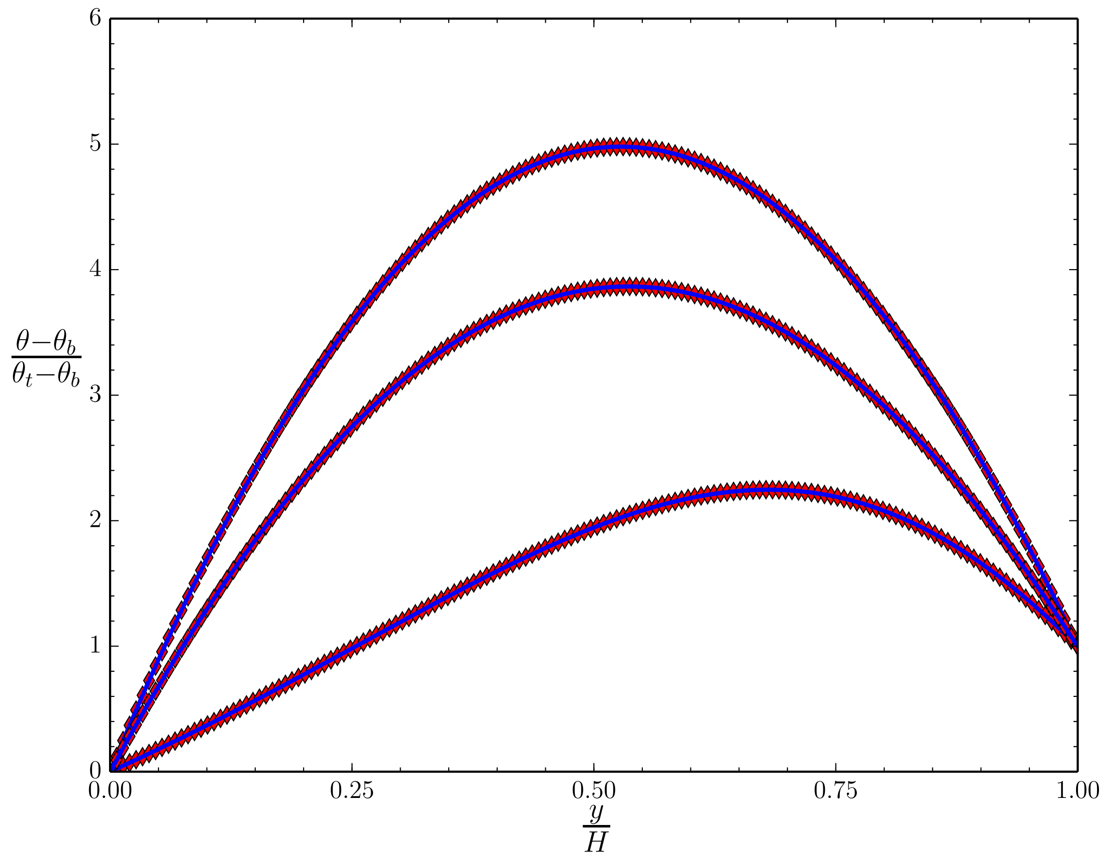


Figure 22: Temperature profiles in thermal Couette flow for various Brinkman numbers. The solid lines indicate an analytical solution and markers ($\diamond$) indicate the simulated solutions. Solutions are obtained on a 257×257 grid.

5.4.2 Rayleigh-Benard Convection

The next validation test we perform is the modeling of Rayleigh-Benard convection. This is a classical benchmark test for validating thermal models. A viscous fluid is enclosed between two parallel stationary walls, such that the fluid is heated from the bottom, and the fluid experiences the gravity force, as shown in Fig. 23. Since the fluid near the bottom wall is hotter, it has a smaller density, while the fluid near the top wall is colder and hence has a larger density. Gravity drives the colder and heavier fluid to the bottom, but this is opposed by the viscosity of the fluid. A balance between these two forces keeps the system in equilibrium and the heat transfer is entirely due to diffusion (conduction). As the temperature difference is raised above a certain threshold, the buoyancy force increases, the static conductive state becomes unstable, and convection occurs abruptly. The non-dimensional number that characterizes the flow is the Rayleigh number, Ra,

$$Ra = \frac{g\beta\Delta\theta H^3}{\nu\kappa} = Pr \times \frac{g\beta\Delta\theta H^3}{\nu^2} \tag{5.15}$$

where g is the gravity acceleration, $\Delta\theta$ is the wall temperature difference, H is the vertical distance between the walls, and ν is the kinematic viscosity of the fluid. The ratio on the right hand side is called the local (non-dimensional) Grashof number, Gr. The Grashof number can be viewed as the ratio of the buoyancy and viscous forces acting on a unit mass of the fluid; while the Rayleigh number characterizes the combined effects of the flow parameters and the fluid material properties. When the Rayleigh number is below a critical value for that fluid, heat transfer is primarily in the form of conduction; when it exceeds a critical value, heat transfer is primarily in the form of convection.

5.4.2.1 Boussinesq Approximation In order to simulate the Rayleigh-Benard convection for the widest possible parameter range (Ra) and without any approximations, a thermal model should ensure total energy conservation, even for compressible flows. However, for a wide variety of engineering flows, the modeling can be simplified by making the incompressibility assumption (density is independent of pressure), even for flows driven by external body forces (gravity for example). Under this assumption, the body force ($\rho\boldsymbol{G}$)

entering the momentum equation can be modeled using the so-called *Boussinesq approximation* [111, 124]. Per the Boussinesq approximation, the density field can be treated as a constant in both the continuity and the momentum equations, except in the body force (gravity) term. That is, $\rho(\boldsymbol{x}, t)$ is considered to be independent of the pressure (i.e., incompressibility is assumed), and to depend *linearly* on the temperature $\theta(\boldsymbol{x}, t)$, such that $\rho\boldsymbol{G} = \rho_{ref}\beta g(\theta(\boldsymbol{x}, t) - \theta_m)\boldsymbol{j}$. With this approximation, the Navier-Stokes equation, can be written as [66]:

$$
\begin{aligned}
\nabla \cdot \boldsymbol{u} &= 0 && (5.16)\\
\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} &= -\frac{1}{\rho_{ref}}\nabla p + \nu\nabla^2\boldsymbol{u} + \beta g(\theta - \theta_{ref})\boldsymbol{j} && (5.17)\\
\frac{\mathrm{D}\theta}{\mathrm{D}t} &= \alpha\nabla^2\theta && (5.18)
\end{aligned}
$$

where ρ_{ref} is the reference (mean) density corresponding to the mean temperature field θ_{ref}. Boussinesq approximations are commonly employed in many flow models, for example, in the analysis of wave propagation in a density-stratified medium, oceanic and atmospheric flows, natural convection based ventilation systems, etc. This approximation is valid only if the Mach number of the flow is small, propagation of sound or shock waves is not considered, the vertical scale of the flow is not too large, and the temperature differences in the fluid are small. Material properties of the fluid, such as μ, k, and c_p are also assumed constant. In terms of the energy equation, heating due to viscous dissipation of energy and pressure work are negligible under the above approximations. Temperature behaves like a passive scalar being advected by the flow.

In the LB context, the Boussinesq approximation is incorporated through the forcing term. The force driving the flow is implemented using the Shan-Chen type '*shifted-equilibrium*' scheme [102, 96]. It should be noted that, here, both velocity and temperature are shifted to account for the presence of a body force. That is:

$$
\begin{aligned}
\boldsymbol{u}^{(eq)} &= \boldsymbol{u} + g\frac{\Delta t}{2} && (5.19)\\
\theta^{(eq)} &= \theta + \frac{(\Delta t)^2 g^2}{4D} && (5.20)
\end{aligned}
$$

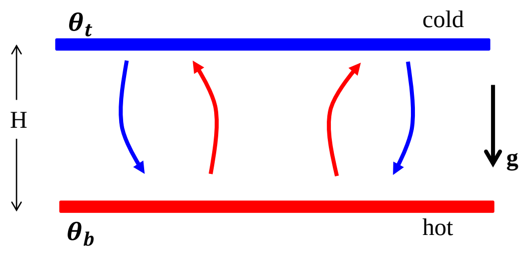


Figure 23: Schematic for Rayleigh-Benard convection

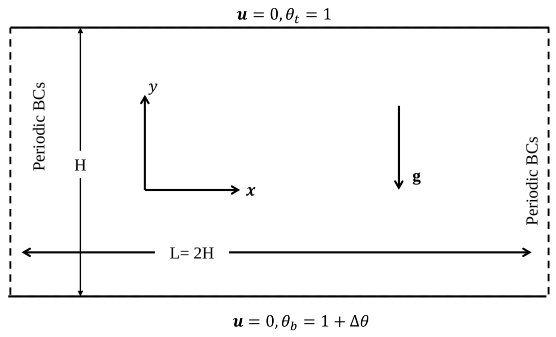


Figure 24: Simulation setup for Rayleigh-Benard convection

Here, $\boldsymbol{u}^{eq}$ and θ^{eq} are the 'shifted' velocity and temperature, respectively, which are used to compute the equilibrium distribution functions. $\boldsymbol{u}$ and θ are the usual (lattice) velocity and temperature, computed through moment summations (Eq. (3.15)).

5.4.2.2 Problem Formulation and Simulation Setup

For simulating the Rayleigh-Bernard convection, we consider two plates separated by a vertical height H, as shown in Fig. 24. Between the plates is a (ideal) fluid with $Pr = 1$. The Prandtl number is constant due to the BGK model used in the simulations. As boundary conditions for the top and bottom walls, constant temperature and no-slip velocity condition are prescribed:

$$\begin{cases} \boldsymbol{u}(\boldsymbol{x},t)|_{y=0} = \boldsymbol{u}(\boldsymbol{x},t)|_{y=H} = 0 \\ \theta(\boldsymbol{x},t)|_{y=0} = \theta_b \\ \theta(\boldsymbol{x},t)|_{y=H} = \theta_t \end{cases} \tag{5.21}$$

Periodicity is maintained in the $x-$ direction for both temperature and velocity. As an initial condition, the fluid is assumed to be at rest and temperature is linearly stratified along the height:

$$\begin{cases} u(\boldsymbol{x}, y,)|_{t=0} & = 0 \\ \theta(\boldsymbol{x}, y)|_{t=0} & = \theta_{ref} + \Delta\theta \frac{y}{H} \end{cases} \tag{5.22}$$

For the fluid under consideration (ideal gas), the coefficient of thermal expansion, β, is defined as:

$$\beta = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial \theta}\right)_p = \frac{1}{\theta}. \tag{5.23}$$

For computing the Rayleigh number, we use the expansion coefficient evaluated at a temperature $\theta_{ref} = \theta_b$. Therefore, $\beta = 1/\theta_b$. We fix the bottom and top wall temperatures to be 1.0 and 1.001, respectively, and $H = 1$. That is $\theta_{ref} = \theta_b = 1.0$ and $\theta_t = 1.001$. Once the height H and the temperature difference $(\theta_t - \theta_b)$ across the walls are fixed, the entire simulation is determined by just two parameters, τ and g. With the above simulation values, the Rayleigh number reduces to:

$$Ra = \frac{g\Delta\theta}{\nu^2}. \tag{5.24}$$

In addition, the characteristic velocity is a free parameter, which is kept small, so as to ensure the small Mach number (incompressibility) condition. Different Rayleigh number simulation results are obtained by varying τ and g. The enhancement of heat transfer due to the onset of Rayleigh-Benard convection can be described by the Nusselt number:

$$Nu = 1 + \frac{\langle u_y T\rangle H}{k\Delta\theta}, \tag{5.25}$$

where u_y is the velocity in the y direction, k is the thermal conductivity of the fluid, and $\langle\cdot\rangle$ represents the average over the whole flow domain. As before, the D2Q37 velocity set along with the fourth-order equilibrium distribution function $\mathcal{O}(\boldsymbol{u}^4, \theta)$ is employed in the simulations. For the top and bottom walls, the boundary scheme as detailed in Sec. 4.4 is used and periodic boundaries are applied in the x direction. The grid size is 257×129 for all the solutions presented here.

5.4.2.3 Results At a Rayleigh number below the critical value, Ra_c, the perturbation given to the system through $\Delta\theta$ is dissipated by the viscosity and the maximum velocity in the domain gradually reduces to zero. The heat transfer in this state is purely conductive. Above the critical Rayleigh number, convection ensues and the maximum velocity eventually reaches a finite value. For the given flow domain, the critical Rayleigh number is given by the linear stability theory and confirmed by laboratory experiments. Per the method outlined in

[100], the critical Rayleigh number is calculated by interpolating the growth rate of maximum velocity at a Rayleigh number slightly higher than Ra_c, and the decay rate at a slightly lower rates. On the grid size of 50×26 and 100×51, we observed Ra_c of 1718.45 and 1713. 71, respectively. This is close to the predicted theoretical value of 1707.76.

The steady-state temperature distribution, in the form of filled contours, at different Rayleigh numbers is shown in Fig. 25. As observed in the figure, heating from below, causes the fluid to flow upwards, while the colder fluid from the top gets pushed downwards, resulting in an increase in temperature in the central region due to mixing of the two streams. As the Rayleigh number increases, this mixing is enhanced, thereby increasing the heat transfer.

Pseudo color maps for the velocity are presented in Fig. 26. From the velocity maps, we observe that when the Rayleigh number is $2000 \leq Ra \leq 5000$, the flow converges to convection rolls which are steady (not shown here); when $5000 \leq Ra \leq 10000$, these rolls oscillate in a sinusoidal fashion instead of converging to a steady pattern. For $Ra = 20000$, the behavior of the flow becomes totally random at first but eventually takes a more structured form. Finally for $Ra > 20000$, the velocities have less structured patterns, as shown in Fig. 26.

To test the quantitative accuracy of the D2Q39 model, we plot the relationship between the Nusselt number, calculated as per Eq. (5.25), and the corresponding Rayleigh number in Fig. 27. The simulation results from Clever and Busse [33] are also included for comparison. The current model is in agreement with [33], and it also agrees with the passive-scalar, DDF, and the entropic LB based thermal models [54, 89, 100].

5.4.3 Discussion

We conclude the presentation on thermal LB by discussing the various existing models and their broad methods of formulation, as presented in the review above. While the Hermite-based LB construction, in theory, offers an elegant, consistent way to obtain a thermal model applicable for the wide range of flows (weakly compressible and compressible), it is *not computationally attractive* for purely incompressible flows (with the Boussinesq approximation).

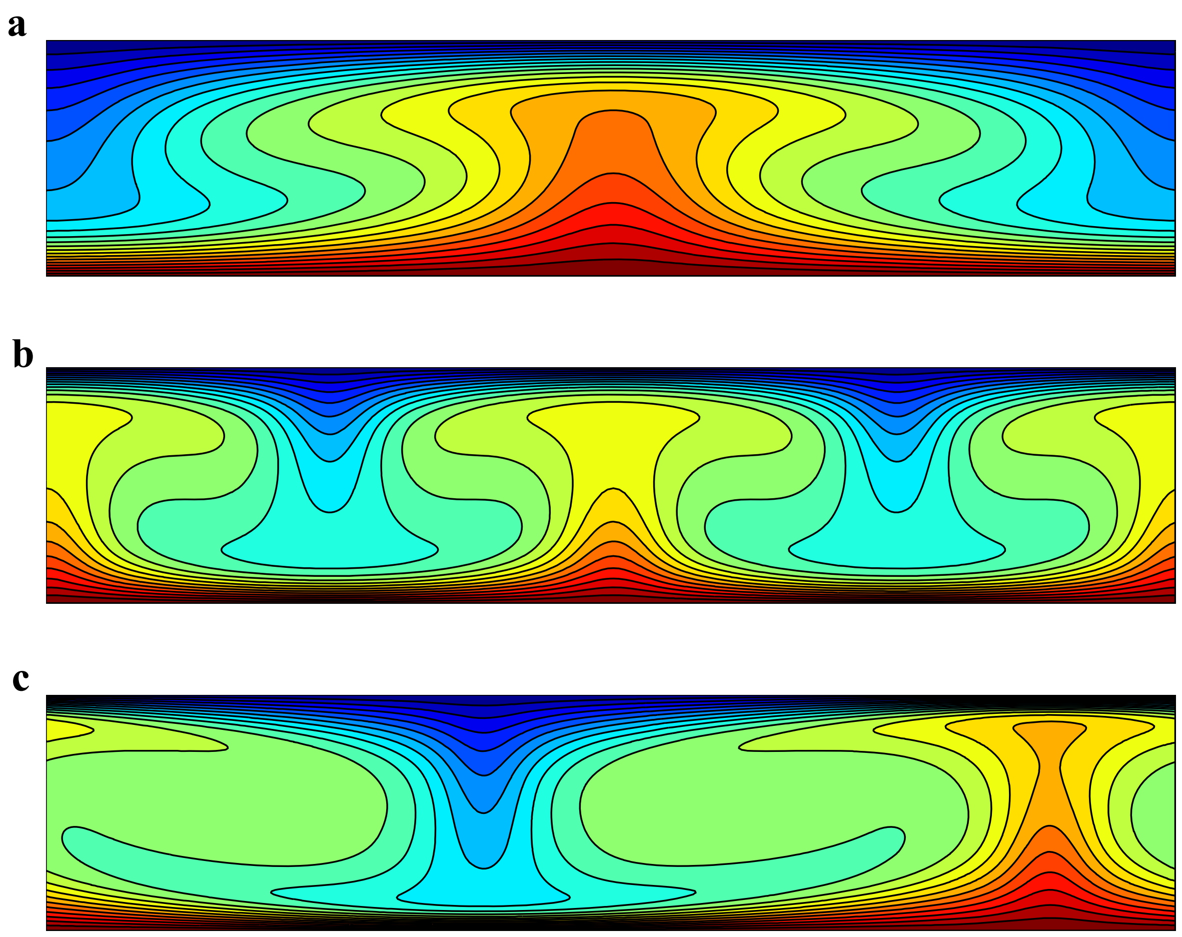


Figure 25: Filled contours for the normalized temperature $(\theta - \theta_b)/(\theta_t - \theta_b)$ for Rayleigh-Benard convection, where (**a**) $Ra = 10^4$, (**b**) $Ra = 5 \times 10^4$, and (**c**) $Ra = 10^5$. Twenty one equally distributed contours have been plotted.

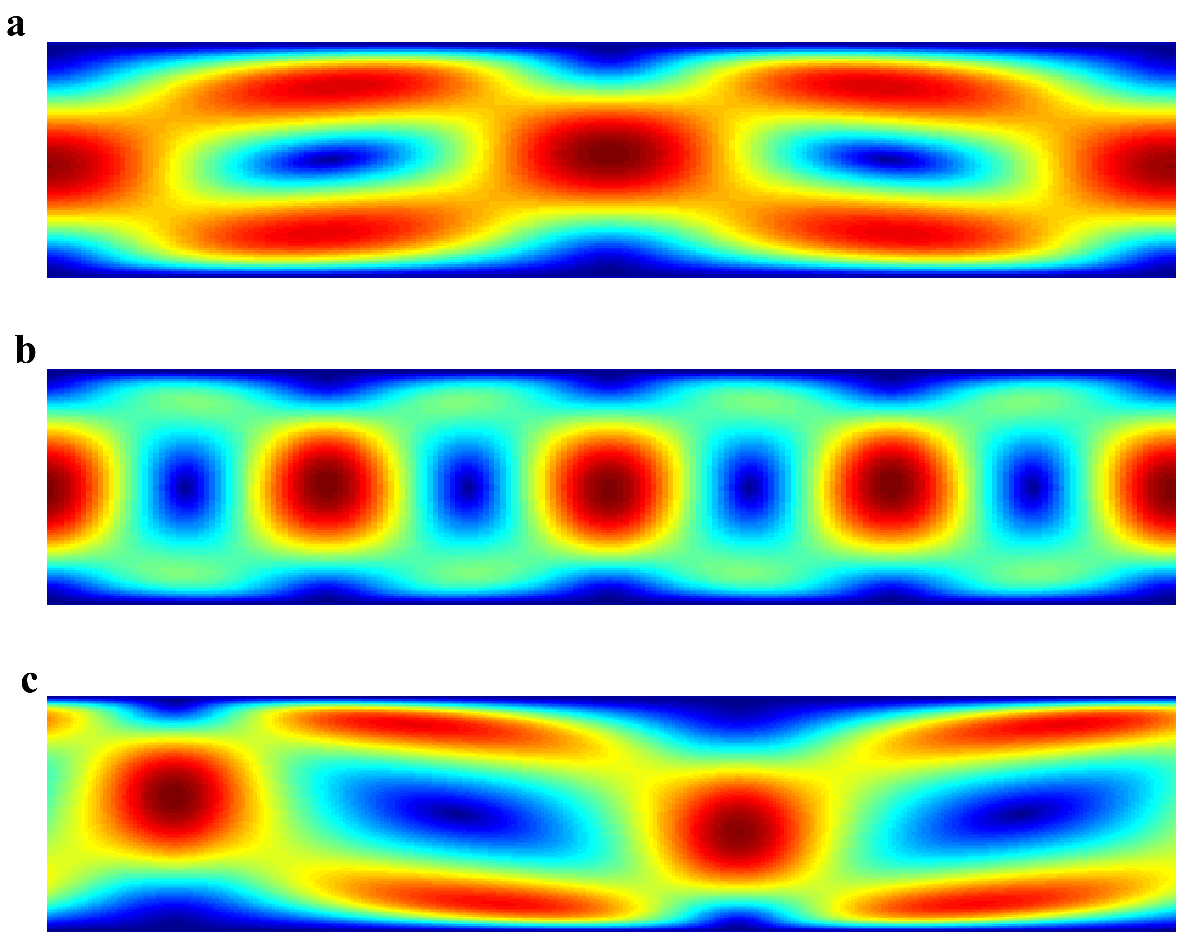


Figure 26: Psuedo color map for velocity $\boldsymbol{u} = \sqrt{u_x^2 + u_y^2}$ for Rayleigh-Benard convection, where (a) $Ra = 10^4$, (b) $Ra = 5 \times 10^4$, and (c) $Ra = 10^5$. Red indicates maximum and blue indicates minimum.

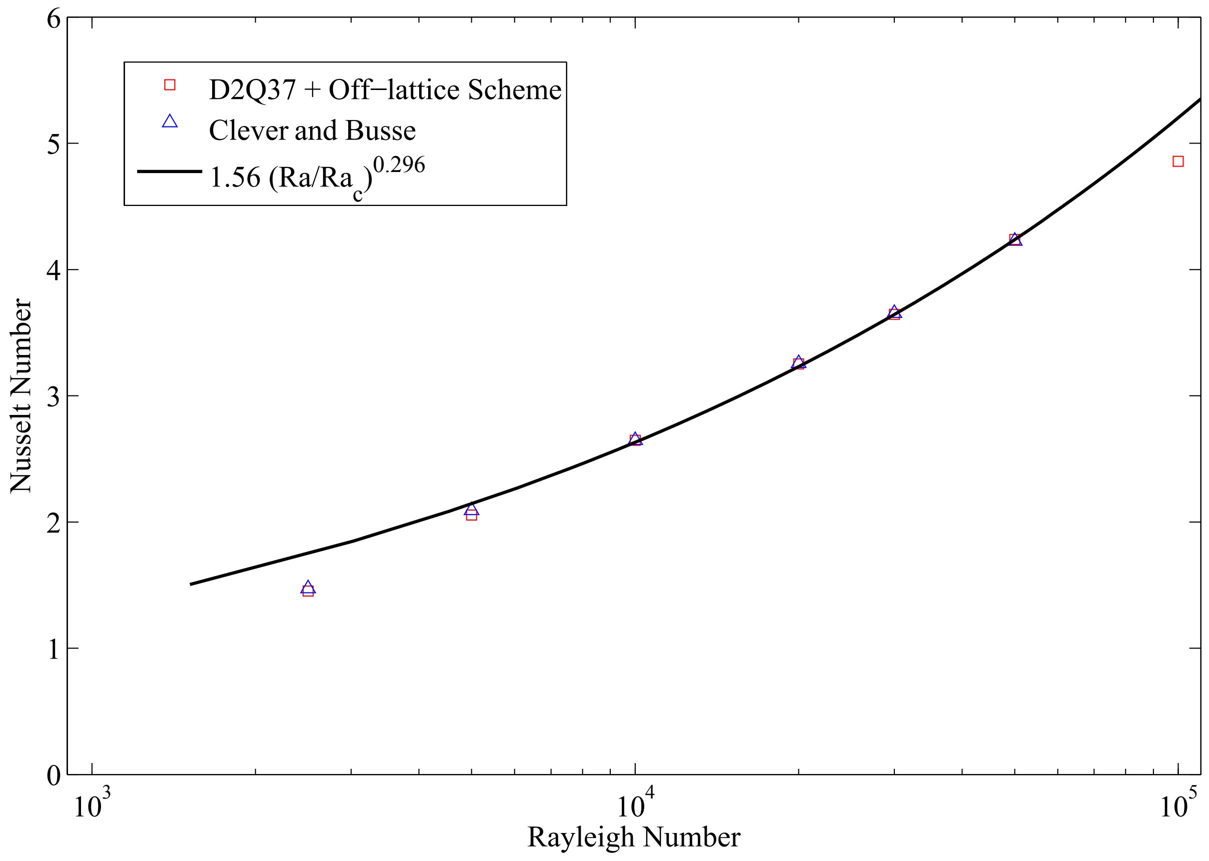


Figure 27: The dependence of Nusselt number on Rayleigh number. Also included are the simulation results by Clever and Busse [33], as well as the empirical power-law: $Nu = 1.56 \times (Ra/Ra_c)^{0.296}$.

We note that, the above results for the incompressible Rayleigh-Benard convection can be obtained accurately and more efficiently using the passive scalar, DDF method or the entropic LB-based guided equilibrium method, instead of the more expensive, D2Q39 model, as presented here. But it should remembered that the main strength of the higher-order models (D2Q39) lays in solving weakly and fully compressible thermal flows, not just incompressible ones. The actual strength of the higher-order models lies in its ability to simulate very high Ra number flows, without any Boussinesq-type assumptions. Investigations in this regard are left for future work. Additionally, the D2Q39-based models also have an inherent ability to incorporate heat sources (not validated here), which is absent in the passive scalar model. It is also attractive for modeling aeroacosutic flows (with suitable turbulence model), and, in general, high-speed compressible flows [72].

6.0 SERIAL AND PARALLEL IMPLEMENTATION OF A OFF-LATTICE HIGHER-ORDER LB CODE

The discussion thus far dealt has with the technical aspects of Hermite-polynomial based LB formulation and its validation on isothermal and thermal benchmark cases. In this chapter, we focus on their computer implementation related aspects. We start by introducing some salient features of the modern, large-scale, CPU-based computing systems and their architecture in Sec. 6.1. The hardware features of these systems are then leveraged to tune, and parallelize a off-lattice higher-order code. First, the algorithm and functions involved in a typical OLB scheme (in this case the BKG scheme code) are described in Sec. 6.2. Thereafter, in Sec. 6.3, we discuss optimization of a OLB code using the cache-optimization techniques. In Sec. 6.4, the optimized serial code is then extended to multicore shared-memory systems using OpenMP.

6.1 MODERN CPU AND HPC ARCHITECTURE

Before discussing the cache-optimization and parallelization techniques employed for the OLB code, it is important to discuss, in brief, some salient features of modern computer hardware architecture. Knowledge of the underlying hardware architecture is crucial in the design, implementation, and performance of any scientific application, and hence an overview is useful. The discussion focuses on multicore CPUs and systems built on them.

6.1.1 CPU Architecture

6.1.1.1 Multicore processors Scientific computing has become a ubiquitous tool in the scientific method alongside theory and experimentation. It is increasingly used across diverse fields such astrophysics, fluid dynamics, drug discovery, and linguistics, to just name a few. This progress over the past four decades has been largely attributed to the rapid advances in computer hardware. As famously predicted by Gordon Moore, for several decades, starting around 1970s, the number of transistors on a chip (computational power) doubled roughly every two years [78]. The increase in computational power was attained mainly by, increasing the number of transistors and their clock-rates in the CPU, along with with several advanced, low-level (parallel) hardware techniques, such as pipelining, super-scalar architecture, instruction-level parallelism, SIMD instructions, etc. [38, 52] Despite these advances, in the mid-2000s, engineers concluded that the performance could not be increased any further, due to an inability to dissipate the immense heat generated by the larger, more complex CPUs.

Multicore CPUs were developed to address the heat dissipation problem associated with singlecore CPUs (uniprocessor) with large clock-rates ($> 3GHz$). In multicore CPUs, instead of having only one core (processor) with a high clock-rate, there are several cores with lower clock-rates. In other words, the cores are not getting faster anymore, but they are being multiplied. Here, a CPU is understood to be a physical unit that occupies a single socket on a board, that contains several cores inside it[1]. Moreover, multicore processors have not only increased the computing power, but also made the processors more energy efficient. Today, it is common to have 4 to 8 cores per processor for home/small business use and, up to 16 cores per processor for enterprise/server use [6].

In addition to consumer and enterprise sectors, multicore processors also have had a significant impact in the field of scientific computing. Before their arrival, large-scale scientific computing was dominated by supercomputers build upon specially designed processors, such

[1]In the past, a CPU was the singular execution component for a computer. Then, multiple CPUs were incorporated into a node of a computing cluster. In the multicore times, individual CPUs were subdivided into multiple cores, each being a unique execution unit. CPUs with multiple cores are sometimes also called 'sockets'. Therefore, a desktop has one CPU with multiple core, whereas a node in a cluster has multiple CPUs, each containing multiple cores. Nodes are building blocks for clusters, regardless of node type.

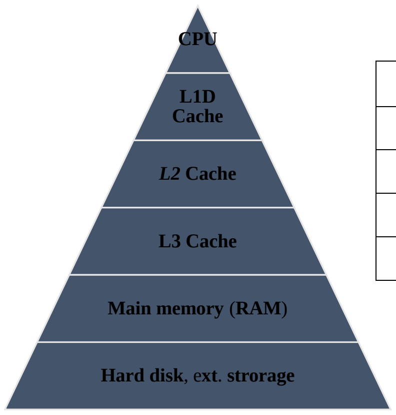
as vector processors by Cray, NEC, etc., and scalar processors, such as PowerPC by IBM. Although such computing systems still exist, most of the modern high-performance computing (HPC) clusters and supercomputers are based on commodity, off-the-shelf, multicore processors, that are not primarily designed for scientific computing [52]. Today, a modern HPC cluster which is used for large-scale scientific applications, has hundreds to thousands of CPUs (with multiple cores) connected to each other via a high-speed interconnect.

6.1.1.2 Cache Hierarchy in Multicore Processors While the overall CPU speed kept on increasing rapidly through multicore CPUs and other low-level hardware features, advances in memory-speed and its bandwidth (i.e. speed and amount of data from memory to the CPU) has lagged behind CPU progress. Very often, memory-access time are much higher than the time required to perform floating-point operations (arithmetic operations), a phenomenon known as the *DRAM gap* [56]. Consequently, for memory-inefficient programs, it is common for a CPU to spend most of its time waiting for data to arrive from memory.

In modern multicore processors, the DRAM gap is ingeniously hidden by using hierarchical levels of memory called as *cache memory,* or just *caches*. Cache memory are low-capacity, high speed memories that are integrated on a chip along with the CPU. Caches contain copies of main memory to speed up access to frequently needed data. In general, the closer the cache to the CPU, the faster the bandwidth, but the smaller its size. A typical memory hierarchy that can found in a multicore CPU is shown in Fig. 28.

6.1.1.3 Memory Architecture **Shared Memory**

In a shared-memory based architecture, all cores in a CPU share a common, global, memory or address space. The cores and the processes running on the cores, can operate simultaneously and independently, but share the same memory resources. Figs. 29 and 30 show a schematic of the two variants of shared-memory architecture. If the layout of the cores around the memory is such that the memory-access times for each core is equal, such shared memory machines are called as Uniform Memory Access (UMA). UMA is also referred to as Symmetrical Multiprocessor machines (SMP). UMA architecture is generally not amenable to large-scale scaling due to memory bandwidth limitations [52].



	Capacity	Bandwidth	Latency
L1D	32 kB	45 GB/s	4 CPU cycles
L2	256 kB	~30 GB/s	10+ CPU cycles
L3	6 MB	~ 20 GB/s	35+ CPU cycles
RAM	8 GB	< 8.5 GB/s	>100 CPU cycles

Figure 28: Typical cache memory hierarchy found in desktop range multicore CPUs.

Computer architecture where a core can access its own memory faster than other core's memory is called a Non-Uniform Memory Access (NUMA) machine (see Fig. 30). NUMA machines are built by scaling (linking) two or more SMP's by means of a bus interconnect. Here, unlike UMA, memory (address space) might be physically distributed, but *logically shared*. Moreover, if cache coherency is maintained across all cores, then such machines are also be called Cache Coherent NUMA (cc-NUMA). A workstation with multiple sockets (with 8-16 cores) can be considered NUMA. On a larger scale, the UV series of SGI, Inc. and the XE6 line of Cray, Inc. are examples of large scale NUMA. Locally, 'Blacklight' of the Pittsburgh Supercomputing Center has a SGI UV1000 system with 4096 cores.

From a programming point of view, UMA and ccNUMA work transparently and the programmer does not need to be concerned about inter-core communication, cache coherency, etc. Therefore, shared-memory systems provides a *user-friendly programming model*, through which an application can be relatively easily parallelized to fully exploit the multicore hardware.

Distributed Memory

In a distributed-memory architecture, the individual CPUs have their own local memory, but the memory addresses in one CPU does not directly (without explicit communication) map to another CPU. Therefore, the memory is not only physically, but also logically dis-

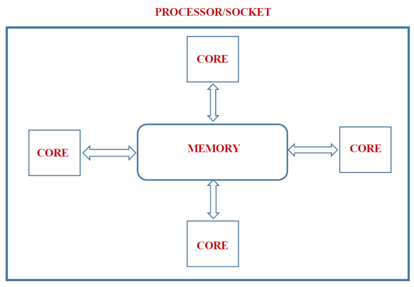


Figure 29: Uniform memory access (UMA) architecture.

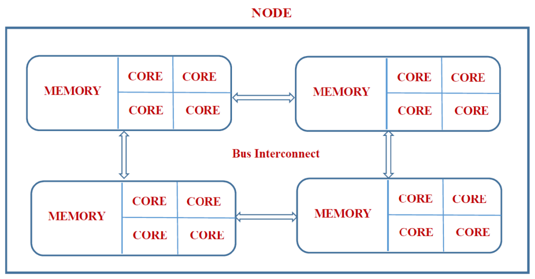


Figure 30: Non-uniform memory access (NUMA) architecture. Here, each core has its own hierarchy of L1 and L2 caches. L3 caches are shared among cores within a socket.

tributed. The CPUs are connected to each other through an interconnect (network), where the speed of interconnect can be as fast as a Infiniband, or as slow as an ethernet network. Fig. 31 shows a schematic of the distributed memory architecture.

A Beowulf cluster is a typical example of distributed memory architecture. This configuration, built on commodity processors, was popular in the 1990 and 2000s since they provided an inexpensive way to attain higher computational power using commodity single-core CPUs. The main advantage of the distributed-memory architecture is that it scales very efficiently with increasing the number of processors and size of memory. It is also cost effective since it is built of off-the-shelf CPUs and networks.

When a processor needs access to data in another processor, the programmer has to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility. Also, it is often difficult to map existing data structures, based on global memory, to a distributed-memory layout. All of these complexities make distributed memory programming much more complicated than shared-memory programming.

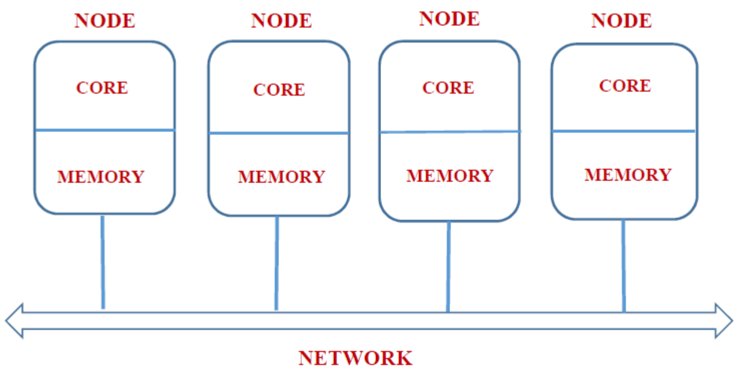


Figure 31: A typical distributed memory architecture with a single CPU per node. Figure adapted from [12].

Hierarchical (hybrid) Memory

Modern large-scale parallel computers are seldom based on purely shared-memory or purely distributed memory architecture; in fact, they are built as a combination of both. That is, several hundreds of shared-memory blocks are connected via a fast high-speed interconnect to form a hybrid-memory system. The CPUs within a shared-memory block, which constitute a node, share a global memory; while CPUs across across the nodes which are connected through the network, do not share a global memory. Fig. 32 shows a schematic for a hybrid memory architecture. Over the past decade, most supercomputers have been built on a hybrid-memory architecture, and this trend is expected to continue in the foreseeable future. In particular, the two-socket per node layout with multicore CPUs in each socket is the most commonly used hybrid configuration due to its optimal price-to-performance ratio.

Coprocessors

Recently, in addition to multicore CPUs, additional hardware called coprocessors are also incorporated within a node. Coprocessors, also sometimes called accelerators, are processors used to supplement the primary processors (CPU), and which are used for performing specialized operations, such as floating point arithmetic, graphics processing, signal processing, etc. Graphics processing units (GPUs), the Field-programmable gate array (FPGA),

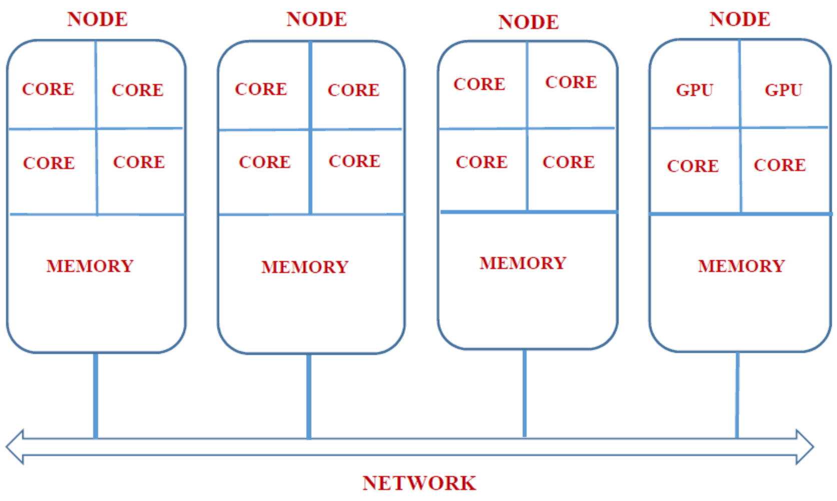


Figure 32: A typical hybrid memory architecture. Figure adapted from [12]

and the recently announced Intel's Many Integrated Chip (MIC Xeon Phi) are examples of coprocessors.

In conclusion, we can see that, various levels of parallelism exists in modern computing systems. Parallelism can be low-level such as instruction pipelining, superscalarity, etc., or relatively higher-level - at the level of multiple cores and multiple CPUs. Clearly, each component of the computer architecture - CPU, memory and the data path- present performance bottlenecks, and opportunities to improve program performance through optimization and parallelization. The programming models needed to exploit these hardware features are addressed in the following sections.

6.1.2 Parallel Programming Models

Parallel computing is a multidisciplinary field that involves developing programs/applications to fully utilize the underlying hardware features, in order to enhance the overall performance. Parallel computing combines elements of computer science and engineering, software engineering, numerical analysis, visualization, in addition to domain-specific knowledge.

Despite the tremendous advancement in computer hardware, commercial and scientific software and applications that utilize the hardware features have typically lagged behind[2]. However, in the past few years, the field of scientific parallel computing is rapidly advancing. Before the advent of multicore CPUs, many low-level hardware parallelism features were handled sufficiently by optimized compilers. Therefore, performance enhancement was often implicit; attained without much developer intervention. However, with the emergence of multicore accelerators, and large HPC clusters, *explicit* developer-intervention was required in order to utilize the hardware parallelism and the increased computational power. This issue is even more relevant for numerically-intensive codes. Therefore, a developer has to develop applications that are written specifically for the available hardware and its features - making the software hardware-dependent.

Clearly, prior knowledge of the available hardware and its architecture is critical. In particular, the memory layout of the underlying hardware system strongly influences the programming model to be adopted. For shared-memory machines (UMA and NUMA), usually, a thread-based model such as *PThreads* or *OpenMP* is adopted on top of a base, compiled, programming language, such as C/C++ or FORTRAN [12, 23]. On the other hand, for distributed-memory architecture, the Message Passing Interface *(MPI)* model is the most widely used option. For hierarchical-memory architectures based only on CPUs, a hybrid model consisting of OpenMP and MPI is used. If coprocessors such as GPUs or MICs are available as part of the hybrid architecture, then directives based programming APIs such *OpenACC* or even OpenMP can be can be used [82].

[2]Notable exceptions include ScaLAPACK, ATLAS, and other parallel linear algebra libraries.

6.1.3 Related Work

The lattice Boltzmann method is often commended for its simple algorithm and its ease of parallelization. Over the past decade, several works have focused on developing optimal data-structures and algorithms for the standard (stream-collide) LB algorithm. For the two-step, standard (stream-collide type) LB algorithm, several sophisticated implementations, such as combined collision and propagation [88], swap-algorithm [74], shift-algorithm, AA pattern [9], etc. have been developed, that enhance speed and/or optimize storage requirements. An excellent review and comparison of these implementation is given in [127]. Application-specific optimizations, such as for porous media flows and hemodynamics, have been developed to optimize memory by using sparse-memory usage [98]. Along with algorithm optimizations, researchers have also developed parallel implementations of the LB method for large-scale CPU and GPU-based heterogeneous platforms [15, 17, 34, 57, 91], and even for the cell architecture (Sony Playstation) [85]. LB based implementations have also been adopted for vector processor-based supercomputers [87, 125, 106] with much success. Interestingly, in terms of the GFlops/s metric, vector computers were found to be best suited for LB simulations, compared to cache-based CPUs. Open source software is also available such as *OpenLB* and *Palabos.*

It should be pointed out that most of the previous works have dealt with the standard LB method along with the $D2Q9$ lattice or, its $3D$ equivalent $D3Q17$ or $D3Q19$. Implementations and optimizations on higher-order (extended) lattices have just started to appear in literature based on the standard algorithm [17, 91, 73].

6.1.4 Relevance of Pursued Work

The standard LB method, based on the Lagrangian streaming-collision scheme, has several computational, numerical, and physical modeling advantages for complex flows, as described in Chapter 3.4. Computationally, the LB method is explicit, involves relatively few floating-point operations per node, and the algorithm is straightforward to implement. Also, because of the spatial locality of the computations, and Cartesian domains, it is easy to optimize and parallelize a LB code for a specific hardware[3].

For industrial-scale laminar/turbulent flows through complex geometries, however, it can be argued that the non-body fitting LB method is still less flexible as compared to general, unstructured, FV based methods. As discussed in Chap. 4, the geometric flexibility the standard LB method is constrained by the requirement of a uniform Cartesian mesh. LB computations for practical flows are also computationally demanding due to the small spatial and temporal resolution requirement. In the case of steady state problems, the convergence is also very slow due to the low Ma number requirement [119, 48]. While some attempts have been made to introduce a block-structured mesh, adaptive mesh, etc., they still appear to be computationally lacking compared to state-of-the art unstructured FV methods.

The off-lattice Boltzmann method, which combines the LB method and the FV/FD discretization of space, may provide the necessary link to deal with complex geometries, yet keep the mesoscale character of the LB method intact. The OLB method is also an attractive scheme for modeling thermal flows on complex geometries, and also flows high Kn number (microscale) flows, especially with higher-order non-space-filling lattices. For the same quadrature accuracy, the number of discrete velocities in a non-space-filling lattice are fewer in number compared to a space-filling lattice, and hence a considerable computational advantage can be achieved. In short, OLB provides a generic extensible framework to model a variety of complex flows using several higher-order lattices at the same time can be extended to non-Cartesian domains. Therefore, as the finite-volume lattice Boltzmann and finite-difference lattice Boltzmann methods - both types of OLB methods - become more popular, it is important to address the computer implementation issues and optimization

[3]The parallelization of LBM on complex geometries, however, is not trivial due to load balancing issues.

techniques on modern hardware platforms. In this work, we focus on the finite-difference type OLB method. *Since we combine the finite-difference type spatial discretization and an off-lattice type temporal discretization, we call the resulting scheme as finite-difference lattice Boltzmann (FDLB) scheme.*

The FDLB algorithm, is fundamentally different that the standard LB algorithm. As discussed later in Sec. 6.2, the FDLB method consists of additional functions or routines that are not applicable in standard LBM. This key difference in the algorithms renders many of the optimized streaming-collision implementations inapplicable for FDLB.

6.1.5 Scope of Work

Traditionally much effort was directed towards developing computationally optimal algorithms, that minimize floating-point operations. However, due to much slower memory bandwidths and data access latencies, *memory optimization* has also become equally important. Often, to utilize the available parallel hardware, efforts are put in towards parallelizing a program, without first optimizing it for serial (single-core) hardware. In fact, the difference between an unoptimized and optimized program can be very large even on a single core level. Parallel versions of a program are also more likely to scale better if they are built upon an optimized single-core program rather than on an unoptimized one [42, 52].

Therefore, having briefly discussed the state-of-the-art hardware architectures, we seek to exploit them, to optimize and parallelize a FDLB program. A *bottom-up* approach is implemented in this work. In particular, the work is divided into two smaller tasks:

1. To develop an OLB code with optimal data-structures and algorithms, such that cache-hierarchy is utilized efficiently (this is described in detail in Sec. 6.3) ; and
2. The cache-optimised FDLB code is then used as a basis to develop a *parallel* FDLB program using OpenMP, suitable for systems with shared-memory architecture (this is presented in Sec. 6.4).

The novelty of this work lies in the fact that, to best of author's knowledge, this work represents the first effort to combine optimization and parallelization of a FDLB scheme with a higher-order lattice (D2Q37).

6.2 FINITE-DIFFERENCE LATTICE BOLTZMANN METHOD

Before discussing the techniques employed for single-core optimization and parallelism, it is useful to discuss some algorithmic features of a typical FDLB code. This helps us to identify the bottlenecks and constraints, and therefore devise better strategies to optimize.

6.2.1 OLB Algorithm

Similar to the standard LB algorithm, the FDLB method is also an inherently unsteady scheme. But, in terms of the algorithm, the FDLB method evolves in a completely different fashion. The characteristic feature of the standard LBM - the streaming process - is no longer applicable in the FDLB method. Instead, a typical FDLB scheme, is similar to time-marching (explicit) schemes, such as Euler methods, Runge-Kutta methods, etc., that are used to solve ordinary differential equations (ODEs). In FDLB, we solve a system of ODEs at each node, where the number of equations to be solved equals the number of discrete directions in the LB model (e.g., nine equations per node in case of $D2Q9$, 37 equations per node for $D2Q37$). Also, depending upon the spatial discretization method used, we have the additional task of computing the gradients (in the FDLB method) or fluxes (in the FV method). A FDLB code has the following four main routines/functions:

1. `EDF()` and `macrovariables()`: The `EDF()` subroutine computes the equilibrium distribution function, while the `macrovariables()` computes the macroscale flow variables - density, velocity and temperature (if applicable). These routines are identical to the standard LBM. `EDF()`, which comprises of computing a complex algebraic expression, is strictly *local* in terms of its spatial data dependency; i.e., $f_i^{eq}(\boldsymbol{x})$ at a particular node $\boldsymbol{x}$ does not depend upon the values at its spatial neighbors $\boldsymbol{x} + \Delta x$, $\boldsymbol{x} + 2\Delta x$, etc. The actual discrete form of the equilibrium distribution, of course, depends upon the velocity lattice used.
2. `gradient()`: This step is performed instead of the streaming process in the standard LBM. `gradient()` computes the spatial gradients (first-order derivatives) and second-order derivatives of the distributions on the fluid nodes. The actual form `gradient()`

depends upon the spatial stencil (scheme) used for computing the derivatives. For example, in the case of the central-differencing scheme, which is used in this work, the stencil consists of one immediate neighbor each in the $\boldsymbol{x}$ and $\boldsymbol{y}$ directions. This stencil, however, remains same even in case of higher-order lattices. In contrast to the streaming step where no floating-point operations are needed, `gradient()` clearly involves additional floating-point operations. Similar gradient computation also exist in multiphase, multicomponent simulations.

3. `update()`: This step involves updating (evolving) the distributions per the evolution equation, for example Eq. 4.18 in Sec. 4.3. The `update()`, which consists of computing algebraic expressions, is applied only to fluid nodes, and depends upon the temporal discretization scheme selected. So, depending upon the scheme selected, different `update()` schemes , such as the scheme listed in Tab. 2, can be adopted. This approach makes the overall code modular. As with `collision()`, this routine is also strictly local.
4. `boundaryCondition`(): In this subroutine, instead of the widely used bounce-back scheme, alternate boundary schemes such as the extrapolation boundary-condition, as described in Sec. 4.4 is performed. This step can also be viewed as an update procedure for distributions at the wall.

It should be noted that additional physics for complex flows, such as multiphase flows, can be added incrementally on top of the three basic functions outlined above. Similarly, structured, *non-Cartesian geometry* can also be added in a modular fashion, through a *coordinate transformation* procedure. The ease of adding additional functionality makes the overall code modular, mainly due to the nature of the LB method. Although the code can be extended to various cases, the focus in this work is on the analysis and optimization of the core FDLB algorithm - `gradient()`, `EDF()`, `update()`.

Memory Requirement of the LB method

A typical LB *research code* is implemented on a *dense-matrix* basis, making the LB method *highly memory intensive*. In a naive, full-matrix implementation, the distributions for all of the nodes in the domain have to be allocated/stored in memory [15]; for example, for the D2Q9 lattice, at each node, nine distributions have to be stored in addition to density and $\boldsymbol{x}$ and $\boldsymbol{y}$-component velocities. Therefore, the per-node memory requirement is quite large. To illustrate this further, consider a two-dimensional, single-phase flow problem to be simulated with the D2Q9 lattice on a regular Cartesian 256^2 ($\sim$ 65000 nodes) grid. In this case, the total memory requirement, including for distributions and flow variables, and assuming double-precision, is around 10 MB. On the other hand, a full-scale three-dimensional production run on a 256^3 grid with the D3Q19 would require around 6 GB! The memory requirement also increases if more complex flows problems, such as thermal, multiphase or multicomponent flows are to be simulated. Additionally, in case of a higher-order lattice, D2Q39 for example, the memory needs per node increases by a factor four compared to to D2Q9, whereas in three dimensions with D3Q19, it can reach a factor of six.

LBM memory requirement is in sharp contrast to that of conventional FV/FD-based CFD, especially in three-dimensions. For the same grid size, the LBM memory requirement is at least an order-of-magnitude higher than FD or FV based methods, where typically stores only the three velocity components and one pressure. A sparse storage implementation in FD/FV based methods would require even less storage. The memory intensive nature of the LB method is an important aspect to consider in designing optimal implementations, so as to avoid memory-bandwidth constraints. The FDLB is even more memory intensive than standard LBM, since, in the standard LBM, through combined streaming and collision the equilibrium distributions need not be stored in a separate variable (multidimensional array).

6.3 SINGLE-CORE OPTIMIZATION

6.3.1 Design Basis

As discussed previously, several options exist for temporal and spatial discretization schemes in the FDLB method. Similarly, several discrete-velocity sets, both space-filling and non-space-filling, can be chosen according to the flow to be simulated. As the base or reference case, BKG scheme is used as the temporal evolution scheme, and the BGK operator is used as the collision model. Correspondingly, the `update()` function is computed per the BKG scheme. Spatial gradients in `gradient()` are computed using the second-order central-difference scheme [60]. The discrete velocities and the corresponding equilibrium distributions developed in Chap. 3 are used.

In terms of the discrete-velocities, the D2Q9 lattice is developed first as the base case. The space-filling D2Q17 and D2Q37 lattices, along with the non-space filling D2Q12 lattice are also developed. An important feature of the program is its modularity with respect to the lattice structure. The program is designed such that, the lattice and the corresponding equilibrium distributions can be chosen in a modular fashion by just a changing a few lines of code. Also, while an optimized D2Q9 based standard LBM cannot be extended to non-D2Q9 lattices without a good deal of reworking; the FDLB developed here is flexible with respect to the lattice employed. Similarly, different temporal schemes for `update()`, such as the AE/CI and GZ schemes, have been coded, and can be incorporated in a modular fashion. Although not implemented here, other schemes for computing the derivatives in the `gradient()`, such as upwind-differencing can be easily coded, and incorporated in the overall program.

The overarching design philosophy behind singlecore optimization is the idea of efficient usage of caches. Some of its basic principle are described below to aid the results described subsequently.

6.3.2 Cache Optimization Principles

As briefly described in Sec. 6.1.1.2, the modern memory is not fast enough for modern CPUs, therefore, obtaining data from memory (RAM) usually takes about a dozen CPU cycles [38, 56]. Cache acts as a buffer or reservoir containing copies of recently used data or code. In the absence of an on-chip cache, a CPU may be rendered idle for short durations, while waiting for data to arrive from the memory. This problem is especially relevant for memory-intensive algorithm such as the FDLB. Therefore, one of the most effective techniques to improve the performance of a serial program is to fully utilize the higher bandwidth and lower latency of cache, i.e., to make the program *cache-optimized*.

Typically, when new data is required, the older data is removed from the cache and newer data is brought in from the main memory [64]. If the problem size is small enough to be entirely accommodated in the cache, we may observe near peak performance of the program. But, as the problem size increases, all of the data cannot be accommodated in the limited cache, and hence data needs to be frequently accessed from main memory. In fact, it is common nowadays to have scientific codes that require hundreds of MB to several GB of data. Clearly no cache is big enough to accommodate such large data. The key principle, therefore, of developing cache-efficient programs is to minimize the amount of data transfer between the CPU and memory such that the data stays as close to the processor (in the cache) as possible. This is called the principle of *locality of reference*.

The locality has two types: temporal locality and spatial locality. *Temporal locality* refers to the statistical probability that recently accessed items are likely to be accessed again in the near future.This is achieved in practice by fusing iterative loops with same loop traversal, using local variables, etc. Similarly, *spatial locality* is achieved if data located physically close to each other in the cache are accessed together in time. For example, operations on arrays or tables.

In the FDLB context, *choice of data-structure,* and *looping for optimal data-access,* are simple and practical ways of achieving cache-optimization. The implementation details as described below are broadly based on principles laid out in [42, 123] and [64].

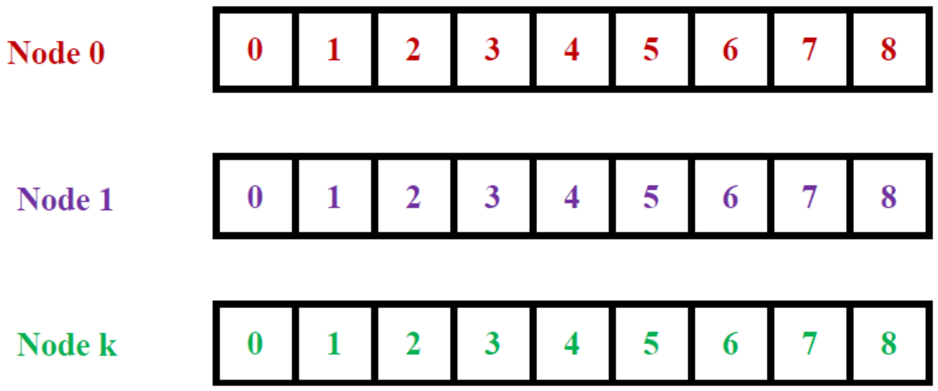


Figure 33: Memory layout of Array of Structure.

6.3.3 Choice of Data Structure

As with the standard LBM, the fundamental data structure/variable in the FDLB method are the distributions, $f_i(\boldsymbol{x})$, at each node. In most research codes, distributions are commonly stored as multidimensional arrays, where the order of the indices of the arrays define their physical layout in memory. The choice of data structure and its layout in memory is key to achieving peak cache-utilization. Applying a dense-matrix approach, for a two-dimensional structured domain, there are two options available for storing distributions namely: `f(NX,NY,k)` or `f(k,NX,NY)`. Here, `k` is the number of discrete velocities (directions) at each spatial node, and `NX` and `NY` are the number of nodes in the x and y dimensions, respectively. Following the row-major convention of C/C++, in the `f(NX,NY,k)` layout, `k` is the fastest moving index; meaning that for each spatial node, the distributions for each of the `k` directions are stored contiguously in memory. A multidimensional array having such memory-layout is the most intuitive layout and, is termed as the *array of structures* (AoS), or in LB parlance - the *collision-optimized* layout. Fig. 33 shows a schematic of the memory layout of the AoS.

On the other hand, in case of `f(k,NX,NY)` layout, `NY` is the fastest moving index, meaning that for each direction `k`, distributions for the spatial nodes are contiguous in

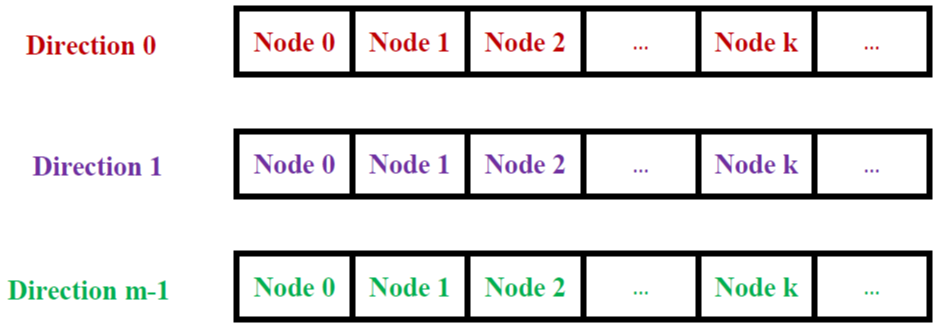


Figure 34: Memory layout of Structure of Array.

memory. This layout is termed as a *structure-of-array* (SoA), or in LB parlance, as the *propagation-optimized* layout. Fig. 34 shows a schematic of the memory layout of the SoA.

SoA and AoS layouts were studied in detail in [123], which concluded that SoA layout is more efficient for cache-based systems. Therefore, we have used the SoA layout in this work. For the actual implementation of the SoA layout, however, instead of the built-in C++ arrays, the `cube` template class from the open-source Armadillo C++ linear algebra package is chosen for $f_i(x)$ and $f_i^{eq}(x)$ [95]. `cube` is a templated class used for storing 3D arrays. So, for example, the distributions for the entire mesh are defined as `f(NX,NY,NV)`, where `NX` is the number of nodes in the x direction, `NY` is the number of nodes in the y direction, and `NV` is the number of discrete velocities. In the `cube` data type, the elements of the three-dimensional array are stored in a slice by slice fashion. Within each slice (matrix), the elements are stored in a column-major fashion. In the LB context, this means that each slice of the cube corresponds to a particular discrete velocity. Therefore, in effect, the `cube` class used here is equivalent to a propagation-optimized layout. Armadillo has many other advantages, such as MATLAB like syntax and functionality, which aids quick prototyping, while maintaining the speed and portability of a compiled language (C++).

It is noted that, since the swapping technique of standard LBM is inconvenient, both $f_i(\boldsymbol{x})$ and $f_i^{eq}(\boldsymbol{x})$ are stored in a separate `cube`. This makes our FDLB implementation require at least twice as much memory storage as the standard LBM.

6.3.4 Data Access Optimization

The presence of a cache is not always a guarantee for better performance. Given a particular memory layout, it largely depends upon the memory access pattern in the loops, and how well it is written to exploit the caches. Data-access optimization of the *three core* functions of the FDLB algorithm are described below.

`collision()`

`collision()` is a straightforward candidate for cache optimization through efficient data access. `collision()` is computationally intensive, but is strictly local in configuration space, which means that it has no data dependencies on neighbors and results in perfectly nested rectangular loops. Taking advantage of the local nature, we can re-structure the loop-order in `collision()` to suit the SoA layout. As shown in the listing below, the velocity-loop, `k`, is made the outer-most loop, since it is the slowest moving index. Within each velocity loop `k`, the innermost (fastest) loop is that for `x`, since within a slice, elements are stored column-wise. This looping-order results in a stride-1 access, and hence good temporal locality is achieved[4]. Floating-points operations at a node are also minimized through pre-evaluation of common sub-expressions such as $\boldsymbol{\xi}_i \cdot \boldsymbol{u}$, $\boldsymbol{u} \cdot \boldsymbol{u}$, $\boldsymbol{\xi}_i \cdot \boldsymbol{\xi}_i$, etc. The psuedocode below shows how this is achieved:

```
for (int k=0; k<NV; k++)   {
   for (int y=0; y<NY; y++)    {
      for (int x=0; x<NX; x++)       {
            if (isFluid(x,y))==1    {
            edotu = ex(k)*ux(x,y) + ey(k)*uy(x,y);
            edotuSq = edotu * edotu;
            uSq = ux(x,y)*ux(x,y) + uy(x,y)*ux(x,y);
            eSq = ex(k)*ex(k) + ey(k)*ey(k);
            geq(x,y,k) = W(k)*density(x,y)*(1.0 + edotu + 0.5*(edotuSq-uSq+(T(
   x,y)-1) ...
         }
      }
   }
}
```

[4]Stride is the number of bytes one has to move to get from one element of an array to the next (or previous) one. So, a stride-1 access (read and write) is more efficient, since the elements of arrays are accessed consecutively, and the array is also most likely to be in registers or cache.

`gradient()` and `update()`

For `gradient()`, the optimization is slightly more involved due to its neighbor dependency. The dependency comes from the fact that, for each discrete velocity (direction), we need to compute the spatial derivative of the distributions, which depends on the distributions at neighboring nodes. Therefore, to minimize non-contiguous data-access, the distributions for neighbors of `f(x,y,k)` such as `f(x+1,y,k)`, `f(x−1,y,k)`, `f(x,y+1,k)`, and `f(x,y−1,k)`, should be in close proximity, preferably in the same cache-level.

As in `collision()`, locality of reference in `gradient()` can be maintained by structuring the loops to suit the memory layout of cube. Since the `SoA` data-structure stores distributions for each direction contiguously, distributions for a particular direction, `k`, can be fully loaded into cache, and the gradients for the entire domain for that particular direction can be computed with minimum cache-misses.

Also, notice that a particular `f(x,y,k)` is accessed in both `gradient()` and `update()`. Hence, *loop fusion* (combining `gradient()` and `update()` in one loop) allows f_i - a large-sized array - to be used fully once loaded in cache, thereby achieving good temporal locality and lower the memory bandwidth. Similar fusion of collisions and streaming was done in [123] - resulting in smaller data transfer between CPU and memory. Moreover by fusing both the functions in a single loop, we need not store the recently computed gradient values separately in a multidimensional array. The gradients computed for direction `k` at a particular node can be used immediately to evaluate the `update()` for that node within the same loop. This procedure of fusion of loops involves breaking some well-defined abstractions and modularity, but nonetheless the computational advantage that can is obtained is substantial. This details of the combined subroutine (termed `gradient_update()`) are shown in the listing below:

```
/* For direction #0 */
for (int y=0; y<NY; y++)    {
   for (int x=0; x<NX; x++)  {
      if (isFluid)== 1     {
            /* Compute first-order spatial-derivatives of f(x,y,0)*/
            /* Compute second-order spatial-derivatives of f(x,y,0)*/
            /* update f(x,y,0)*/
         }
      }
   }
```

```
/* For direction #1 */
for (int y=0; y<NY; y++)     {
   for (int x=0; x<NX; x++)   {
      if (isFluid)== 1      {
            /* Compute first-order spatial-derivatives of f(x,y,1)*/
            /* Compute second-order spatial-derivatives of f(x,y,1)*/
            /* update f(x,y,1)*/
         }
      }
   }
/* For direction #2 */
...
```

We have also observed that having nine smaller `gradient_update()` loops corresponding to the nine directions is more efficient than having a single large loop. This technique is commonly referred as *loop unrolling*. The above listing shows how this is done. This technique however, can be tedious in the case for higher-order lattices, especially in three-dimensions, where there can be up to 108 velocities depending upon the chosen lattice. The payoff, however, is substantial.

In short, the techniques of loop unrolling and loop fusion for `gradient()` and `update()` have considerably reduced the run times. For a given mesh size, these two techniques have resulted in a reduction of about 20-28% in case of $D2Q9$, and about 34% in case of $D2Q37$.

6.3.5 Results and Observations

6.3.5.1 Test Basis To test the efficacy of the various serial-optimization techniques described previously, we utilize three PCs with different commercial off-the-shelf multicore CPUs for testing. Their hardware specifications are described in Tab. 4.

The Taylor-vortex flow (TVF) as described in Sec. 4.5.1 is chosen as the test problem. TVF is a flow on a periodic domain and hence the complexities of specific boundary treatment can be avoided and only the core algorithm can be addressed. The finite-difference scheme is used for spatial discretization and the BKG scheme for temporal discretization. The `gcc` compiler with the `-O2` optimization level is used for compiling the source codes on Linux (Ubuntu)- based machines. `clock()` from the `<time.h>` library is used to measure the program execution time. The program is run for a fixed 20,000 iterations in each grid's

Table 4: Specifications for the hardware used in performance evaluation

CPU Specifications			
CPU Name	i5-2320	i5-760	i7-960
Microarchitecture	Sandybridge	Lynnfield	Bloomfield
No. of cores	4	4	4
No. of Threads	4	4	8
Clock Speed (GHz)	3	2.8	3.2
Max Frequency (GHz)	3.3	3.33	3.5
Memory Specifications			
RAM (GB)	8	8	8
Memory Types	DDR3-1066/1333	DDR3-1066/1333	DDR3-800/1066
# of Memory Channels	2	2	3
Max Memory Bandwidth (GB/s)	21	21	25.6
L1 Cache	4 X 32 KB	4 X 32 KB	4 x 32 KB
L2 Cache	4 x 256 KB	4 x 256 KB	4 x 256 KB
L3 Cache (Smart)	6 MB shared	8 MB shared	8 MB shared
Cache alignment		64	64

case. The optimization techniques were implemented incrementally to check whether they did indeed provide a performance enhancement. The decrease in runtime (speed-up), as discussed below, represents the combined effects of all of the implemented optimization techniques. The speed-up is compared against the base case which consists of a straightforward implementation of the FDLB algorithm without any of the optimization techniques outlined in the previous section.

6.3.5.2 Results Figs. 35, 36 and, 37 show the speed-up obtained for different grid-sizes, on different CPUs, as a result of the cache-optimization techniques. From the above figures, we can observe that, at least a 2X *performance improvement* on different CPUs was achieved through the cache-optimization techniques. Although all CPUs have four cores each, the i7-960-based hardware gives the highest speed-up for a given mesh size. This can be primarily attributed to fact that, among the three CPUs, the i7-960 CPU has the highest clock speed and, also the highest maximum memory bandwidth, which reduces the time take to transfer data from memory to the CPU.

From the figures we can also see that the *performance enhancement is better for larger-sized grids*. This is mainly due to the higher computation to communication ratio (CCR) associated with larger grids. CCR can be broadly defined as the ratio of time spent calculating to the time spent communicating. Since memory bandwidths are typically smaller than CPU clock-rates, a higher CCR, therefore, leads to more computations (floating point operations) for a given amount of data that is already loaded in cache. Thus, in a large-sized grid, the proportionately larger distributions are processed much more efficiently once they are loaded in to the cache through a loop. In other words, the loop-overheads are much smaller than the computational effort. On the other hand, with smaller sized grids, the loop overheads can be much higher, and sometimes, even comparable to the computational effort, thereby reducing the scale-up.

In the figures, we have also indicated the effects of the so-called *cache-trashing* phenomenon[5]. In engineering simulations it is common practice to use power-of two-grids, such

[5]Cache thrashing is a phenomenon where the memory is accessed in a pattern that leads to multiple memory locations competing for the same cache lines, resulting in excessive cache misses and therefore, higher runtime.

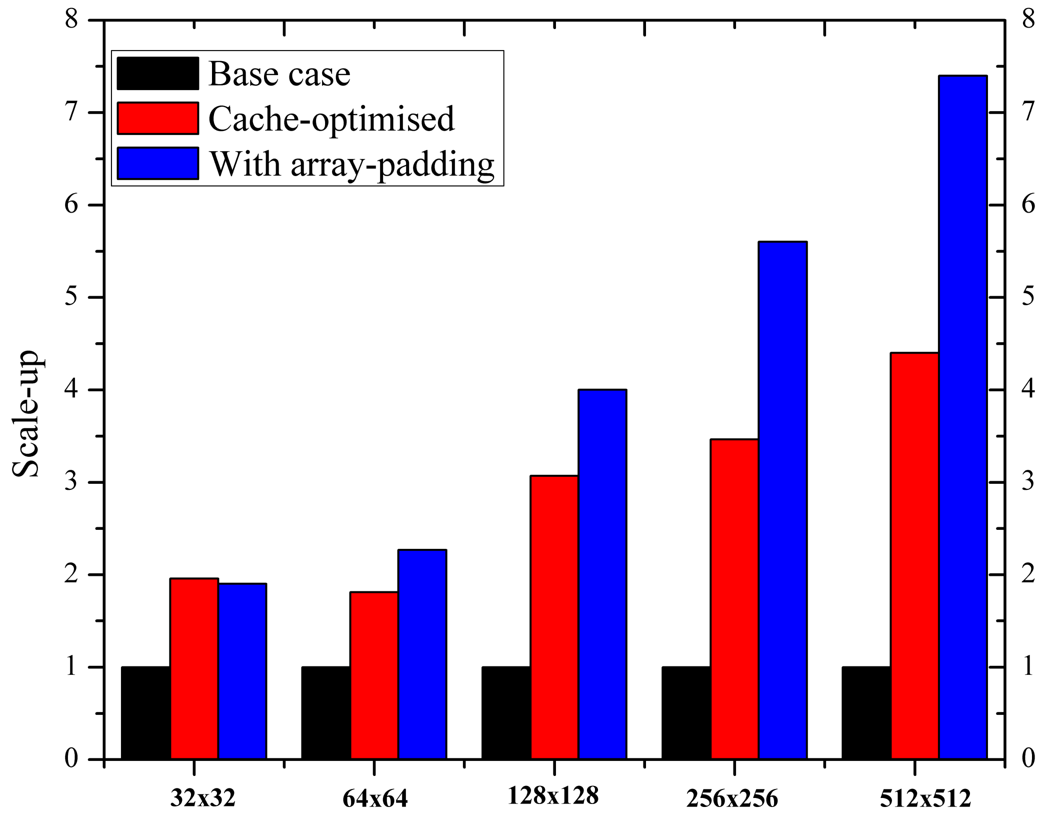


Figure 35: Speed-up observed as a result of cache-optimization techniques on i5-760 based hardware.

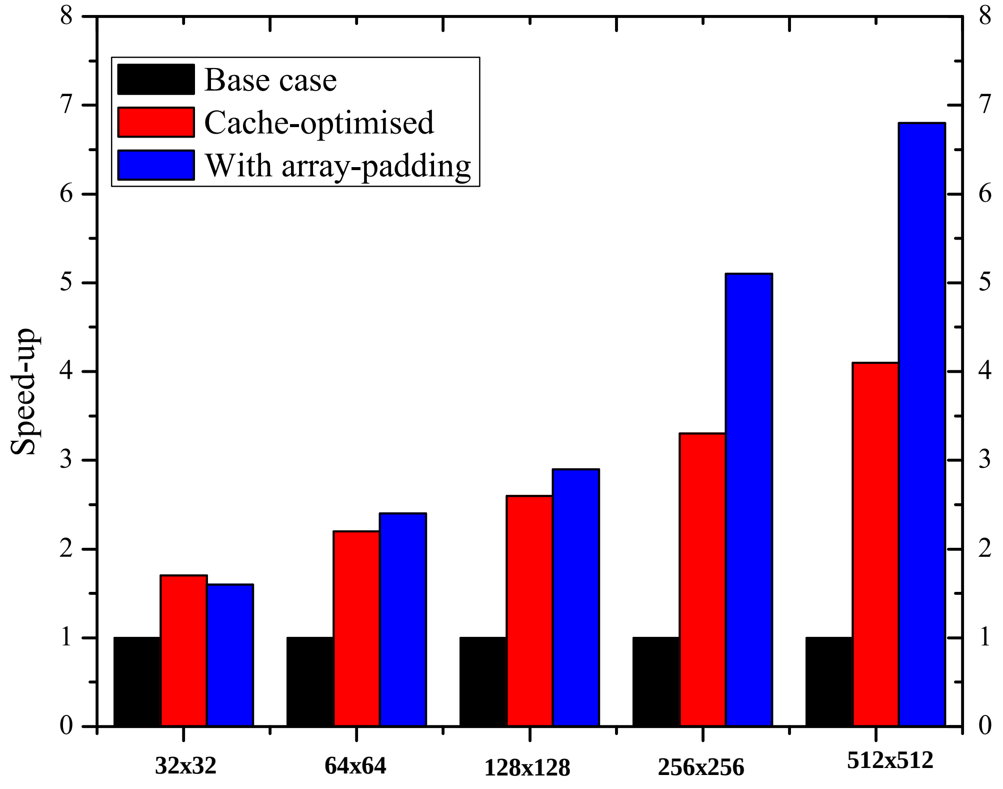


Figure 36: Speed-up observed as a result of cache-optimization techniques on i5-2320 based hardware.

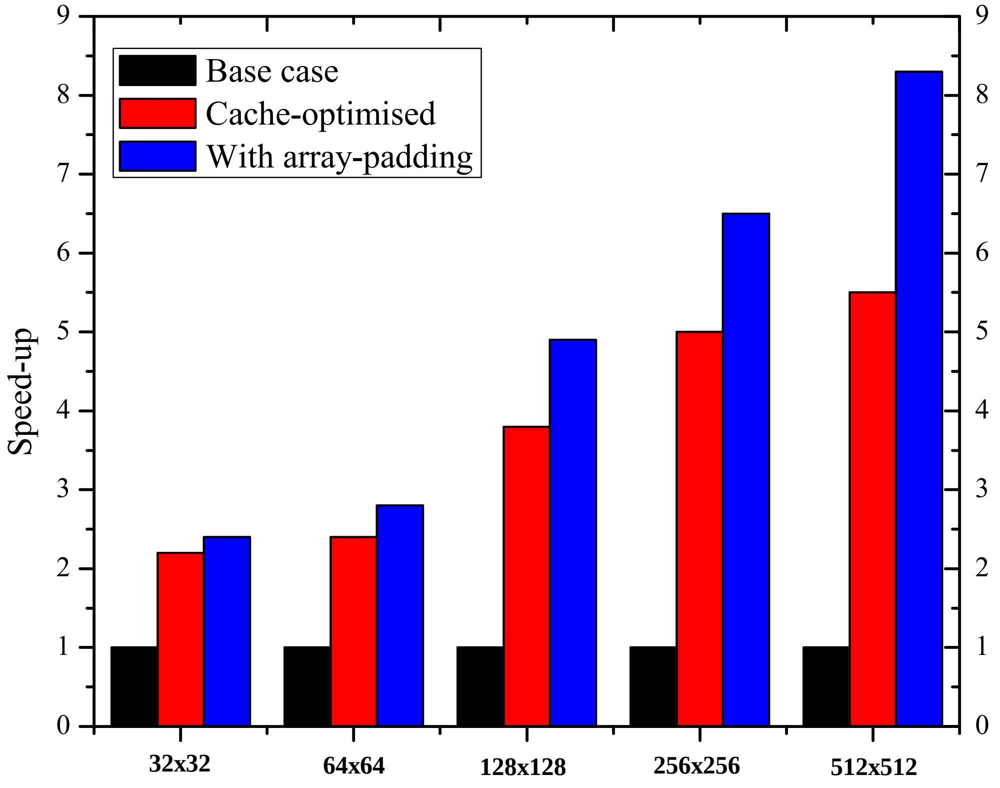


Figure 37: Speed-up observed as a result of cache-optimization techniques on i7-960 based hardware.

as $32^2, 64^2, 128^2$, etc., since they allow easy analysis for grid-independence studies. However, it is well-known that a SoA layout is prone to cache-trashing phenomenon [99]. In simpler terms, it means that, everything else remaining same, a power-of two grid, such as a 256^2 grid will take substantially longer computational time compared to non power-of-two grid such as a 257^2 grid. Cache trashing can be easily eliminated by *padding the arrays*; i.e., using 33^2, 65^2, 129^2, instead of 32^2, 64^2, 128^2, respectively.

The above positive results indicate that through optimal data-structure and simple loop restructuring techniques (unrolling and fusion), substantial performance gains can be obtained even on desktop-level hardware.

MLUPS

The speedup metric, as described above, is useful only for comparing different implementations on the *same* hardware. Often, it is useful to compare the performance of the FDLB code relative to other LBM implementations developed by other researchers on different architectures. The obvious metric - Flop/s - is not the best metric since this can vary widely based on factors, such as the implementation of the model, compilers, hardware, etc. A more meaningful metric in the LB context, is the computational work done per unit time, which can be quantified as the *MLUPs* metric - million lattice point updates per second [15, 123]. MLUPs is a widely used metric to compare various LB implementation and it relates directly to the wall clock time for solving the problem. It measures the runtime of a program depending only on domain size and number of time steps used in the simulation [91]. Eq. (6.1) shows how the peak number of potential MLUPs is calculated for a specific simulation. Here $T(S)$ is the run-time corresponding to S iterations (steps) and N is the total number of nodes in the domain:

$$P = \frac{N \times S}{T \times 10^6} \tag{6.1}$$

MLUPs for the FDLB code was evaluated for different computational work loads (grid sizes) per Eq. (6.1), and are plotted in Fig. 38

The MLUPs numbers for the FDLB code developed here are comparable (slightly lower) than comparable singlecore standard LB implementations. The maximum MLUPs for the FDLB code 2.7 MLUPs, obtained on a 512^2 grid on a i7-960 based machine.

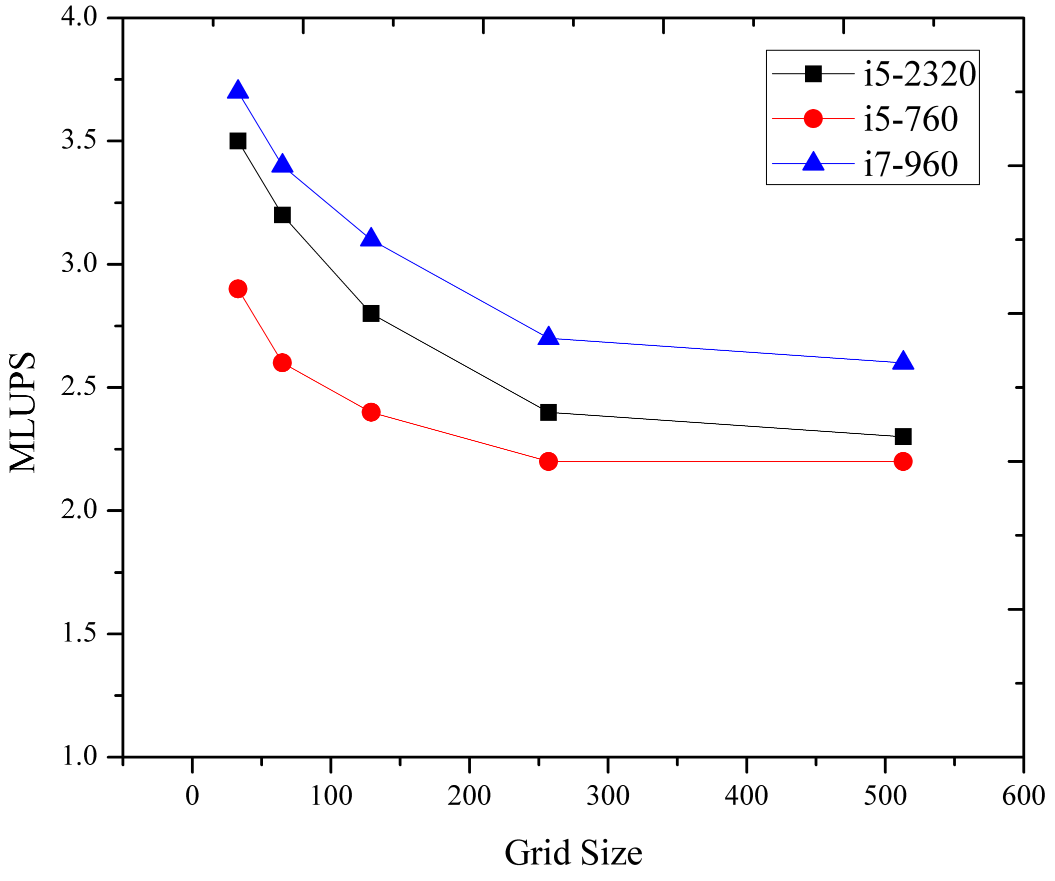


Figure 38: MFLUPs observed on different COTS CPUs based on the grid size. Grid sizes above are square grids, e.g. $33^2, 65^2$, etc.

Table 5: Computational time (in seconds) for various grids sizes with higher-order lattices. Numbers in parenthesis indicate the increase in compute-time relative to D2Q9.

Grid size	**D2Q9**	**D2Q12**	**D2Q17**	**D2Q37**
65^2	34.6 *(1X)*	47.4 *(1.4X)*	66.9 *(2.12X)*	159.2 (4.63X)
257^2	236.1 *(1X)*	321 *(1.34X)*	453 *(1.95X)*	1050.7 (4.45X)

Additional computational effort of higher-order lattices

Next, we investigate the increase in computational effort that is associated with using higher-order lattices. It is obvious that using higher-order lattices increases the computational effort, however, the qualitative and quantitative nature of this increase is unclear (linear, quadratic, etc.) To do this, we compare the computational time of a sample flow problem (isothermal Taylor-vortex flow) which is simulated with lattices of different order of quadratures. Fifth, seventh, and ninth order quadratures in two-dimensions are tested. Both space and non-space filling versions of the quadratures are compared. Importantly, the discrete form of the equilibrium distribution is chosen accurately, corresponding upon the quadrature of the lattice. That is, terms up to $\mathcal{O}(\boldsymbol{u}^2, \theta)$ are retained for the fifth-order lattice (D2Q9), $\mathcal{O}(\boldsymbol{u}^3, \theta)$ is retained in the equilibrium distribution for the seventh-order lattice (D2Q12 and D2Q17) and terms up to $\mathcal{O}(\boldsymbol{u}^4, \theta)$ are retained for ninth-order lattices (D2Q37). As before, the BKG temporal scheme is used along with FD spatial discretization. Simulations are performed for various grid sizes, and the number of time-iterations is fixed at $25,000$ in each case. Tab. 5 summarizes the results of these tests.

From Tab. 5, we can observe that, with higher-order lattices, the computational time (effort) increases *non-linearly* with an increase in the number of velocities included in the lattice. The increase stems mainly from the computational effort involved in evaluating the equilibrium distributions, whose complexity increases with an increase order of the model. The non-linearity is only expected to increase in three-dimensions and with more complex physics (multiphase, multicomponent, etc.).

Compute-bound vs. memory-bound

Studies on sophisticated standard LB implementations based on D2Q9 lattice have revealed that the overall LB algorithm is strongly *memory-bound*, primarily due to the streaming step [81, 125]. For a problem of a given size, a subroutine or algorithm is termed memory-bound, if the total compute-time is limited more by the memory bandwidth rather than the CPU clock-speed. Similar observations in the higher-order lattices context were made in [91], who concluded that the standard LB algorithm with D3Q39 is 'highly bandwidth limited'. An important consequence of an algorithm being memory-bound is that, it limits its ability to scale on large-scale computing platforms, due to their low parallel (hardware) efficiency. Determining whether a code or algorithm is CPU or memory-bound requires detailed code profiling, including keeping track of counts of memory accesses, cache misses, and floating-point operations, etc. This process, which is hardware-dependent, can get very complex and requires sophisticated profiling tools, such as valgrind, Intel Performance Counter monitor, etc. Hence, we discuss and compare only the qualitative aspects of the nature of the FDLB code.

The high-memory-bandwidth of streaming in standard LBM comes the fact that, for updating each node in the grid, we need to read/load from eight neighboring distributions, in the case of the D2Q9 lattice. Similarly, thirty-six neighboring distributions are required for updating a node with the D2Q37 lattice. Larger neighborhood of the higher-order lattices becomes even more severe when the single node LB program is extrapolated to multiple nodes, where data transfer on the interconnect is slower compared to the SMP/NUMA bus.

A FDLB algorithm, on the other hand, has a smaller spatial neighborhood compared to standard LB for the same lattice. For example, for D2Q9, the `gradient()` at each node has a four node (two each in x and y coordinates) neighborhood dependency compared to the eight node neighborhood in standard LBM's `streaming()`. More importantly, due to the nature of the FDLB algorithm, the four neighborhood dependency remains the *same* even with higher-order lattices, such as D2Q17 or D2Q37; while in the standard LBM the dependency increases to sixteen and thirty-six, respectively. It should be noted, however, that in the FDLB case, the amount of data to be transferred for each update of the node remains the same as standard LB, but the data is *sourced more closely* compared to std.

LBM. The number of memory accesses are the same in terms of bytes of memory but they are accessed closer in FDLB compared to standard LBM. In other words, the flop count for the advection part increases, but, the memory bandwidth decreases. The same reasoning is applicable to three-dimensional problems.

We probe this aspect qualitatively. Tab. 6 shows the number of floating-point operations required in the FDLB algorithm and standard LB with different lattices, for a particular code implementation. Clearly, there is an increase in Flops in FDLB due to the addition of the `gradient()` and `update()`, which are not present in standard LBM. However, the increase in Flops is sourced more closely in FDLB compared to the standard LBM. It should be noted that the estimates provided in Tab.6 are highly implementation and compiler dependent; but an inference on the qualitative trend can be extended to the general case. Therefore, we can conclude that the combination of the FDLB algorithm and higher-order lattices appears to be much *less memory-bound* compared to standard LBM. This can be an important factor to consider for the scalability of LB simulations on large-scale parallel hardware. However, much more rigorous testing is required before we can draw this conclusion with certainty. The investigation in this regard is left for future work.

Table 6: Estimated number of FLOPs. $\mathcal{O}(\boldsymbol{u}^2, \theta)$ equilibrium is considered for D2Q9, $\mathcal{O}(\boldsymbol{u}^3, \theta)$ for D2Q12 and D2Q17, and $\mathcal{O}(\boldsymbol{u}^3, \theta)$ for D2Q37. The discrete forms of equilibrium are as listed in Chap. 3.

	D2Q9	**D2Q12**	**D2Q17**	**D2Q37**
`collision()`	180	360	510	2970
`gradient() and update()`	770	1020	1445	3150

6.4 SHARED-MEMORY PARALLELIZATION

The previous section detailed the steps taken to optimize a FDLB code on single-core cache-based CPU. In the following section, the same optimized implementation is extended for shared-memory machines, in order to take advantage of the hardware parallelism available through multicore CPUs.

6.4.1 Introduction

The ubiquitous presence of multicore CPUs have put a tremendous amount of computing power even at an desktop level. As described in Sec. 6.1.1.1, even on a larger scale, many HPC nodes are also based on COTS multicore CPUs [52, 126]. From a programming perspective, multicore CPUs on a workstation, and CPUs within a HPC node, constitute a shared-memory parallel (SMP) machine. The programming paradigm used to exploit the parallelism of SMPs is called shared-memory programming. Among the many SMP programming APIs, such as PThreads, CILK, TBB, etc., OpenMP is the most popular in the scientific computing community. It is noted that compiler based shared-memory parallelization exists, but their performance is much inferior to hand-tuned parallelization using OpenMP and others.

6.4.1.1 OpenMP From an ease of programming and time-to-results perspective, OpenMP offers many advantages [83]. One of the most important advantages is that OpenMP allows a developer easy access to all data from all cores without having to perform explicit communication. Due to the transparent data availability, an application can be easily parallelized to take advantage of the several cores available. This is unlike distributed-memory or GPU based programming, where the task of communicating and transferring data between CPUs across the nodes is explicit, and hence, extensive code modifications and restructuring are required.

Another useful feature of OpenMP is the fact that it allows *incremental parallelism.* That is, a developer can identify or profile the most time-consuming section in the program, and attempt to parallelize just that through OpenMP's parallel directives. The regions without parallel directives run sequentially, as usual. The program can then be tested to validate correctness and measure performance. If a sufficient speedup has been achieved, additional loops can be parallelized further. In fact, often, significant parallelism can be achieved through just three or four parallel directives.

Due to the advantages mentioned above, shared-memory parallelism through OpenMP presents an attractive, and often, *optimal choice for small and medium sized scientific problems*. OpenMP serves as good first-attempt to evaluate the suitability of an algorithm to scale on a few cores/CPUs, before efforts are put to scale it to much larger number of CPUs. Shared-memory parallelism is also often the only performance enhancement option, if only cheaper computing resources (desktops and workstations) are available.

6.4.1.2 OpenMP Details OpenMP consists of compiler directives, runtime library routines, and environment variables, on top of compiled languages of Fortran and C/C++ - all designed to take the burden of low-level details away from the application developed. Several OpenMP compilers are available and several x-86 based architectures support it. The detailed OpenMP API details can be found in [83] and Chapman *et al.* [23] provides in-depth implementation details and techniques.

OpenMP is based on the concept of threads. A thread, is the smallest independent unit of processing that can be scheduled by an operating system. All threads share a common memory address space and mutually accessible data, but they can also have their own user-defined private variables. Simply put, threads represents workers who share work by working independently and in parallel. The exact number of threads in the team determined by several ways. A common method is the '*one thread per core'* principle.

For parallelizing a program, a developer identifies regions in the code where work can be divided and run concurrently. This is done by inserting compiler (parallel) directives (e.g., `pragma` in C/C++) in the code in the appropriate sections. During run-time, sections without parallel directives run in serial, the serial part being executed by a master threads. Regions where parallel directive are encountered, a team of threads perform the work in parallel. This technique is also called as the 'fork-join' method.

While OpenMP is relatively simpler to implement, there are some important issues that need to be adressed. Important among them are the concepts of *race condition* and *load balancing*. Race condition occurs when two threads access the same variable at the exact time and at least one of them does a write operation. Load balancing is the concept that the total work is shared *equally* among all the threads such that no thread(s) is over-burdened, thereby affecting the program run-time adversely.

6.4.2 Implementation Details

The OpenMP extension for the higher order FDLB program is primarily built on the previous described cache-optimized FDLB code. As described before, most of the computationally intensive regions of the FDLB code are multiple *for* loops. Therefore, only one OpenMP directive, namely `omp parallel for` and its various clauses, are used extensively in this work. A few other directives are used for synchronization and some other functions. Domain decomposition model is used to partition (divide) the computational work. That is, the physical domain is implicitly sub-divided into number of sub-domains, and the data associated with each sub-domain is handled independently by each thread (core).

Creation, maintenance, and destruction of threads has computational overheads. Therefore, the most optimal approach to parallelizing a higher-order FDLB algorithm with OpenMP is to create threads at the start of the main (temporal) loop using the `#pragma parallel` construct. An overall structure of the approach is given in the code snippet below:

```
#pragma omp parallel
{
   while (convergence_criteria < epsilon or iter< maxIter)
   {
      collision();       /* Work sharing loop 1 */
      gradient_update();    /* Work sharing loop 2 */
      macro_variables();    /* Work sharing loop 3 */
   #pragma omp barrier
   check_convergence();
   iter++
   }
}
   #pragma master
   file_io()             /* I/O operation
```

This way, the various threads are spawned and destroyed only once during the entire program. For each time-iteration, each thread executes the `collision()`, `gradient()` and `update()` for a certain fixed number of nodes that are assigned to it. At the end of the each time-iteration, the threads synchronize with each other through a `#pragma omp barrier`. At the end of the time loop (or after convergence), all of the threads join into the master thread through `#pragma master`, which performs file input/output and other tasks before exiting the program.

Unfortunately, this approach cannot be implemented due to restrictions of the OpenMP's `parallel` construct which allows only one `if` clause within its scope [23]. `if` clauses are needed in each function to deal with generic fluid and boundary scenarios. It is used to check, through a Boolean variable `(is_a_fluid)`, if the node is a fluid node or solid/wall/obstacle node.

Therefore, the alternative pursued in this work is to create threads at the start of each subroutine. The essence of this approach is shown in the code fragment below:

```
while (convergence_criteria < epsilon or iter< maxIter)
   {
      #pragma omp for
      collision();       /* Work sharing loop 1 */
      #pragma omp for
      gradient_update();    /* Work sharing loop 2 */
      #pragma omp for
      macro_variables();    /* Work sharing loop 3 */
   #pragma omp barrier
   convergence_criterion();
   iter++
   }
   #pragma master
   file_io()            /* I/O operation
```

In this approach, we can see that, for each time iteration, threads are created and destroyed several times. It is noted that, for simpler flow conditions/geometries, the `if` clause and `is_a_fluid` can be done away with to gain further efficiency. However, in this work, the `if` clauses are maintained.

For synchronization of all the threads before the next time iteration, the `omp barrier` directive is used at the end of each time iteration. In our case, since the domain considered is Cartesian and structured, load balancing on individual threads is equal and automatic. That is, load balancing is achieved implicitly by sharing the iteration space equally among various threads. The amount of work done by each thread at each node is equal and hence load is balanced. At the end of time iterations, `omp master` is used to perform file I/O operation. A brief description of the OpenMP implementation of the two major subroutines in the FDLB algorithm is provided below.

collision()

This is perhaps the simplest subroutine to parallelize using OpenMP. The pseudocode below shows the OpenMP implementation of `collision()`.

```
#pragma omp parallel for collapse(4) schedule (static) private (edotu,edotuSq,
    uSq,eSq)
for (int k=0; k<NV; k++)   {
   for (int y=0; y<NY; y++)    {
      for (int x=0; x<NX; x++)        {
            if (isFluid(x,y))==1 {
            edotu = ex(k)*ux(x,y) + ey(k)*uy(x,y);
            edotuSq = edotu*edotu;
            uSq = ux(x,y)*ux(x,y) + uy(x,y)*ux(x,y);
            eSq = ex(k)*ex(k) + ey(k)*ey(k);
            feq(x,y,k) = W(k)*density(x,y)* (1.0 + edotu + 0.5*(edotuSq-uSq +
    ...
         }
      }
   }
}
```

As mentioned before, `EDF()` is fully local, which means that to compute `feq(i,j)`, we need read and write various variables at the same local `(i,j)` and not its neighbors. This property precludes the need for communication (data transfer/synchronization) with other threads, even at the boundaries. Locality of computation also implies that no two threads will read or write an element in the array at the same time, thereby the race condition is avoided. As previously mentioned, since the load is balanced, the `static` clause is used to distribute the work among threads. Also, since `EDF()` has perfectly nested rectangular loops with no dependencies, the `collapse` clause is used which leads to better efficiency. The task of computing hydrodynamic variables is also a fully local function, and hence techniques used for `EDF()` are also applied to `macro_variables()`.

gradient_update()

A simplified code snippet of the OpenMP implementation of the combined `gradient()` and `update()` is shown below. Most of the features of the `gradient_update()` as described in Sec.6.3.4 are retained. The only difference here is that individual loops for each velocity (direction) are not implemented here. Individual loops cause repeated creation and destruction of threads, thereby increasing the overheads; instead one large nested loop covering all directions and space is implemented.

```
#pragma omp for collapse(4) schedule (static) private (h)
for (int k=0;k<NV;k++)  {
   for (int y=0; y<NY; y++)    {
      for (int x=0; x<NX; x++)  {
         if (isFluid)==1         {
               h = f(x,y,k) - feq(x,y,k);
               /* Compute first-order spatial-derivatives of f*/
               /* Compute second-order spatial-derivatives of f*/
               /* update f(x,y,k)*/
         }
      }
   }
}
```

6.4.3 Results

Test Setup

The OpenMP-based FDLB program is benchmarked for its performance and scalability on the Center for Simulation and Modeling (SAM)'s 'Frank' cluster at the University of Pittsburgh. Frank's hardware consists of over 7800 cores spread over 384 nodes of both shared-memory and distributed-memory configuration, all connected via a Infiniband network. A variety of CPU speeds and socket specifications are available, all running Red Hat Enterprise Linux 6. See [20] for a complete details of the cluster. In this work, however, a shared-memory cluster named *shared* is used. Each *shared* node has four sockets; with each socket containing a 12-core AMD Magny Cours (6172) 2.1 GHz CPU. Since we cannot utilize cores across the node in a shared–memory configuration, only a maximum of 48 cores are available for testing. On the software side, the `gcc` compiler is used with `-fopenp` flag along with the `O2` optimization level. The number of threads (and therefore cores) in the program is specified through the `OMP_NUM_THREADS` environment variable.

The OpenMP - FDLBM code is first verified for its accuracy. Velocity profiles identical to Figs. 7 and 8 were obtained from the cluster run; which proves that parallel simulations results are the as accurate as the serial one. The testing procedure adopts a *strong-scaling* approach, where the amount of computational work is fixed, and deceasing run-times are sought by OpenMP based parallelism. Both D2Q9 and D2Q17 lattices are tested. The temporal and spatial discretization and other simulation details are the same as described in Sec. 6.3.5.1.

Performance Metrics

In parallel computing, performance improvement is a subjective term. To quantify objectively the performance of the OpenMP-based FDLBM program, two related performance metrics are used here - parallel scale-up (S), and parallel efficiency (ϵ). The scale-up is relevant in the strong-scaling context to quantify the reduction in computational time resulting from running the a program on N cores instead of one core, for a fixed amount of computational work. It is defined as follows:

$$S = \frac{wall - time\, on\, one\, core}{wall - time\, on\, N\, cores}. \tag{6.2}$$

Scale-up is useful to answer the question: everything remaining the same, how fast does a program run on N cores versus on one core. Parallel efficiency, on the other hand, quantifies how efficiently the resources (N cores) are utilized by the parallelized program. It is defined as:

$$\epsilon = \frac{wall - time\, on\, N\, cores}{N \times wall - time\, on\, one\, core} = \frac{speedup\, on\, N\, cores}{N} \tag{6.3}$$

It is clear that an ideal (perfect) parallel program should have $\epsilon \to 1$. It is noted that the wall-times used in the analysis consists purely of the time taken to run the core FDLB algorithm (`collision()`, `gradient_update()`, `macro_variable()`), and therefore does not include ancillary tasks, such as file I/O, initialization, etc. The wall-times are measured using `OMP_GET_WTIME()`.

6.4.3.1 Observations Fig. 39 shows the speedup curve of the OpenMP-FDLB code. The horizontal axis shows the number of cores (threads) utilized and the vertical axis shows the corresponding speedup, as obtained from Eq. (6.2).

Because of strong scaling, the execution time decreases with an increasing number of cores, and hence the speedup increases with the number of cores. The speedup for the D2Q17 lattice is also (slightly) better than the D2Q9. This is because, in case of D2Q17, the number of computations per node is much higher than that for D2Q9. This leads to a higher computation to communication ratio (CCR) in D2Q17 and hence the better speedup. Tests with the D2Q37 lattice are the subject of future investigations.

However, as the number of cores is increased, the speedup starts to deviate from the ideal (linear) curve. This results in a lower parallel efficiency as indicated in Fig. 40. The less than ideal speedup is possibly due to the overheads associated with the creation and destruction of an increasing number of threads at every time-step. OpenMP has higher overheads compared to other threads based tools such as PThreads. The cc-NUMA architecture of the cluster also adds to the overheads. This is because, to maintain cache-coherency, additional protocols are required to be maintained, adding to the overall overheads [23, 52]. It is expected that the impact of all the overheads on total computational time is expected to become smaller, if the computational grid in the domain is high, and if the number of threads is not too high.

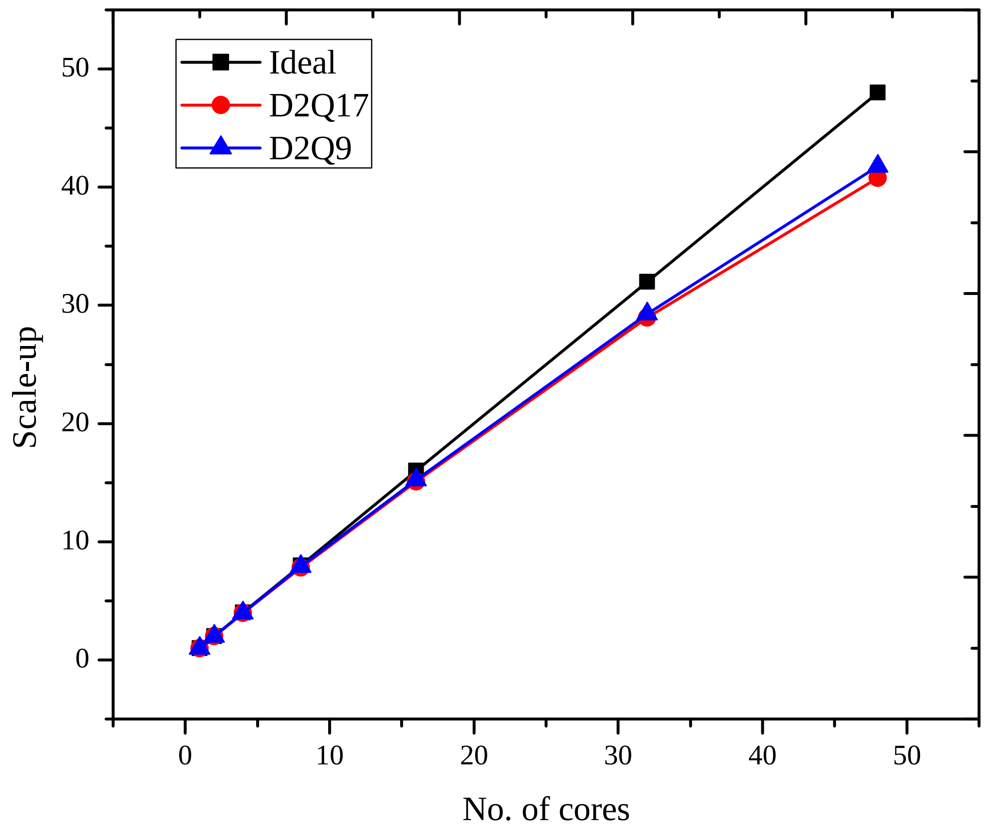


Figure 39: Scale-up obtained through OpenMP based parallelism of a OLB code. The computational domain is 513^2 and $20,000$ time-iterations are performed.

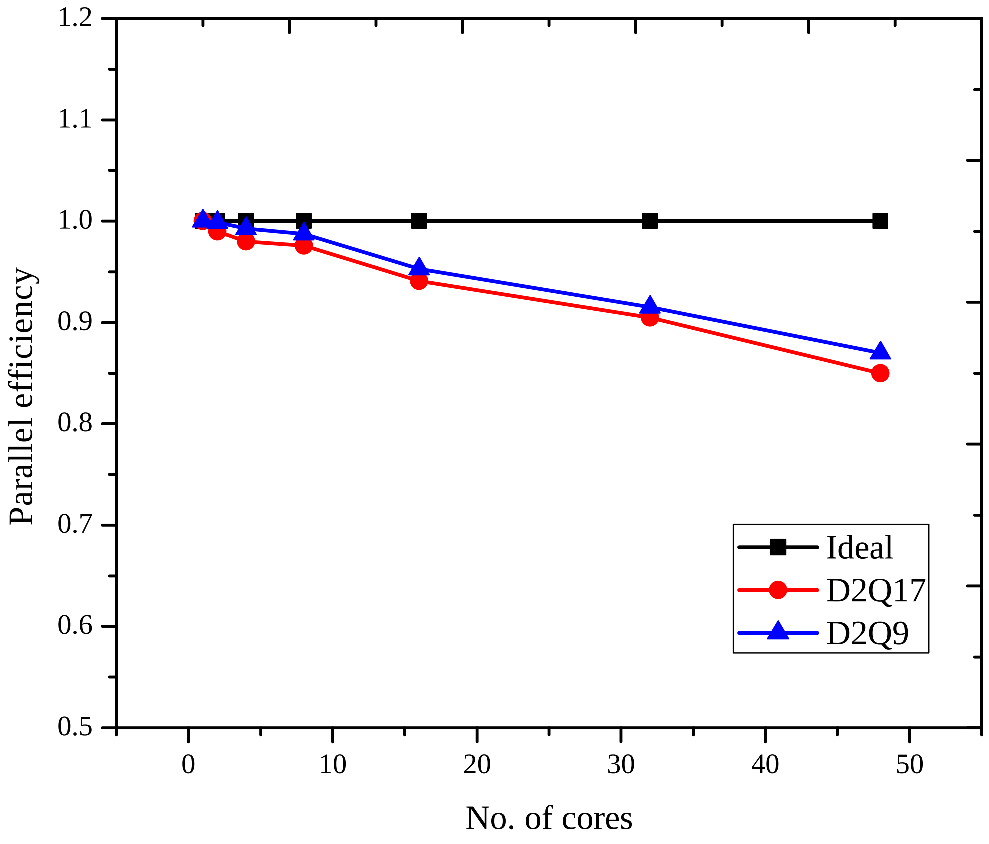


Figure 40: Parallel efficiency for the OpenMP-FDLB code .

6.5 DISCUSSION

The work presented here is exploratory in nature, developed primarily as a tool to investigate the several OLB schemes and higher-order lattices for thermal problems. Still, many important conclusions can be drawn from the results presented here. They are:

1. The SoA data structure is an efficient choice for storing the fundamental variable (distributions). Through this data structure and optimal looping, good performance enhancement for research codes can be obtained even on singlecore desktops.
2. The computational effort of higher-order lattices appears to increases non-linearly with the increase in number of velocities in the lattice. The increase is mainly from the non-linear increase in the computational complexity involved in evaluating the equilibrium distributions.
3. The FDLB method might be less memory-bound for higher-order lattices, compared to stream-collide type schemes. This feature might aid better scaling of LB applications on a larger number of cores.
4. Shared-memory parallelization of FDLB simulations shows good scalability (with a parallel efficiency of 0.8) on systems with up to 50 cores. For better scalability and higher *sustained performance* in GFlop/s, other platforms might present better results.
5. More memory-efficient data structures should be developed to facilitate a wider use of higher-order lattices for complex physics.
6. Benefits of optimization and parallelization are relatively larger for bigger grids compared to smaller grids due to a larger computation to communication ratio.

7.0 CONCLUSIONS AND OUTLOOK

The primary focus of this dissertation was to investigate the theoretical and numerical aspects of the higher-order lattice Boltzmann models for thermal flows and their implementations in an off-lattice framework. In this regard, we have made several contributions, a summary of which is given below.

7.1 PRIMARY CONTRIBUTIONS

7.1.1 Kinetically consistent thermal LB model

Based on the framework first presented by Grad and later extended by Shan, Chen, and colleagues [45, 105], we presented a detailed mathematical derivation of a kinetically consistent, energy conserving, thermal lattice Boltzmann model. This formulation, obtained directly from the Boltzmann equation, was shown to be capable of approximating the weakly compressible, and compressible thermal behavior of ideal-gas hydrodynamics, thus obviating the previous low-Mach number assumption. Physically, it means that this formulation allows a kinetically-accurate description of flows involving high heat transfer and/or flows at finite Mach number. In an off-lattice framework, the thermal model, consisting of the D2Q39 lattice and fourth-order Maxwellian, was validated for the Rayleigh-Benard convection and the thermal Couette flow. Our tests showed that the D2Q39-based thermal model is capable of modeling incompressible and weakly compressible thermal flows accurately. In the validation process, through a finite-difference-type boundary treatment, we also extended the applicability of higher-order lattices to domains with solid boundaries, previously restricted.

7.1.2 Off-lattice Boltzmann schemes

Various time-marching schemes for solving the discrete-velocity Boltzmann equation were presented and analyzed in detail. These schemes are potentially useful for extending the geometric flexibility of the LB method for body-fitted grids, and also for use with higher-order discrete velocity sets-both space-filling or non space-filling. Towards that end, the various time-evolution schemes were analyzed and their numerical stability a function of maximum allowable Δt was studied in detail. We showed that the characteristic-based scheme of Bardow *et al [11].* offers the best numerical stability among comparable time-marching schemes. Due to the enhanced numerical stability of the characteristics-based scheme, we found that the $\Delta t < \tau$ restriction no longer applies, enabling larger time-steps, and thereby reducing the computational run-time. The off-lattice schemes were also successfully extended to higher-order LB models.

7.1.3 Serial and Parallel Implementation of higher-order models

We presented the algorithm and single-core optimization techniques for a off-lattice, higher-order LB code. Using simple cache optimization techniques and a proper choice of the data-structures, we obtain a 5-7X improvement in performance compared to a naive, unoptimized code. Thereafter, the optimized code was parallelized using OpenMP. Scalability tests indicated that the code has a parallel efficiency of 80% on shared-memory systems with up to 50 cores. A preliminary analysis of the higher-order models also showed that they are *less memory-bound* if the off-lattice temporal schemes are used instead of the regular stream-collide scheme.

7.2 FUTURE WORK

As with most research, answers to certain question raise newer ones. This work is no exception. Therefore, a few of the areas specifically related to the work presented here, that need future in-depth investigation and analysis are identified and described below.

7.2.1 Quantitative assessment of thermal model

The validation performed for the thermal models in Chap. 5 was done on a limited parameter range and in the incompressible limit. Future tests should rigorously test the D2Q39-based model by performing parametric tests to gauge the effects of higher temperature gradients, addition of heat sources, smaller relaxation times, non-Cartesian geometry, etc., on the numerical stability and accuracy of the model, and in both two and three dimensions. Similarly, rigorous benchmark tests should be performed to quantify the accuracy of using the finite-difference type boundary treatment with higher-order lattices. Finally, to employ the real power of the Hermite-based LB formulations, the D2Q39-based model should be tested for high Mach number and high Rayleigh number flows. For fully compressible flows of real gases, where $Pr \neq 1$, the multiple-relaxation time collision model, developed in the Hermite-polynomial framework, can be used. Moreover, the thermal model developed here still has fixed specific heat ratio. Therefore, for the model to be able to simulate fluids with a variable specific heat ratio, which play an important role in compressible flows, it has to include the effects of the gases' internal degrees of freedom.

7.2.2 Boundary Treatment

One of the most important advantages of LB methods, the exact space discretization of particle's advection (no dispersion or diffusion) and algorithmic simplicity, both are lost by using off-lattice time marching schemes. Clearly, the stream-collide scheme is more desirable both for accuracy and efficiency reasons. One of the major impediments to a wider adaptation of the higher-order lattices is the lack of the simple and local bounceback type wall boundary treatment that is employed in conjunction with the stream-collide scheme. In the absence of such a boundary scheme, the off-lattice temporal schemes, despite all their disadvantages, offer an attractive option for employing the higher-order lattices with non-periodic boundaries. Hence, schemes analogous to bounceback should be developed for the higher-order LB models. Similarly, treatment of curved boundaries in the higher-order lattice context also needs to be addressed. In general, the volumetric LB formulation might be better suited for higher-order LB models with non-periodic boundaries than point-wise (stream-collide) formulation. Advanced, numerical techniques, such as local or adaptive grid refinement, also become more difficult to implement for higher-order models due to their extended neighborhood.

7.2.3 Interpolation versus off-lattice Boltzmann schemes

The interpolation-supplemented LB and off-lattice Boltzmann are both schemes that were developed to address the issue of breaking the space-time coupling. While, the former introduces errors due to interpolation, the latter (as shown in this work) introduces errors due to numerical diffusion. Therefore, their comparative study would be useful for future investigation of non-space-filling lattices.

7.2.4 Third-generation LB method

Over the past decade, alternate LB formulations with different collision models, such as the multiple-relaxation time (MRT) and entropic models have been developed to address the deficiencies of the BGK model. Of the two, the entropic LB formulation is formulated to describe the thermo-hydrodynamics in a kinetically-consistent fashion, like the Hermite based LB formulation. Therefore, a extremely pertinent study for future work would be to conduct a comprehensive comparison of the two formulations with respect to their stability, accuracy, computational efficiency, parameter range, etc.

7.2.5 Computational Advancement

The field of high performance computing is a rapidly evolving field, where every three to five years there are substantial changes in HPC architecture. Since software generally outlasts hardware, any new scientific software development should ideally be done with future hardware trends in mind. However, historically, software development has lagged behind the hardware advances, and this issue persists even today. Many traditional engineering software applications are not parallelized and many of those that are parallelized do not scale well beyond 50-100 cores. Optimization, tuning and parallelization of an application needs a careful map of the algorithm to the platform to utilize every bit of the performance and parallelism, which required expertise and man-hours. More specifically, in the off-lattice, higher-order models context, some of the avenues for future research include:

1. *Data structures*: Customized (as opposed to arrays) data-structures and their associated algorithms should be designed for higher-order LB simulations. A particular concern that needs to be addressed is the larger neighborhood of the higher-order models.

2. *Detailed profiling*: A detailed serial and parallel profiling of the off-lattice, higher-order LB code should give us additional insights and opportunities to optimize the algorithm for different platforms. Advanced debugging and profiling tools such as GProf, PAPI, TotalView, etc., can be useful for this purpose.

3. *Memory vs. CPU-bound analysis:* Quantitative investigation of memory bound/CPU-bound nature of the FDLB method. Such an investigation is important for future scaling studies of the higher-order method.

4. *Distributed-memory platform*: The shared-memory parallelization, due to hardware limitations, cannot be scaled beyond 50-100 cores. Truly large-scale higher-order LB codes would need to be parallelized using MPI. In the higher-order LB context, domain decomposition and mapping of complex domains onto equally-loaded processors is a challenge.

5. *Coprocessors*: Coprocessors, such as GPUs, and MICs (Intel's Xeon Phi), are becoming more common in the high-end computing sphere. Several GPU based LB implementations have been implemented with success, while MIC-based implementations have just begun to appear. An interesting debate has begun in the HPC field comparing the efficacy of GPUs versus MICs for parallelization, along with different metrics, such as price to performance, time to deployment, sustained vs. peak performance, etc. A comparison of this sort in the LBM context would be useful.

7.3 OUTLOOK

7.3.1 Industrial Lattice Boltzmann

It can argued that, barring the exception of Exa Corporation, the LB method has been mostly confined to academic circles. This is in spite of its many advantages- computational and modeling- over traditional CFD. The fact that a single, self-contained methodology, that can deal with large variety of complex multi-physics flows, from high-speed aerodynamics to bio/medical physics, in itself is significant! In the author's opinion, this versatility, in a way, is also an impediment. Due to a first-mover advantage for the traditional PDE-based CFD, and a background in physics (kinetic theory) rather than engineering, the LB method, is often looked upon with skepticism in CFD community with respect to its capabilities. The perception is that the LB method can perform complex physics on simple geometries, but lacks the ability/experience to perform complex physics on complex (non-Cartesian) engineering-scale geometries. Hence, there is a clear need for the LB method to tackle large, three-dimensional, multi-physics type industrial applications, especially those where traditional PDE-based solvers are inadequate and where the LB method has its strengths. An example of such an application can be a high-fidelity, boiling simulation in nuclear reactors, multiphase, multicomponent flow in microfluidic devices, simulation of multiphase (particle-laden) flow in fluidized beds, etc. Such high-profile, yet rigorous, studies would highlight the strengths of the LB method, while offering paths for further model development. To further highlight its capabilities, it is also useful for LB researchers (at least in the U.S.) to present their findings in traditional CFD arenas, such as the AIAA, ASME and APS conferences.

On a less technical point, the author believes that a conceptual barrier exists for the wide-spread adoption of the LB method not only in the industry, but even in engineering (mechanical/aerospace) departments. While PDE-based CFD methods have their basis in conceptually simpler conservation laws, the steep learning curve of (kinetic theory) basis of the LB theory makes it more appealing for physicists rather than practicing engineers.

7.3.2 Multidisciplinary LB software development

Ease of programming of the algorithm, simple data-structures, and the redundancy of grid generation, have played an important role in the growing popularity of the LB method, as evidenced in the increasing publications in the field. Indeed, a full-fledged LB code can be written in less than 2-3 pages of code, even for a fairly complex flow. A majority of these studies (including those of the author) are geared towards method/physics development, as opposed to real-world applications. However, the true impact of the method on engineering, design, and ultimately, everyday human life depends upon a collaborative effort between software developers (including HPC and visualization experts), LB model developers, and practicing (design) engineers.

Most of the LB methods development, including what has been done by this author, happens in a *'one code per student'* basis. This is also referred as the *'not invented here syndrome'*. While this approach is useful for a complete and thorough understand of the elements inside a LB algorithm/code, this practice is not scalable to address large, engineering-scale flow problems as pointed out earlier. To do this, we need to leverage some rigorously validated open-source codes not just for large projects, but even for (smaller) new model development. Although the learning curve is higher, significantly more complex problems can be solved accurately, by using open-source codes. These codes often include highly optimized data-structures algorithms and software engineering practices, through which a particular new physics model can be developed and tested quickly, and on large computational domains.

Finally, a word about the extent to which a LB code can be optimized, tuned, and parallelized for a particular platform is in order. Different hardware platforms have their own advantages/disadvantages, depending upon the metric under consideration. For example, a shared-memory platform enables relatively quick software development, but is not scalable to a large number of cores, and has a low sustained-to-peak performance ratio. On the other hand, for the LB method, vector processors are much better than cache-based systems, in terms of *sustained performance* in GFlop/s. Given the decline of vector computers, the only way to bridge the sustained-peak performance gap, and attain scalable petascale or even exascale level LB simulations, is to perform further low-level, hardware-specific optimizations, such as SIMDization, software pre-fetching, etc. Given this fact, the question arises: How much optimization is enough? Designing such highly efficient software requires hardware, software and numerics expertise, along with time and access to HPC resources. Highly hardware-specific tuning hinders generic, portable software, that can run on desktops or large HPC clusters. Clearly, future LB software development would be a process that will involve numerous trade-offs.

APPENDIX A

A PRIMER ON HERMITE POLYNOMIALS

In this appendix we will discuss different properties of the Hermite polynomials that are relevant for the LB method. The results presented here are a summary of the work by Harold Grad [44].

A.1 INTRODUCTION

Hermite polynomials are a type of classical *orthogonal polynomial* sequence, like the Chebyshev or Legendre polynomials. The Hermite polynomials are orthogonal on the interval $(-\infty, \infty)$ with respect to the weight function $\omega(x)$. They can be defined by means of their Rodrigues formula. Explicitly, in 1D we have:

$$\mathcal{H}_n(x) = \frac{(-1)^n}{\omega(x)} \frac{d^n \omega(x)}{dx^n}, \quad n = 0, 1, 2, \cdots, -\infty < x < \infty \tag{A.1}$$

where the weight function is

$$\omega(x) = \exp\left[-\frac{x^2}{2}\right] \tag{A.2}$$

Hermite polynomials form a complete orthogonal set on the interval $-\infty < x < \infty$ with respect to $\omega(x)$:

$$\int_{-\infty}^{\infty} \omega(x) \mathcal{H}_n(x) \mathcal{H}_m(x) dx = n! \delta_{nm} \tag{A.3}$$

and generalized Fourier expansion:

$$f(x) = \omega(x) \sum_{n=0}^{\infty} \frac{1}{n!} a_n \mathcal{H}_n(x) \quad \text{where} \quad a^{(n)} = \int_{-\infty}^{\infty} f(x) \mathcal{H}_n(x) dx \tag{A.4}$$

$a^{(n)}$ is the spectral expansion coefficient. A complete list of properties of Hermite-polynomials can be found in [3] and [1].

A.2 HERMITE POLYNOMIALS IN VELOCITY SPACE

Per the Rodrigues' formula, in a D-dimensional Cartesian space ($\boldsymbol{\xi}$), the Hermite polynomial of degree n is defined by the n^{th} derivative of the weight function $\omega(\boldsymbol{\xi})$, which is given as:

$$\omega(\boldsymbol{\xi}) = \frac{1}{(\sqrt{2\pi})^D} \exp\left[-\frac{\xi^2}{2}\right], \tag{A.5}$$

where $\xi^2 = \boldsymbol{\xi} \cdot \boldsymbol{\xi}$. Therefore, we obtain the Hermite polynomial of degree n as a n^{th} rank symmetric tensor whose components are:

$$\boldsymbol{H}^{(n)}_{\alpha_1 \cdots \alpha_n}(\boldsymbol{\xi}) = \frac{(-1)^n}{\omega(\boldsymbol{\xi})} \frac{\partial}{\partial \xi_{\boldsymbol{\alpha}}} = \frac{(-1)^n}{\omega(\boldsymbol{\xi})} \frac{\partial}{\partial \xi_{\alpha_1}} \cdots \frac{\partial \omega(\boldsymbol{\xi})}{\partial \xi_{\alpha_n}} \tag{A.6}$$

where $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ and $\alpha_i = 1, 2, ..., D$. In the tensor notation, we have:

$$\boldsymbol{H}^{(n)}(\boldsymbol{\xi}) = \frac{(-1)^n}{\omega(\boldsymbol{\xi})} \nabla^n \omega(\boldsymbol{\xi}) \tag{A.7}$$

In this work, the first five Hermite polynomials are useful. They are given explicitly as:

$$\begin{aligned} \boldsymbol{H}^{(0)}(\boldsymbol{\xi}) = &\ \tfrac{(-1)^0}{\omega(\xi)} \omega(\boldsymbol{\xi}) = &\ 1, \\ \boldsymbol{H}^{(1)}_{\alpha}(\boldsymbol{\xi}) = &\ \tfrac{(-1)^1}{\omega(\boldsymbol{\xi})} \tfrac{\partial \omega(\boldsymbol{\xi})}{\partial \xi_\alpha} = &\ \xi_\alpha, \end{aligned}$$

$$\mathcal{H}^{(2)}_{\alpha 1 \alpha 2}(\boldsymbol{\xi}) = \frac{(-1)^2}{\omega(\boldsymbol{\xi})} \frac{\partial^2 \omega(\boldsymbol{\xi})}{\partial \xi_{\alpha 1} \xi_{\alpha 2}} = \xi_{\alpha 1} \xi_{\alpha 1} - \delta_{\alpha 1 \alpha 2}, \tag{A.8}$$

$$\mathcal{H}^{(3)}_{\alpha 1 \alpha 2 \alpha 3}(\boldsymbol{\xi}) = \frac{(-1)^3}{\omega(\boldsymbol{\xi})} \frac{\partial^3 \omega(\boldsymbol{\xi})}{\partial \xi_{\alpha 1} \xi_{\alpha 2} \xi_{\alpha 3}} = \xi_{\alpha 1} \xi_{\alpha 2} \xi_{\alpha 3} - \big(\delta_{\alpha 1 \alpha 2} \xi_{\alpha 3} + \delta_{\alpha 1 \alpha 3} \xi_{\alpha 2} + \delta_{\alpha 2 \alpha 2} \xi_{\alpha 1}\big),$$

$$
\begin{aligned}
\mathcal{H}^{(4)}_{\alpha1\alpha2\alpha3\alpha4}(\boldsymbol{\xi}) &= \frac{(-1)^4}{\omega(\boldsymbol{\xi})}\frac{\partial^4\omega(\boldsymbol{\xi})}{\partial\xi_{\alpha1}\xi_{\alpha2}\xi_{\alpha3}\xi_{\alpha4}} = \xi_{\alpha1}\xi_{\alpha2}\xi_{\alpha3}\xi_{\alpha4} - \Bigg(\delta_{\alpha1\alpha2}\xi_{\alpha3}\xi_{\alpha4} + \delta_{\alpha1\alpha3}\xi_{\alpha2}\xi_{\alpha4} + \delta_{\alpha1\alpha4}\xi_{\alpha2}\xi_{\alpha3} \\
&+ \delta_{\alpha2\alpha3}\xi_{\alpha1}\xi_{\alpha4} + \delta_{\alpha2\alpha4}\xi_{\alpha1}\xi_{\alpha3} + \delta_{\alpha3\alpha4}\xi_{\alpha2}\xi_{\alpha3} \\
&+ \delta_{\alpha1\alpha2}\delta_{\alpha3\alpha4} + \delta_{\alpha1\alpha3}\delta_{\alpha2\alpha4} + \delta_{\alpha1\alpha4}\delta_{\alpha2\alpha3}\Bigg).
\end{aligned}
$$

Here, $\delta_{\alpha1\alpha2}$ is the Kronecker delta function with value 1 if $\alpha1 = \alpha2$, and 0 otherwise. In a compact tensor notation, we have:

$$
\begin{aligned}
\boldsymbol{H}^{(0)}(\boldsymbol{\xi}) &= 1 \\
\boldsymbol{H}^{(1)}(\boldsymbol{\xi}) &= \boldsymbol{\xi} \\
\boldsymbol{H}^{(2)}(\boldsymbol{\xi}) &= \boldsymbol{\xi\xi} - \boldsymbol{\delta} \\
\boldsymbol{H}^{(3)}(\boldsymbol{\xi}) &= \boldsymbol{\xi\xi\xi} - 3\boldsymbol{\xi\delta} \\
\boldsymbol{H}^{(4)}(\boldsymbol{\xi}) &= \boldsymbol{\xi\xi\xi\xi} - 6\boldsymbol{\xi\xi\delta} + 3\boldsymbol{\delta}
\end{aligned}
\tag{A.9}
$$

A useful recurrence relation is given by:

$$
\boldsymbol{\xi H}(n) = \boldsymbol{H}^{(n+1)} + \nabla_{\boldsymbol{\xi}}(\boldsymbol{H}^{(n)}) \tag{A.10}
$$

Or explicitly:

$$
\begin{aligned}
\xi_0\mathcal{H}^{(n)}_{\alpha1,\alpha2,\cdots\alpha_n} &= \mathcal{H}^{(n)}_{\alpha0,\alpha1,\cdots\alpha n} + \partial_{\xi_{\alpha0}}\mathcal{H}^{(n)}_{\alpha1,\alpha2,\cdots\alpha n} \\
&= \mathcal{H}^{(n)}_{\alpha_0,\alpha_1,\cdots\alpha_n} + \sum_{i=1}^{n}\delta_{\alpha0\alpha i}\mathcal{H}^{(n-1)}_{\alpha1\cdots\alpha_{i-1}\alpha_{i+1}\cdots\alpha_n}
\end{aligned}
\tag{A.11}
$$

Another useful relation is for the derivative of Hermite polynomials with respect to velocity. In a compact form we have:

$$
\nabla_{\boldsymbol{\xi}}(\boldsymbol{H}^{(n)}) = n\boldsymbol{H}^{(n-1)}\boldsymbol{\delta} \tag{A.12}
$$

APPENDIX B

GAUSS-HERMITE QUADRATURE

The following tables compile the Gauss-Hermite quadrature set (weights and abscissae) in 2D and 3D for various degrees of quadrature, based on works of Shan *et al.* [105, 101] and Nie *et al.* [80]. Besides these, other two-dimensional quadrature set include the D2Q16 and D2Q25, which are based on the entropic LB formulation, and the D2V25 (W1) and D2V25 (W6) based on 'quadrature with prescribed abscissas.' It is noted that D2Q17 and D2Q37 presented here are the same as the D2V17 and D2V37 presented in [107, 115]. Also listed here the D2Q5 and D3Q7 lattices that do not have enough isotropy to sufficiently represent fluids, but are adequate for the advection–diffusion schemes. The subscript FS represents a fully symmetric set of velocities. Per the convention proposed by Shan *et al.* [105], the quadrature rule $E^d_{D,n}$ is defined in D space dimensions, is exact for a polynomial degree n or degree of precision of the quadrature, and has d points or quadrature nodes. c_l denotes the lattice constant.

2D 5th-order Quadrature						
Quadrature	**Lattice**	$\boldsymbol{\xi}_i$	p	w_i	**lattice constant**	**Remarks**
$E^7_{2,5}$	D2Q7	$(0,0)$	1	$\frac{1}{2}$		**Off-lattice**
		$2\cos\frac{n\pi}{3}, \sin\frac{n\pi}{3}$	6	$\frac{1}{12}$		
$E^9_{2,5}$	D2Q9	$(0,0)$	1	$4/9$	$c_l = \sqrt{3}$	**On-lattice**
		$(c_l,0)_{FS}$	4	$1/9$		
		$(\pm c_l, \pm c_l)$	4	$1/36$		

2D 7th-order Quadrature						
Quadrature	**Lattice**	$\boldsymbol{\xi}_i$	p	w_i	**lattice constant**	**Remark**
$E_{2,7}^{12}$		$(r,0)_{FS}$	4	$\frac{1}{36}$	$r^2=6$	**Off-lattice**
		$(\pm s,\pm s)$	4	$\frac{(5+2\sqrt{5})}{45}$	$s^2=\frac{(9-3\sqrt{5})}{4}$	
		$(\pm t,\pm t)$	4	$\frac{(5-2\sqrt{5})}{45}$	$t^2=\frac{(9+3\sqrt{5})}{4}$	
$E_{2,7}^{16}$		$(r,0)_{FS}$	4	$\frac{(5-2\sqrt{6})}{48}$	$r^2=3+\sqrt{6}$	**Off-lattice**
		$(s,0)_{FS}$	4	$\frac{(5+2\sqrt{6})}{48}$	$s^2=3-\sqrt{6}$	
		$(r,s)_{FS}$	8	$\frac{1}{48}$		
$E_{2,7}^{17}$	D2Q17	$(0,0)$	1	$\frac{(575+193\sqrt{193})}{8100}$	$c_l^2=\frac{(125+5\sqrt{193})}{72}$	**On-lattice**
		$(c_l,0)_{FS}$	4	$\frac{(3355-91\sqrt{193})}{18\,000}$		
		$(\pm c_l,\pm c_l)$	4	$\frac{(655+17\sqrt{193})}{27\,000}$		
		$(\pm 2c_l,\pm 2c_l)$	4	$\frac{(685-49\sqrt{193})}{54\,000}$		
		$(\pm 3c_l,0)_{FS}$	4	$\frac{(1445-101\sqrt{193})}{162\,000}$		
$E_{2,7}^{21}$	D2Q21	$(0,0)$	1	$\frac{91}{324}$	$c_l^2=\frac{3}{2}$	**On-lattice**
		$(c_l,0)_{FS}$	4	$\frac{1}{12}$		
		$(\pm c_l,\pm c_l)$	4	$\frac{2}{27}$		
		$(\pm 2c_l,0)_{FS}$	4	$\frac{7}{360}$		
		$(\pm 2c_l,\pm 2c_l)$	4	$\frac{1}{432}$		
		$(\pm 3c_l,0)_{FS}$	4	$\frac{1}{1620}$		

2D 9th-order Quadrature					
Quadrature	**Lattice**	$\boldsymbol{\xi}_i$	p	w_i	**Remark**
$E_{2,9}^{37}$	$D2Q37$	$(0,0)$	1	0.23315066913235250228650	**On-lattice**
		$(c_l,0)_{FS}$	4	0.10730609154221900241246	
		$(\pm c_l,\pm c_l)$	4	0.05766785988879488203006	
		$(2c_l,0)_{FS}$	4	0.01420821615845075026469	
		$(\pm 2c_l,\pm 2c_l)$	4	0.0010119375926735754754	
		$(3c_l,0)_{FS}$	4	0.00024530102775771734547	
		$(c_l,2c_l)_{FS}$	8	0.00535304900051377523273	
		$(c_l,3c_l)_{FS}$	8	0.00028341425299419821740	
		$c_l = 1.19697977039307435897239$			

3D 5th-order Quadrature						
Quadrature	**Lattice**	$\boldsymbol{\xi}_i$	p	w_i	**lattice constant**	**Remark**
$E_{3,5}^{13}$		$(0,0,0)$	1	$\frac{2}{5}$	$r^2 = \frac{(5+\sqrt{5})}{2}$	**Off-lattice**
		$(\pm r, \pm s, 0)$	4	$\frac{1}{20}$		
		$(0, \pm r, \pm s)$	4	$\frac{1}{20}$	$s^2 = \frac{(5-\sqrt{5})}{2}$	
		$(\pm s, 0, \pm r)$	4	$\frac{1}{20}$		
$E_{3,5}^{15}$	$D3Q15$	$(0,0,0)$	1	$\frac{2}{9}$	$c_l = \sqrt{3}$	**On-lattice**
		$(c_l, 0, 0)_{FS}$	6	$\frac{1}{9}$		
		$(\pm c_l, \pm c_l, \pm c_l)$	8	$\frac{1}{72}$		
$E_{3,5}^{19}$	$D3Q19$	$(0,0,0)$	1	$\frac{1}{3}$	$c_l = \sqrt{3}$	**On-lattice**
		$(c_l, 0, 0)_{FS}$	6	$\frac{1}{18}$		
		$(c_l, c_l, 0)$	12	$\frac{1}{36}$		
$E_{3,5}^{27}$	$D3Q27$	$(0,0,0)$	1	$\frac{8}{27}$	$c_l = \sqrt{3}$	**On-lattice**
		$(c_l, 0, 0)_{FS}$	6	$\frac{2}{27}$		
		$(c_l, c_l, 0)$	12	$\frac{1}{54}$		
		$(\pm c_l, \pm c_l, \pm c_l)$	8	$\frac{1}{216}$		

3D 7th-order Quadrature						
Quadrature	**Lattice**	$\boldsymbol{\xi}_i$	p	w_i	**lattice constant**	**Remark**
$E_{3,7}^{27}$		$(0,0,0)$	1	$\frac{721\pm 8\sqrt{15}}{2205}$		**Off-lattice**
		$(r, 0, 0)_{FS}$	6	$\frac{270\mp 46\sqrt{15}}{15435}$	$r^2 = \frac{15\pm\sqrt{15}}{2}$	
		$(s, s, 0)$	12	$\frac{162\pm 41\sqrt{15}}{6174}$	$s^2 = 6 \mp \sqrt{15}$	
		$(\pm t \pm t, \pm t)$	8	$\frac{783\mp 202\sqrt{15}}{24696}$	$t^2 = 9 \pm \sqrt{15}$	
$E_{3,7}^{39}$	$D3Q39$	$(0,0,0)$	1	$\frac{1}{12}$	$c_l^2 = \frac{3}{2}$	**On-lattice**
		$(c_l, 0, 0)_{FS}$	6	$\frac{1}{12}$		
		$(\pm c_l, \pm c_l, \pm c_l)$	8	$\frac{1}{27}$		
		$(2c_l, 0, 0)_{FS}$	6	$\frac{2}{125}$		
		$(2c_l, 2c_l, 0)_{FS}$	12	$\frac{1}{432}$		
		$(3c_l, 0, 0)_{FS}$	6	$\frac{1}{1620}$		

3D 9th-order Quadrature

Quadrature	**Lattice**	$\boldsymbol{\xi}_i$	p	w_i	**Remark**
$E_{3,9}^{121}$	$D3Q121$	$(0,0,0)$	1	0.030591622029486006442469	**On-lattice**
		$(c_l,0,0)_{FS}$	6	0.09851595103726339186467	
		$(\pm c_l,\pm c_l,\pm c_l)$	8	0.02752500532563812386479	
		$(c_l,2c_l,0)_{FS}$	24	0.00611102336683342432241	
		$(2c_l,2c_l,0)_{FS}$	12	0.00042818359368108406618	
		$(3c_l,0,0)_{FS}$	6	0.00032474752708807381296	
		$(2c_l,3c_l,0)_{FS}$	24	0.00001431862411548029405	
		$(\pm 2c_l,\pm 2c_l,\pm 2c_l)$	8	0.00018102175157637423759	
		$(c_l,3c_l,0)_{FS}$	24	0.00010683400245939109491	
		$(\pm 3c_l,\pm 3c_l,\pm 3c_l)$	8	0.00000069287508963860285	
lattice constant: c_l =1.19697977039307435897239					

APPENDIX C

LIST OF ACRONYMS AND SYMBOLS

C.1 ACRONYMS

- BC: Boundary conditions
- BGK: Bhatnagar-Gross-Krook
- BBGK: Boltzmann-BGK
- BBGKY: Bogoliubov–Born–Green–Kirkwood–Yvon
- CFD: Computational fluid dynamics
- CE: Chapman-Enskog expansion procedure
- DVBE: Discrete velocity Boltzmann equation
- EOS: Equation of state
- EDF: Equilibrium distribution function
- IC: Initial conditions
- FDM:Finite difference method
- FEM: Finite element method
- LBGK: Lattice Boltzmann-BGK
- LGCA: Lattice gas cellular automata
- Ma: Mach number
- MFLOPS: Million floating point operations
- MRT: Multiple relaxation time
- OLB: Off-lattice Boltzmann

C.2 SYMBOLS

C.2.1 Greek

- α: spatial-coordinate index
- ρ : Density
- ϵ : Internal energy of the particle per unit mass
- $\boldsymbol{\xi}$: velocity in continuum (Dimensional and non-dimensional)
- $\boldsymbol{\xi}_i$: Discretized, non-dimensional velocity
- τ: (single) relaxation time (Dimensional and non-dimensional)
- λ : Modified relaxation time
- λ : Bulk viscosity or second viscosity
- $\boldsymbol{\delta}$: Kronecker delta function
- σ : Stress tensor
- Π : Total stress tensor
- ν : Kinematic viscosity of the fluid
- $\boldsymbol{\Omega}$: Collision kernel
- ω : Weight function
- Δt : time-step
- $\Delta\, \boldsymbol{x}$ **:** spatial step

C.2.2 Roman

- $\boldsymbol{a}^{(n)}$: $n^{th}-$order expansion coefficient tensor of the *full* distribution, f, $n = 0, 1, \ldots$
- $\boldsymbol{a}_0^{(n)}$: $n^{th}-$order expansion coefficient tensor of $f^{(0)}$, $n = 0, 1, \ldots$
- $\boldsymbol{a}_k^{(n)}$: $n^{th}-$order expansion coefficient tensor of $f^{(k)}$, $n = 0, 1, \ldots$, $k = 0, 1$.
- c_s : Non-dimensional speed of sound
- D: Number of spatial dimensions
- $\boldsymbol{e}_i$: integer valued discrete velocities, $i = \{\cdots - 2, -1, 0, 1, 2 \cdots\}$
- $\boldsymbol{g}$: acceleration

- $f^{(k)}$: kth level distribution function on according to the Chapman-Enskog expansion procedure; $k = 0, 1, ...$
- f_i : discrete distribution corresponding to the ith discrete velocity
- f_i^c :collided (post-collision) distribution
- $f^{(0)}$: Maxwell-Boltzmann distribution
- $f_i^{(eq)}$: (discretized) equilibrium distribution
- $\boldsymbol{F}$: external force
- $\boldsymbol{j} \equiv \rho \boldsymbol{u}$: Fluid momentum
- Kn : Knudsen number
- k_B : Boltzmann constant
- $\mathscr{H}$: Hamiltonian of $N-$body system
- $\mathcal{O}$: Of the order of
- $\boldsymbol{Q}$: Third-order moment
- $\mathcal{Q}$: Order of precision of Gauss-Hermite quadrature
- $\boldsymbol{q}$: heat or internal energy flux
- $\boldsymbol{P}$: Second-order moment
- $E_{D,m}^{d}$: quadrature with d points in D dimension with m degree of accuracy of the quadrature
- $\boldsymbol{x}$: particle position vector
- w_i : Discrete weights

BIBLIOGRAPHY

[1] *Hermite Polynomials on Wikipedia.*

[2] T. Abe, *Derivation of the Lattice Boltzmann Method by Means of the Discrete Ordinate Method for the Boltzmann Equation*, J. Comput. Phys., 131 (1997), pp. 241–246.

[3] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover Publications, 1965.

[4] C. K. Aidun and J. R. Clausen, *Lattice-Boltzmann Method for Complex Flows*, Annu. Rev. Fluid Mech., 42 (2010), pp. 439–472.

[5] F. Alexander, S. Chen, and J. Sterling, *Lattice Boltzmann Thermohydrodynamics*, Phys. Rev. E, 47 (1993).

[6] AMD, *AMD Opteron 6300 Series Processors*, 2014.

[7] S. Ansumali and I. V. Karlin, *Single relaxation time model for entropic lattice Boltzmann methods*, Phys. Rev. E, 65 (2002), p. 56312.

[8] S. Ansumali, I. V. Karlin, and H. C. Öttinger, *Minimal entropic kinetic models for hydrodynamics*, Europhys. Lett., 63 (2003), pp. 798–804.

[9] P. Bailey, J. Myre, S. Walsh, D. Lilja, and M. Saar, *Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors*, in 2009 Int. Conf. Parallel Process., IEEE, Sept. 2009, pp. 550–557.

[10] A. Bardow and A. Gusev, *Multispeed models in off-lattice Boltzmann simulations*, Phys. Rev. E, 77 (2008), p. 025701.

[11] A. Bardow, I. V. Karlin, and A. A. Gusev, *General characteristic-based algorithm for off-lattice Boltzmann simulations*, Europhys. Lett., 75 (2006), pp. 434–440.

[12] B. Barney, *Introduction to Parallel Computing.*

[13] G. K. Batchelor, *An Introduction to Fluid Dynamics*, Cambridge University Press, 2000.

[14] R. Benzi, S. Succi, and M. Vergassola, *The lattice Boltzmann equation: theory and applications*, Phys. Rep., 222 (1992), pp. 145–197.

[15] M. J. Bernsdorf, *Simulation of Complex Flows and Multi-Physics with the Lattice-Boltzmann Method*, PhD thesis, University of Amsterdam, 2008.

[16] 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*, Phys. Rev., 94 (1954), pp. 511–525.

[17] L. Biferale, F. Mantovani, M. Pivanti, F. Pozzati, M. Sbragaglia, A. Scagliarini, S. F. Schifano, F. Toschi, and R. Tripiccione, *An optimized D2Q37 Lattice Boltzmann code on GP-GPUs*, Comput. Fluids, 80 (2013), pp. 55–62.

[18] G. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, Oxford University Press, 1994.

[19] N. Cao, S. Chen, S. Jin, and D. Martínez, *Physical symmetry and lattice symmetry in the lattice Boltzmann method*, Phys. Rev. E, 55 (1997), pp. R21–R24.

[20] Center of Simulation and Modeling, *Frank — CORE.SAM.*

[21] C. Cercignani, *The Boltzmann Equation and Its Applications*, Springer, New York, NY, USA, 1987.

[22] C. Cercignani and R. Penrose(Foreword), *Ludwig Boltzmann: The Man Who Trusted Atoms*, Oxford University Press, 2006.

[23] B. Chapman, G. Jost, and R. van van van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press; Scientific and Engineering Computation edition, 2007.

[24] S. Chapman and T. Cowling, *The Mathematical Theory of Non-uniform Gases*, Cambridge University Press, 3rd ed., 1991.

[25] H. Chen, *Volumetric formulation of the lattice Boltzmann method for fluid dynamics: Basic concept*, Phys. Rev. E, 58 (1998), pp. 3955–3963.

[26] H. Chen and X. Shan, *Fundamental conditions for N-th-order accurate lattice Boltzmann models*, Phys. D Nonlinear Phenom., 237 (2008), pp. 2003–2008.

[27] S. Chen and G. D. Doolen, *Lattice Boltzmann Method*, Anuual Rev. Fluid Mech., 30 (1998), pp. 329–364.

[28] S. Chen, D. Martínez, and R. Mei, *On boundary conditions in lattice Boltzmann methods*, Phys. Fluids, 8 (1996), pp. 2527–2536.

[29] S. Chen, Z. Wang, X. Shan, and G. D. Doolen, *Lattice Boltzmann computational fluid dynamics in three dimensions*, J. Stat. Phys., 68 (1992), pp. 379–400.

[30] Y. Chen, H. Ohashi, and M. Akiyama, *Thermal lattice Bhatnagar-Gross-Krook model without nonlinear deviations in macrodynamic equations*, Phys. Rev. E, 50 (1994), pp. 2776–2783.

[31] S. Chikatamarla and I. Karlin, *Lattices for the lattice Boltzmann method*, Phys. Rev. E, 79 (2009), p. 046701.

[32] S. S. Chikatamarla and I. V. Karlin, *Entropy and Galilean invariance of lattice Boltzmann theories*, Phys. Rev. Lett., 97 (2006), p. 190601.

[33] R. M. Clever and F. H. Busse, *Transition to time-dependent convection*, J. Fluid Mech., 65 (1974), p. 625.

[34] E. Davidson, *Message-passing for Latiice Boltzmann*, PhD thesis, The University of Edinburgh, 2008.

[35] M. O. Deville and T. B. Gatski, *Mathematical Modeling for Complex Fluids and Flows*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] D. D'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.-S. Luo, *Multiple-Relaxation-Time Lattice {B}oltzmann Models in Three Dimensions*, Phil. Trans. R. Soc. A, 360 (2002), pp. 437–451.

[37] I. D'Humieres, Dominique Ginzburg, M. Krafczyk, P. Lallemand, and L.-S. Luo, *Multiple-relaxation-time lattice Boltzmann models in three dimensions*, Philos. Trans. A. Math. Phys. Eng. Sci., 360 (2011), pp. 437–451.

[38] V. Eijkhout, *Introduction to High Performance Scientific Computing:*, 2013.

[39] C. L. Fefferman, *Existence and Smoothness of the Navier-Stokes Equation.* \url{http://www.claymath.org/millennium/Navier-Stokes_Equations/navierstokes.pdf}, 2000.

[40] U. Frisch, B. Hasslacher, and Y. Pomeau, *Lattice-Gas Automata for the Navier-Stokes Equation*, Phys. Rev. Lett., 56 (1986), pp. 1505–1508.

[41] U. Ghia, K. N. Ghia, and C. T. Shin, *High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method*, J. Comput. Phys., 48 (1982), pp. 387–411.

[42] S. Goedecker and A. Hoisie, *Performance Optimization of Numerically Intensive Codes (Software, Environments and Tools)*, Society for Industrial and Applied Mathematics, 2001.

[43] F. Golse, *Boltzmann-Grad limit*, Scholarpedia, 8 (2013), p. 9141.

[44] H. Grad, *Note on N-dimensional hermite polynomials*, Commun. Pure Appl. Math., 2 (1949), pp. 325–330.

[45] ——, *On the kinetic theory of rarefied gases*, Commun. Pure Appl. Math., 2 (1949), pp. 331–407.

[46] ——, *Thermodynamik der Gase / Thermodynamics of Gases*, vol. 3 / 12 of Handbuch der Physik / Encyclopedia of Physics, Springer Berlin Heidelberg, Berlin, Heidelberg, 1958.

[47] Z. Guo and C. Shu, *Lattice Boltzmann Method and Its Applications in Engineering*, World Scientific Publishing Company, 2013.

[48] Z. Guo, T. Zhao, and Y. Shi, *Preconditioned lattice-Boltzmann method for steady flows*, Phys. Rev. E, 70 (2004), p. 066706.

[49] Z. Guo and T. S. Zhao, *Explicit finite-difference lattice Boltzmann method for curvilinear coordinates*, Phys. Rev. E. Stat. Nonlin. Soft Matter Phys., 67 (2003), p. 066709.

[50] Z. Guo, T. S. Zhao, and Y. Shi, *Preconditioned lattice-Boltzmann method for steady flows*, Phys. Rev. E, 70 (2004), p. 66706.

[51] Z. Guo, C. Zheng, and B. Shi, *An extrapolation method for boundary conditions in lattice Boltzmann method*, Phys. Fluids, 14 (2002), p. 2007.

[52] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, Chapman & Hall/CRC, first ed., 2011.

[53] S. Harris, *An Introduction to the Theory of the Boltzmann Equation*, Dover Publications, 2011.

[54] X. He, S. Chen, and G. D. Doolen, *A Novel Thermal Model for the Lattice Boltzmann Method in Incompressible Limit*, J. Comput. Phys., 300 (1998), pp. 282–300.

[55] X. He and L.-S. Luo, *Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation*, Phys. Rev. E, 56 (1997), pp. 6811–6817.

[56] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Elsevier, 2012.

[57] V. Heuveline, M. J. Krause, and J. Latt, *Towards a hybrid parallelization of lattice Boltzmann methods*, Comput. Math. with Appl., 58 (2009), pp. 1071–1080.

[58] F. J. Higuera and J. Jiménez, *Boltzmann Approach to Lattice Gas Simulations*, EPL (Europhysics Lett., 9 (1989), p. 663.

[59] F. J. Higuera, S. Succi, and R. Benzi, *Lattice Gas Dynamics with Enhanced Collisions*, Europhys. Lett., 9 (1989), pp. 345–349.

[60] C. Hirsch, *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*, Butterworth-Heinemann, second ed., 2007.

[61] K. Huang, *Statistical Mechanics*, Wiley, 2nd ed., 1987.

[62] D. Kandhai, W. Soll, S. Chen, A. Hoekstra, and P. Sloot, *Finite-Difference Lattice-BGK methods on nested grids*, Comput. Phys. Commun., 129 (2000), pp. 100–109.

[63] J. M. V. A. Koelman, *A Simple Lattice Boltzmann Scheme for Navier-Stokes Fluid Flow*, Europhys. Lett., 15 (1991), pp. 603–607.

[64] M. Kowarschik and C. Weiss, *An Overview of Cache Optimization Techniques and Cache-aware Numerical Algorithms*, tech. report.

[65] G. M. Kremer, *An Introduction to the Boltzmann Equation and Transport Processes in Gases*, Springer Science & Business Media, 2010.

[66] P. K. Kundu, I. M. Cohen, and D. R. Dowling, *Fluid Mechanics*, Academic Press, 5th ed., 2011.

[67] P. Lallemand and L.-S. Luo, *Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability*, Phys. Rev. E, 61 (2000), pp. 6546–6562.

[68] J. Latt and B. Chopard, *Lattice Boltzmann method with regularized pre-collision distribution functions*, Math. Comput. Simul., 72 (2006), pp. 165–168.

[69] ——, *Straight velocity boundaries in the lattice Boltzmann method*, Phys. Rev. E, 77 (2008), p. 056703.

[70] T. Lee and C.-L. Lin, *A Characteristic Galerkin Method for Discrete Boltzmann Equation*, J. Comput. Phys., 171 (2001), pp. 336–356.

[71] T. Lee and C. L. Lin, *An Eulerian description of the streaming process in the lattice Boltzmann equation*, J. Comput. Phys., 185 (2003), pp. 445–471.

[72] Y. Li and X. Shan, *Lattice Boltzmann method for adiabatic acoustics.*, Philos. Trans. A. Math. Phys. Eng. Sci., 369 (2011), pp. 2371–80.

[73] F. Mantovani, M. Pivanti, S. Schifano, and R. Tripiccione, *Performance issues on many-core processors: A D2Q37 Lattice Boltzmann scheme as a test-case*, Comput. Fluids, 88 (2013), pp. 743–752.

[74] K. Mattila, J. Hyväluoma, T. Rossi, M. Aspnäs, and J. Westerholm, *An efficient swap algorithm for the lattice Boltzmann method*, Comput. Phys. Commun., 176 (2007), pp. 200–210.

[75] G. McNamara and B. Alder, *Analysis of the lattice Boltzmann treatment of hydrodynamics*, Phys. A Stat. Mech. its Appl., 194 (1993), pp. 218–228.

[76] G. McNamara, A. Garcia, and B. Alder, *Stabilization of thermal lattice Boltzmann models*, J. Stat. Phys., 81 (1995), pp. 395–408.

[77] G. McNamara and G. Zanetti, *Use of the Boltzmann Equation to Simulate Lattice-Gas Automata*, Phys. Rev. Lett., 61 (1988), pp. 2332–2335.

[78] G. Moore, *Cramming More Components Onto Integrated Circuits*, Proc. IEEE, 86 (1998), pp. 82–85.

[79] F. Nannelli and S. Succi, *The Lattice Boltzmann Equation on Irregular Lattices*, J. Stat. Phys., 68 (1992), pp. 401–407.

[80] X. B. Nie, X. Shan, and H. Chen, *Galilean invariance of lattice Boltzmann models*, EPL (Europhysics Lett., 81 (2008), p. 34005.

[81] L. Oliker, A. Canning, J. Carter, J. Shalf, and S. Ethier, *Scientific Computations on Modern Parallel Vector Systems*, in Proc. ACM/IEEE SC2004 Conf., IEEE, 2004, pp. 10–10.

[82] OpenACC, *OpenACC Application Programming Interface*, tech. report, 2013.

[83] OpenMP Architecture Review Board, *OpenMP Application Program Interface, Version 3.0*, tech. report.

[84] C. Peng, *The Lattice Boltzmann Method for Fluid Dynamics: Theory and Applications*, masters thesis, Ecole Polytechnique Federale De Laussane (EPFL), 2013.

[85] L. Peng, K.-i. Nomura, T. Oyakawa, R. Kalia, A. Nakano, and P. Vashishta, *Parallel Lattice Boltzmann Flow Simulation on Emerging Multi-core Platforms*, in Euro-Par 2008 Parallel Process., E. Luque, T. Margalef, and D. Benitez, eds., Springer Berlin Heidelberg, 2008, pp. 763–777.

[86] P. C. Philippi, L. a. Hegele, L. O. E. Dos Santos, and R. Surmas, *From the continuous to the lattice Boltzmann equation: The discretization problem and thermal models*, Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys., 73 (2006), p. 056702.

[87] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, and T. Zeiser, *Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures*, in Proc. ACM/IEEE SC2004 Conf., IEEE, Nov. 2004, pp. 21–21.

[88] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rüde, *Optimization and profiling of the cache performance of parallel lattice Boltzmann codes*, Parallel Process. Lett., 13 (2003), pp. 549–560.

[89] N. I. Prasianakis and I. V. Karlin, *Lattice Boltzmann method for thermal flow simulation on standard lattices*, Phys. Rev. E, 76 (2007), p. 16702.

[90] Y. H. Qian, D'Humières, and P. Lallemand, *Lattice BGK Models for Navier-Stokes Equation*, EPL (Europhysics Lett., 17 (1992), p. 479.

[91] A. P. Randles, V. Kale, J. Hammond, W. Gropp, and E. Kaxiras, *Performance Analysis of the Lattice Boltzmann Model Beyond Navier-Stokes*, 2013 IEEE 27th Int. Symp. Parallel Distrib. Process., (2013), pp. 1063–1074.

[92] P. R. Rao and L. A. Schaefer, *Numerical stability of explicit off-lattice Boltzmann schemes: A comparative study*, J. Comput. Phys., 285 (2015), pp. 251–264.

[93] D. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press, 2nd ed., 2004.

[94] M. B. Reider and J. D. Sterling, *Accuracy of discrete-velocity BGK models for the simulation of the incompressible Navier-Stokes equations*, Comput. Fluids, 24 (1995), pp. 459–467.

[95] C. Sanderson, *Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments*, tech. report, NICTA, Australia, Oct. 2010.

[96] M. Sbragaglia, R. Benzi, L. Biferale, H. Chen, X. Shan, and S. Succi, *Lattice Boltzmann method with self-consistent thermo-hydrodynamic equilibria*, J. Fluid Mech., 628 (2009), p. 299.

[97] A. Scagliarini, L. Biferale, M. Sbragaglia, K. Sugiyama, and F. Toschi, *Lattice Boltzmann methods for thermal flows: Continuum limit and applications to compressible Rayleigh Taylor systems*, Phys. Fluids, 22 (2010), p. 055101.

[98] M. Schultz, M. Krafczyk, J. Tölke, and E. Rank, *Parallelization Strategies and Efficiency of CFD Computations in Complex Geometries Using Lattice Boltzmann Methods on High-Performance Computers*, in High Perform. Sci. Eng. Comput., vol. 21 of Lecture Notes in Computational Science and Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 21 ed., 2002, pp. 115–122.

[99] SGI, *Optimizing Cache Utilization*.

[100] X. Shan, *Simulation of Rayleigh-Benard convection using a lattice Boltzmann method*, Phys. Rev. E, 55 (1997), pp. 2780–2788.

[101] ——, *General solution of lattices for Cartesian lattice Bhatanagar-Gross-Krook models*, Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys., 81 (2010), p. 036702.

[102] X. Shan and H. Chen, *Lattice Boltmann model for simulating flows with multiple phases and components*, Phys. Rev. E, 47 (1993), pp. 1815–1820.

[103] X. Shan and H. Chen, *A general multiple-relaxation-time Boltzmann collision model*, Int. J. Mod. Phys. C Comput. Phys. I& Phys. Comput., 18 (2007), pp. 635–643.

[104] X. Shan and X. He, *Discretization of the Velocity Space in the Solution of the Boltzmann Equation*, Phys. Rev. Lett., 80 (1998), pp. 65–68.

[105] X. Shan, X.-F. Yuan, and H. Chen, *Kinetic theory representation of hydrodynamics: a way beyond the NavierStokes equation*, J. Fluid Mech., 550 (2006), p. 413.

[106] A. G. SHET, K. SIDDHARTH, S. H. SORATHIYA, A. M. DESHPANDE, S. D. SHERLEKAR, B. KAUL, and S. ANSUMALI, *On vectorization for lattice based simulations*, Int. J. Mod. Phys. C, 24 (2013), p. 1340011.

[107] D. N. Siebert, L. A. Hegele, and P. C. Philippi, *Lattice Boltzmann equation linear stability analysis: Thermal and athermal models*, Phys. Rev. E, 77 (2008), p. 026707.

[108] D. N. Siebert, L. A. Hegele Jr., R. Surmas, L. O. E. Dos Santos, and P. C. Philippi, *Thermal Lattice Boltzmann in Two Dimensions*, Int. J. Mod. Phys. C Comput. Phys. I& Phys. Comput., 18 (2007), pp. 546–555.

[109] P. Skordos, *Initial and boundary conditions for the lattice Boltzmann method*, Phys. Rev. E, 48 (1993), pp. 4823–4842.

[110] V. Sofonea and R. F. Sekerka, *Viscosity of finite difference lattice Boltzmann models*, J. Comput. Phys., 184 (2003), pp. 422–434.

[111] E. A. Spiegel and G. Veronis, *On the Boussinesq Approximation for a Compressible Fluid.*, Astrophys. J., 131 (1960), p. 442.

[112] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Oxford University Press, 2001.

[113] S. Succi, I. Karlin, and H. Chen, *Colloquium: Role of the H theorem in lattice Boltzmann hydrodynamic simulations*, Rev. Mod. Phys., 74 (2002), pp. 1203–1220.

[114] S. Succi, M. Sbragaglia, and S. Ubertini, *Lattice Boltzmann Method*, Scholarpedia, 5 (2010), p. 9507.

[115] R. Surmas, C. E. Pico Ortiz, and P. C. Philippi, *Simulating thermohydrodynamics by finite difference solutions of the Boltzmann equation*, Eur. Phys. J. Spec. Top., 171 (2009), pp. 81–90.

[116] A. Tamura, K. Okuyama, S. Takahashi, and M. Ohtsuka, *Three-dimensional discrete-velocity BGK model for the incompressible Navier Stokes equations*, Comput. Fluids, 40 (2011), pp. 149–155.

[117] C. Teixeira, H. Chen, and D. M. Freed, *Multi-speed thermal lattice Boltzmann method stabilization via equilibrium under-relaxation*, Comput. Phys. Commun., 129 (2000), pp. 207–226.

[118] E. Turkel, *Preconditioned methods for solving the incompressible and low speed compressible equations*, J. Comput. Phys., 72 (1987), pp. 277–298.

[119] E. Turkel, *Preconditioning Techniques in Computational Fluid Dynamics*, Annu. Rev. Fluid Mech., 31 (1999), pp. 385–416.

[120] S. Ubertini and S. Succi, *A generalised lattice Boltzmann equation on unstructured grids*, Commun. Comput. Phys., 3 (2008), pp. 342–356.

[121] G. Vahala, P. Pavlo, L. Vahala, and N. S. Martys, *Thermal Lattice-Boltzmann Models (TLBM) for Compressible Flows*, Int. J. Mod. Phys. C, 09 (1998), pp. 1247–1261.

[122] M. Watari and M. Tsutahara, *Possibility of constructing a multispeed Bhatnagar-Gross-Krook thermal model of the lattice Boltzmann method*, Phys. Rev. E, 70 (2004), p. 16703.

[123] 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.

[124] F. White, *Viscous Fluid Mechanics*, McGraw-Hill Science/Engineering/Math, 3rd ed., 2005.

[125] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick, *Optimization of a lattice Boltzmann computation on state-of-the-art multicore platforms*, J. Parallel Distrib. Comput., 69 (2009), pp. 762–777.

[126] S. Williams, A. Waterman, and D. Patterson, *Roofline: an insightful visual performance model for multicore architectures*, Commun. ACM, 52 (2009), p. 65.

[127] M. Wittmann, T. Zeiser, G. Hager, and G. Wellein, *Comparison of different propagation steps for lattice Boltzmann methods*, Comput. Math. with Appl., 65 (2013), pp. 924–935.

[128] D. A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*, no. no. 1725 in Lattice-gas Cellular Automata and Lattice Boltzmann Models: An Introduction, Springer, 2000.

[129] R. Zhang, X. Shan, and H. Chen, *Efficient kinetic method for fluid simulation beyond the Navier-Stokes equation*, Phys. Rev. E, 74 (2006), p. 046703.

[130] O. C. Zienkiewicz and R. Codina, *A general algorithm for compressible and incompressible flow Part I the split, characteristic-based scheme*, Int. J. Numer. Methods Fluids, 20 (1995), pp. 869–885.