Supplementary Document

Yixin Chen, Wei Li, Rui Fan and Xiaopei Liu

I. Fundamentals of Fluid Dynamics

Fluid simulation can usually be described by the well-known Navier-Stokes equations. At constant temperature, the most general governing equations for fluid flows are the isothermal compressible Navier-Stokes equations (CNSE), which are written in the following form as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0 \ , \tag{1}$$

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} - \boldsymbol{\sigma}) = \rho \boldsymbol{g} \ , \tag{2}$$

where t is time; ρ and $\boldsymbol{u}$ are density and velocity fields to be solved for; p is the pressure field, which can usually be determined by the equation of state (EoS): $p = p(\rho, T)$, where T is the temperature; for ideal gas, $p = \rho RT$. $\boldsymbol{g}$ is the external force per unit volume (force density), such as gravity; $\boldsymbol{I}$ is the second-order identity tensor (identity matrix), and $\boldsymbol{\sigma}$ is the shear stress tensor defined as:

$$\boldsymbol{\sigma} = 2\rho\nu\boldsymbol{S} - \rho\nu'(\nabla \cdot \boldsymbol{u})\boldsymbol{I} \ , \tag{3}$$

where ν and ν' are the shear and bulk viscosities, respectively; for ideal gas model, $\nu' = 2\nu/3$; and $\boldsymbol{S}$ is the strain rate tensor defined as:

$$\boldsymbol{S} = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right) . \tag{4}$$

Due to inherent compression waves (e.g., sound waves), CNSE are more difficult to solve numerically. To prevent the influence by compression waves, incompressible Navier-Stokes equations (INSE) are proposed by setting ρ to be a constant:

$$\nabla \cdot \boldsymbol{u} = 0 \ , \tag{5}$$

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}\right) = -\nabla p + \rho\nu\nabla^2 \boldsymbol{u} + \rho \boldsymbol{g} \ , \tag{6}$$

which introduces an equation (a Poisson equation) that needs to be solved globally over the entire simulation domain. This is computationally very expensive. In addition, for both CNSE and INSE, in order to have stable numerical solution, implicit formulations are usually adopted, which results in more global equations to solve, further decreasing the computational performance. As argued in the main paper, solving these global equations are usually not easy to be well parallelized.

II. Boltzmann Equation

Although there are many ways to parallelize CNSE and INSE solvers, the mathematical and computational difficulty behind prevent us to have even more efficient solution methods. The efficiency can be much improved if the whole model equation can be changed such that its numerical form is purely local and easy to handle. Thus, in the domain of statistical mechanics, Boltzmann equation plays such a role [1], [2]. Boltzmann equation describes the evolution of a field of distribution $f = f(\boldsymbol{c}, \boldsymbol{x}, t)$, which specifies the probability of finding microscopic particles moving with velocity $\boldsymbol{c}$ at position $\boldsymbol{x}$ and time t (note that $\boldsymbol{c}$ is microscopic velocity):

$$\frac{\partial f}{\partial t} + \boldsymbol{c} \cdot \nabla f = \Omega(\rho, \boldsymbol{u}) + \boldsymbol{g} \cdot \nabla_{\boldsymbol{c}} f \ , \tag{7}$$

where $\boldsymbol{g}$ is the same external force per unit volume as in Eq.(2), and Ω is the inter-particle collision, which changes the distribution function f towards its local equilibrium distribution f^{eq}. Eq.(7) can be used to obtain macroscopic fields (density ρ, momentum $\rho\boldsymbol{u}$ and momentum flux tensor $\boldsymbol{\Pi}$) by taking the zero-th, first and second order moments of f w.r.t $\boldsymbol{c}$ as:

$$\rho = \int f d\boldsymbol{c}, \qquad \rho\boldsymbol{u} = \int \boldsymbol{v} f d\boldsymbol{c}, \qquad \boldsymbol{\Pi} = \int \boldsymbol{v}^2 f d\boldsymbol{c} \ , \tag{8}$$

where $\boldsymbol{\Pi}$ includes nonlinear momentum advection, pressure and shear stress tensor as: $\boldsymbol{\Pi} = \rho\boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} - \boldsymbol{\sigma}$, and pressure p can be computed as:

$$p = \frac{1}{3}\int \|\boldsymbol{c} - \boldsymbol{u}\|_2^2 f d\boldsymbol{v} \ . \tag{9}$$

Note that $\boldsymbol{c}$ is different from macroscopic velocity $\boldsymbol{u}$ – when $\boldsymbol{u}$ is zero, $\boldsymbol{c}$ is not necessarily zero. By applying the zero-th and first order moments (in Eq.(8)) to both sides of Eq.(7), we can obtain Eqs.(1)&(2) exactly, with the constraints that:

$$\int \Omega d\boldsymbol{c} = 0 \ , \qquad \int \boldsymbol{c}\Omega d\boldsymbol{c} = 0 \ , \tag{10}$$

Thus, a proper collision model for Ω should at least satisfy the above constraints. The most frequently used collision model is the iso-thermal single-relaxation-time Bhatnagar-Gross-Krook (BGK) model [3]:

$$\Omega(\rho, \boldsymbol{u}) = -\frac{f - f^{eq}(\rho, \boldsymbol{u})}{\tau} , \tag{11}$$

where τ is the relaxation time determined by kinematic viscosity ν, and $f^{eq}(\rho, \boldsymbol{u})$, which only depends on macroscopic quantities, is the local equilibrium distribution modeled by the Maxwell-Boltzmann distribution with constant temperature as:

$$f^{eq}(\rho, \boldsymbol{u}) = \frac{\rho}{(2\pi)^{d/2}} \exp\left(-\frac{\|\boldsymbol{c} - \boldsymbol{u}\|_2^2}{2}\right) , \tag{12}$$

where d is the number of dimension (e.g., in 3D case, $d = 3$). It can be verified mathematically that BGK model can satisfy the constraints in Eq.(10) and recover Eqs.(1) and (2) exactly. Note that for simulating incompressible fluid flows using kinetic method, we usually enforce low-speed constraint by limiting the value range of $\boldsymbol{u}$ not close to the speed of sound, which in this model is normalized to 1 (note that the speed of sound will later be rescaled after discretization).

III. Derivation of Lattice Boltzmann Equations

Eq.(7) seems numerically easier to solve since the differential operator is linear, which can result in a conservative numerical scheme with constant time step that is expected to be well suited for turbulent flow simulations. However, Eq.(7) is still difficult to be solved due to high dimension (there is an additional $\boldsymbol{c}$ as a dependent variable of f, which makes f a 7D function in 3D space, for example). Thus, some mathematical derivations should be performed in order to obtain lattice Boltzmann equations (LBE) as the numerical discretization of continuous Boltzmann equation. Among existing approaches, Hermite series expansion is a promising and mathematically more fundamental approach for derivation [4], where the continuous probability distribution function f can be represented as an expansion of Hermite series as:

$$f(\boldsymbol{c}, \boldsymbol{x}, t) = \omega(\boldsymbol{c}) \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{a}^{(n)}(\boldsymbol{x}, t) : \boldsymbol{H}^{(n)}(\boldsymbol{c}) , \tag{13}$$

where superscript '(n)' indicates rank-n tensor; the operator ":" denotes full tensor contraction, and the weighting function $\omega(\boldsymbol{c})$ is defined as:

$$\omega(\boldsymbol{c}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{\|\boldsymbol{c}\|_2^2}{2}\right) , \tag{14}$$

and the Hermite polynomial basis $\boldsymbol{H}^{(n)}(\boldsymbol{c})$ is computed based on the weighting function as:

$$\boldsymbol{H}^{(n)}(\boldsymbol{c}) = \frac{(-1)^n}{\omega(\boldsymbol{c})} \nabla^n \omega(\boldsymbol{c}) , \tag{15}$$

Due to ortho-normal property of the Hermite polynomial basis, the related coefficients can be calculated as:

$$\boldsymbol{a}^{(n)}(\boldsymbol{x}, t) = \int \frac{f(\boldsymbol{c}, \boldsymbol{x}, t)}{\omega(\boldsymbol{v})} \boldsymbol{H}^{(n)}(\boldsymbol{c}) d\boldsymbol{v} . \tag{16}$$

The reason we use Hermite series expansion is that its first a few orders (up to second order) correspond to macroscopic variables:

$$\boldsymbol{a}^{(0)} = \rho , \qquad \boldsymbol{a}^{(1)} = \rho \boldsymbol{u} , \qquad \boldsymbol{a}^{(2)} = \boldsymbol{\Pi} - \rho \boldsymbol{I} . \tag{17}$$

When discretizing the Boltzmann equation, we would like to only well approximate the macroscopic variables, e.g., ρ, $\rho\boldsymbol{u}$ and $\boldsymbol{\Pi}$ (see Eq.(8)). Thus, we need to approximate $\boldsymbol{a}^{(0)}$ to $\boldsymbol{a}^{(2)}$ with sufficient accuracy using discrete microscopic velocities $\boldsymbol{c}_i$. By employing Gauss-Hermite quadrature to approximate the integral in Eq.(16), we have:

$$\boldsymbol{a}^{(n)}(\boldsymbol{x}, t) = \sum_{i=0}^{q-1} \frac{w_i}{\omega(\boldsymbol{c}_i)} f(\boldsymbol{c}_i, \boldsymbol{x}, t) \boldsymbol{H}^{(n)}(\boldsymbol{c}_i) , \tag{18}$$

where q is the number of discrete microscopic velocities; and w_i are the quadrature weights (also called lattice weights). There can be different types of discretization for $\boldsymbol{c}_i$, and the commonly used structures in 2D and 3D are the D2Q9 and D3Q27 structures ($q = 3^d$) as illustrated in the main paper, which are written specifically in 2D as:

$$\boldsymbol{c}_i = \begin{cases} (0,0), & i = 0 , \\ (\pm 1, 0)c, (0, \pm 1)c, & i = 1, 2, 3, 4 , \\ (\pm 1, \pm 1)c, & i = 5, 6, 7, 8 , \end{cases} \tag{19}$$

and in 3D as:

$$\boldsymbol{c}_i = \begin{cases} (0,0,0), & i = 0\ , \\ (\pm 1,0,0)c, (0,\pm 1,0)c, (0,0,\pm 1)c, & i = 1,2,...,6\ , \\ (\pm 1,\pm 1,0)c, (\pm 1,0,\pm 1)c, (0,\pm 1,\pm 1)c, & i = 7,8,...,18\ , \\ (\pm 1,\pm 1,\pm 1)c, & i = 19,20,...,26\ , \end{cases} \tag{20}$$

where c is a microscopic velocity constant, and their corresponding lattice weights w_i in 2D are:

$$w_i = \begin{cases} 4/9, & i = 0\ , \\ 1/9, & i = 1,2,3,4\ , \\ 1/36, & i = 5,6,7,8\ , \end{cases} \tag{21}$$

and in 3D are:

$$w_i = \begin{cases} 8/27, & i = 0\ , \\ 2/27, & i = 1,2,...,6\ , \\ 1/54, & i = 7,8,...,18\ , \\ 1/216, & i = 19,20,...,26\ . \end{cases} \tag{22}$$

By defining discrete velocity distribution functions as:

$$f_i(\boldsymbol{x},t) = w_i f(\boldsymbol{c}_i,\boldsymbol{x},t)/\omega(\boldsymbol{c}_i)\ , \tag{23}$$

we can calculate the macroscopic fields as:

$$\rho = \sum_{i=0}^{q-1} f_i\ , \qquad \rho\boldsymbol{u} = \sum_{i=0}^{q-1} \boldsymbol{c}_i f_i\ , \qquad \boldsymbol{\Pi} = \sum_{i=0}^{q-1} \boldsymbol{c}_i^2 f_i\ , \qquad p = \frac{1}{3}\sum_{i=0}^{q-1} \|\boldsymbol{c}_i - \boldsymbol{u}\|_2^2 f_i\ . \tag{24}$$

Thus, we can write Eq.(7) in terms of discrete velocity distribution functions with BGK model as:

$$\frac{\partial f_i}{\partial t} + \boldsymbol{c}_i \cdot \nabla f_i = -\frac{f_i - f_i^{eq}}{\tau} + \boldsymbol{g} \cdot \nabla_{\boldsymbol{c}_i} f_i\ . \tag{25}$$

The key problem is how f_i^{eq} should be constructed. Since Eq.(12) is in a Gaussian form which is difficult to discretize w.r.t $\boldsymbol{c}$, we need convert it to a polynomial form instead. By Hermite polynomial expansion of Eq.(12) using Eq.(15) up to second order, we can obtain the discretized BGK equilibrium distributions as:

$$f_i^{eq}(\rho,\boldsymbol{u}) \approx w_i\rho\left(1 + \frac{\boldsymbol{c}_i\cdot\boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{c}_i\cdot\boldsymbol{u})^2}{2c_s^4} - \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2c_s^2}\right), \tag{26}$$

where c_s is the speed of sound. Numerically discretizing Eq.(25) further over space and time using standard Taylor expansion, we can obtain the LBE as:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i\delta t, t+\delta t) - f_i(\boldsymbol{x},t) = \Omega_i + G_i\ . \tag{27}$$

where δt is the time step, and G_i is the external force term by projecting $\boldsymbol{g}$ into distribution function space (The specific form of G_i will be specified later). In many LBE formulation, it is suggested to use normalized units for stability reason, where we usually enforce $c = 1$, $\delta t = 1$, and $c_s = 1/\sqrt{3}$, resulting in the following normalized LBE as:

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i, t+1) - f_i(\boldsymbol{x},t) = \Omega_i + G_i\ . \tag{28}$$

Eq.(28) can be solved by a standard (backward) streaming process which only involves one-ring neighbor:

$$f_i(\boldsymbol{x}, t+1) = f_i^*(\boldsymbol{x} - \boldsymbol{c}_i, t)\ . \tag{29}$$

and a per-grid-node collision process:

$$f_i^*(\boldsymbol{x},t) = f_i(\boldsymbol{x},t+1) + \Omega_i + G_i\ . \tag{30}$$

The above two processes are repeatedly applied to progress the evolution of Boltzmann equation over time in order to solve the fluid flows, which is very easy to be parallelized since all the above operations only involve one-ring neighbor access and copy plus an independent algebraic computation – no global equations need to be solved. Note that since the collision process conserves the mass and momentum, the macroscopic fields (ρ and $\boldsymbol{u}$) will only be changed after the streaming process.

IV. Relaxation Modeling

In Eq.(28), the collision term Ω_i is originally modeled by a single-relaxation-time BGK model:

$$\Omega_i = \frac{f_i - f_i^{eq}}{\tau} = r(f_i - f_i^{eq}) \ , \tag{31}$$

where $r = 1/\tau$ is the single-time relaxation rate, and τ is purely determined by kinematic viscosity as:

$$\tau = 3\nu + 1/2 \ . \tag{32}$$

It is well known that such a relaxation is numerically quite unstable for turbulent high Reynolds number flows, since different hydrodynamic modes are highly coupled together during the relaxation process, with the same rate towards their local equilibrium distributions, which may generate serious dispersion to break down the simulation for dynamics which is sensitive to numerical errors, such as turbulent flow simulations.

To overcome this problem (or at least largely reduce the dispersion problem), we can first write the relaxation in a more general form:

$$\boldsymbol{\Omega} = \boldsymbol{R}(\boldsymbol{f} - \boldsymbol{f}^{eq}) \ , \tag{33}$$

where $\boldsymbol{f}$, $\boldsymbol{f^{eq}}$ and $\boldsymbol{\Omega}$ collect all discretized distribution functions (of dimension 3^d) as well as their collision results at each grid node, and $\boldsymbol{R}$ ($3^d \times 3^d$) is a relaxation matrix, which is too general that exactly determining each value for $\boldsymbol{R}$ is very difficult. As a very special (extreme) case, for single-relaxation-time BGK model, $\boldsymbol{R}$ reduces to a diagonal matrix with constant diagonal elements: $\boldsymbol{R} = \boldsymbol{I}/\tau$, where $\boldsymbol{I}$ is the $3^d \times 3^d$ identity matrix.

A. Multiple-relaxation-time model

To establish a more efficient way to construct the relaxation matrix $\boldsymbol{R}$ with much reduced degrees of freedom, we assume $\boldsymbol{R}$ to be constructed by a matrix factorization process: $\boldsymbol{R} = -\boldsymbol{M}^{-1}\boldsymbol{D}\boldsymbol{M}$, where $\boldsymbol{M}$ is a pre-determined projection matrix (specified later), and $\boldsymbol{D}$ is a diagonal matrix whose non-zero components $\boldsymbol{D}_i$ ($i = 0, 1, ..., 3^d - 1$) contain multiple relaxation rates. This is the multiple-relaxation-time (MRT) model that has been advocated in the LBE literature [5], which tries to perform the relaxation in a more mode-independent way. Based on this formulation, MRT model is usually written as:

$$\boldsymbol{\Omega} = -\boldsymbol{M}^{-1}\boldsymbol{D}\boldsymbol{M}(\boldsymbol{f} - \boldsymbol{f}^{eq}) = -\boldsymbol{M}^{-1}\boldsymbol{D}(\boldsymbol{m} - \boldsymbol{m}^{eq}) \ , \tag{34}$$

and we usually construct $\boldsymbol{M}$ by taking different orders to moments of f_i so that some of the elements in $\boldsymbol{m}$ contain macroscopic variables. Thus, $\boldsymbol{M}$ is also called moment projection matrix in MRT model.

B. Construction of moment space projection matrix

There can be multiple ways to construct $\boldsymbol{M}$. The earliest method is to construct $\boldsymbol{M}$ by raw moments (RM-MRT)[6], whose elements are constructed by an ascending order of the monomials $\prod_{\alpha=0}^{d-1} \boldsymbol{c}_{\alpha,j}^{p_\alpha}$, where α indexes each component of a d-dimensional vector; $j = 0, 1, ..., 3^d - 1$; $p_\alpha \in \{0, 1, 2\}$ are the possible orders for each component, and $\boldsymbol{c}_{\alpha,j}$ indicates the α-th component of the j-th lattice velocity $\boldsymbol{c}_j$. Note that p_α always increases from 0 to 2 while α always starts from 0 to $d-1$. In particular, $\boldsymbol{M}$ can be constructed by a proper combination of the monomials. Assuming $\boldsymbol{M}$ can be written as a collection of its row vectors $\boldsymbol{m}_i$: $\boldsymbol{M} = [\boldsymbol{m}_0, \boldsymbol{m}_1, ..., \boldsymbol{m}_{q-1}]^T$, then each row vector can be constructed in 2D as:

$$\begin{aligned}
&\boldsymbol{m}_0 = \{\|\boldsymbol{c}_j\|^0\}, \\
&\boldsymbol{m}_1 = \{\boldsymbol{c}_{j,x}\}, \boldsymbol{m}_2 = \{\boldsymbol{c}_{j,y}\}, \\
&\boldsymbol{m}_3 = \{\boldsymbol{c}_{j,x}^2 + \boldsymbol{c}_{j,y}^2\}, \boldsymbol{m}_4 = \{\boldsymbol{c}_{j,x}^2 - \boldsymbol{c}_{j,y}^2\}, \boldsymbol{m}_5 = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}\}, \\
&\boldsymbol{m}_6 = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}\}, \boldsymbol{m}_7 = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}^2\}, \\
&\boldsymbol{m}_8 = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}^2\},
\end{aligned} \tag{35}$$

and in 3D as:

$$\begin{aligned}
&\boldsymbol{m}_0 = \{\|\boldsymbol{c}_j\|^0\}, \\
&\boldsymbol{m}_1 = \{\boldsymbol{c}_{j,x}\}, \boldsymbol{m}_2 = \{\boldsymbol{c}_{j,y}\}, \boldsymbol{m}_3 = \{\boldsymbol{c}_{j,z}\}, \\
&\boldsymbol{m}_4 = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}\}, \boldsymbol{m}_5 = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_6 = \{\boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_7 = \{\boldsymbol{c}_{j,x}^2 - \boldsymbol{c}_{j,y}^2\}, \boldsymbol{m}_8 = \{\boldsymbol{c}_{j,x}^2 - \boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_9 = \{\boldsymbol{c}_{j,x}^2 + \boldsymbol{c}_{j,y}^2 + \boldsymbol{c}_{j,z}^2\}, \\
&\boldsymbol{m}_{10} = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}^2 + \boldsymbol{c}_{j,x}\boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_{11} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y} + \boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_{12} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,z} + \boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_{13} = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}^2 - \boldsymbol{c}_{j,x}\boldsymbol{c}_{j,z}^2\}, \\
&\boldsymbol{m}_{14} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y} - \boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_{15} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,z} - \boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_{16} = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}\}, \\
&\boldsymbol{m}_{17} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}^2 + \boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,z}^2 + \boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_{18} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}^2 + \boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,z}^2 - \boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_{19} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}^2 - \boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,z}^2\}, \\
&\boldsymbol{m}_{20} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_{21} = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_{22} = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}^2\}, \boldsymbol{m}_{23} = \{\boldsymbol{c}_{j,x}\boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}^2\}, \\
&\boldsymbol{m}_{24} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}\boldsymbol{c}_{j,z}\}^2, \boldsymbol{m}_{25} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}\}, \boldsymbol{m}_{26} = \{\boldsymbol{c}_{j,x}^2\boldsymbol{c}_{j,y}^2\boldsymbol{c}_{j,z}^2\},
\end{aligned} \tag{36}$$

where $j = 0, 1, ..., q-1$ is the index of the row vector.

The above construction of $\boldsymbol{M}$ is called the non-orthogonal version of RM-MRT model, since the rows of $\boldsymbol{M}$ are not orthogonal. $\boldsymbol{M}$ can also be orthogonalized by Grad-Schmidt orthogonalization process. Due to better performance (in terms of accuracy and stability), RM-MRT model usually uses orthogonal version. However, it has been found that RM-MRT model violates Galilean invariance, which is a major source of error to introduce instability. To remedy this drawback, we can use central moments (CM-MRT) to construct $\boldsymbol{M}$ [7], as motivated from Eq.(12), where all $\boldsymbol{c}_j$ are replaced by $\bar{\boldsymbol{c}}_j = \boldsymbol{c}_j - \boldsymbol{u}$. By using central moments to construct $\boldsymbol{M}$, it is found mathematically that Galilean invariance is much better respected than using raw moments, thus significantly reducing the inherent error (mostly in a form of dispersion) and increase both accuracy and stability. Like in RM-MRT model, the construction of $\boldsymbol{M}$ is non-orthogonal, which can also be orthogonalized. However, unlike RM-MRT model, it has been found that the non-orthogonal version of CM-MRT model could perform better [8], which is the relaxation model we employ in our paper.

It deserves further discussion for the inverse of $\boldsymbol{M}$ ($\boldsymbol{M}^{-1}$) appearing in Eq.(34). For RM-MRT model, since all $\boldsymbol{c}_j$ are constant, $\boldsymbol{M}$ is thus a constant matrix, and $\boldsymbol{M}^{-1}$ is also a constant matrix, which can be pre-computed before simulation. However, for CM-MRT model, since $\boldsymbol{u}$ has been involved in $\boldsymbol{M}$: $\boldsymbol{M} = \boldsymbol{M}(\boldsymbol{u})$, it is no longer a constant and varies over space and time, which is the same for $\boldsymbol{M}^{-1}$. Thus, both $\boldsymbol{M}$ and $\boldsymbol{M}^{-1}$ should be computed on the fly. Initially, one may think of numerical inverse to compute $\boldsymbol{M}^{-1}$ at each grid node, but it costs too many local memory which is very difficult to be well parallelized on the GPU. In practice, $\boldsymbol{M}^{-1}$ can be analytically computed, and we directly use its analytical form in the CM-MRT collision model, which is not only more efficient, but also much more accurate. Note that $\boldsymbol{M}^{-1}$ is algebraically complex, and simplification should be done for higher computational performance, which we will discuss later.

C. Determining relaxation rates

The diagonal components $\boldsymbol{D}_i$ in the relaxation matrix $\boldsymbol{D}$ are different relaxation rates, which can be divided into low-order ($\boldsymbol{D}_{i\leq i^*}(i^* = (d^2+3d)/2-1)$) and high-order ($\boldsymbol{D}_{i>i^*}$) relaxation rates, where the low-order relaxation rates correspond to relaxations for physical quantities which can be fixed, while the high-order relaxation rates correspond to relaxations for non-physical quantities which should be determined in some other ways.

a) Determining low-order relaxation rates: Low-order relaxation rates are all fixed based on their physical correspondence. For $i = 0$, $\boldsymbol{D}_{i=0}$ is the zero-th order relaxation rate (corresponding to relaxation for density), which is always conserved, and we choose $\boldsymbol{D}_{i=0}=0$ (any other values are also acceptable). For $i = 1, 2, ..., d$, $\boldsymbol{D}_{i=1,2,...,d}$ are first-order relaxation rates, which correspond to momentum that could be changed by external forces; thus we always set $\boldsymbol{D}_{i=1,2,...,d} = 2$. For $i = d+1, ..., i^*$, $\boldsymbol{D}_{i=d+1,...i^*}$ contain the second-order relaxation rates, which are related to the kinematic viscosity ν by:

$$\boldsymbol{D}_i = (3\nu + 1/2)^{-1}, \qquad i = d+1, ..., i^* . \tag{37}$$

b) Determining high-order relaxation rates: High-order relaxation rates are more tricky to set, and many previous methods assume them to be 1 for CM-MRT models, resulting in too strong a filtering that shear instability could not be well simulated, especially for turbulent flows at high Reynolds numbers. It has been a long time in LBE literature that the role these high-order relaxation rates play in flow simulation is seldom discussed and analyzed. Recently, Li et al.[9] realized that appropriate high-order relaxation rates should be determined in order to capture reasonable shear instability in turbulence flows without obvious dispersion error. They formulated the high-order relaxation rates $\boldsymbol{D}_i$ ($i=i^*+1, ..., q-1$) in the same form as Eq.(37), but determined some other "viscosity" (replacing ν in Eq.(37) by $\hat{\nu}'_i$ as artificial viscosities for high-order relaxation rates) as:

$$\hat{\nu}'_i = \left(a\frac{|\nabla \boldsymbol{u}|}{g_{max}} + b \right) \nu'_i, \quad i > i^*, \tag{38}$$

where $a = -4$ and $b = 5$ are model parameters that they choose, but can be tuned; $g_{max} \in [0.1, 0.13]$ is the maximum gradient magnitude for normalization. ν'_i is the fixed high order artificial viscosity in 3D:

$$\begin{aligned} \nu'_i &= 0.005, \quad i = 9, 10, ..., 16, \\ \nu'_i &= 0.007, \quad i = 17, 18, ..., 22, \\ \nu'_i &= 0.009, \quad i = 23, 24, ..., 25, \\ \nu'_i &= 0.01, \quad\ \ i = 26. \end{aligned} \tag{39}$$

Note that the above model is empirical, but based on an important argument that dispersion errors can be suppressed if we enlarge "artificial viscosity" for high-order relaxation rates at smooth regions in order to prevent them from generating. We call the non-orthogonal central-moment relaxation model with the above adaptive setting of high-order relaxation rates the adaptive central-moment multiple-relaxation-time (ACM-MRT) model.

Very recently, a more systematic method to determine the high-order relaxation rates has been proposed in graphics [10]. They tried to find optimal rates locally at grid location $\mathbf{x}_k$ and time t by minimizing the following functional:

$$\epsilon(\mathbf{x}_k, t) = \frac{\|\delta^t(\rho)_k\|}{\overline{\rho}} + \frac{\|\delta^t(\rho\mathbf{u})_k\|}{\overline{\|\rho\mathbf{u}\|}} + \frac{\|\delta^t(\mathbf{\Pi})_k\|}{\overline{\|\mathbf{\Pi}\|}}, \tag{40}$$

where the temporal difference operator is: $\delta^t(\cdot)=(\cdot)^{t+\Delta t}-(\cdot)^t$, and the overline indicates averaging over the whole simulation process. However, directly minimizing this functional is computationally very expensive, and a linear regression model was developed with optimal model parameters. For more details of this approach, please refer to [10]. Note that from the computational perspective, this approach is as cheap as [9], but has higher accuracy. Both the approaches can be adopted in our implementation, but the in the main paper, all data are reported based on the method of [9].

V. External Force Approximation

When external force (e.g., gravity force) is present, we should consider how G_i in Eq.(26) are calculated. There have been many works investigating external force computation in lattice Boltzmann method. One of the force models that can be used is by Guo et al. [11], who proposed the following force approximation:

$$G_i = \boldsymbol{g}\cdot\nabla_{\boldsymbol{c}_i} f_i \approx \boldsymbol{g}\cdot\nabla_{\boldsymbol{c}_i} f_i^{eq} = (1-\frac{1}{2\tau})w_i\left(\frac{\boldsymbol{c}_i-\boldsymbol{u}}{c_s^2}+\frac{\boldsymbol{c}_i\cdot\boldsymbol{u}}{c_s^4}\cdot\boldsymbol{g}\right)\ . \tag{41}$$

Such an approximation can be projected into the (central) moment space, similarly as that for distribution functions:

$$\boldsymbol{G}^* = \boldsymbol{M}^{-1}\left(\boldsymbol{I}-\frac{\boldsymbol{R}}{2}\right)\boldsymbol{M}\boldsymbol{G} = \boldsymbol{M}^{-1}\left(\boldsymbol{I}-\frac{\boldsymbol{R}}{2}\right)\tilde{\boldsymbol{G}}\ , \tag{42}$$

where $\boldsymbol{G}$ collects all G_i at a grid node before performing force computation (which are equal to Eq.(41)); $\tilde{\boldsymbol{G}}$ is the force vector in moment space; and $\boldsymbol{G}^*$ contains all force terms after moment-space relaxation, which are finally used for the force term in Eq.(30).

VI. Immersed Boundary Kinetic Method

Usually, boundary conditions in lattice Boltzmann method are satisfied where the distribution functions f_i are locally manipulated in such a way that the low-order moments (e.g., momentum) match the desired conditions (e.g., no-slip at the boundary). Typical boundary conditions of this kind are bounce-back or regularized boundary conditions [12]. Immersed boundary (IB) kinetic method is an alternative, where forcing term can be employed around the solid boundaries to enforce (velocity) boundary conditions. To achieve this goal, the solid boundary surface should first be uniformly sampled, with some sampling techniques such as Poisson-disk surface sampling [13]. Then a force is computed at each solid sample to ensure boundary condition, which is then used to modify the velocity of the surrounding fluid nodes inside the simulation domain. The local property of IB method, especially in the setting of dynamic solids with complex geometries, makes it much more efficient and flexible than other boundary treatment method, which is the reason we employ in this paper.

The idea behind IB-based kinetic method is simple. We first assume that the fluid region has no solid objects inside (only with domain boundary); then when solids are present, we can interpolate the fluid momentum ($\rho\boldsymbol{u}$) to the solid boundaries, which may deviate from the desired momentum that the boundary conditions require. To remedy this difference, penalty forces (impulses) are applied at the solid boundary, which are spread by the same interpolation kernel to the nearby fluid nodes to update the fluid momentum. In the following, we specify in detail this idea:

- *Interpolation*: We first interpolate the fluid momentum from the grid nodes to the solid sample points (over the solid boundary surfaces) by an interpolation kernel $K(r)$ which is conservative:

$$\begin{aligned}(\rho\boldsymbol{u})(\boldsymbol{x}_s) = \mathcal{I}[(\rho\boldsymbol{u})(\boldsymbol{x})]_f &= \int(\rho\boldsymbol{u})(\boldsymbol{x})\delta(\boldsymbol{x}-\boldsymbol{x}_s)d\boldsymbol{x}\\ &\approx \sum_{j\in D_s}(\rho\boldsymbol{u})(\boldsymbol{x}_{f,j})K(\boldsymbol{x}_{f,j}-\boldsymbol{x}_s)\Delta v,\end{aligned} \tag{43}$$

  where D_s is a set of indices containing the fluid nodes around the solid sample at $\boldsymbol{x}_s$, and $\boldsymbol{x}_{f,j}$ are fluid node locations within the set; $\Delta v=\Delta x\Delta y$ in 2D and $\Delta v=\Delta x\Delta y\Delta z$ in 3D, where Δx, Δy and Δz are grid spacings along x-, y- and z-dimensions; $K(\boldsymbol{x}_{f,j}-\boldsymbol{x}_s)$ is a smoothed approximation to the Dirac delta function (which will be specified later). After interpolation, the interpolated momentum may deviate from the desired momentum at the solid boundary, which results in an impulse (force) to be added to the solid sample:

$$\boldsymbol{g}^{s\to f}(\boldsymbol{x}_s) = \frac{1}{\Delta t}\left((\rho\boldsymbol{u})(\boldsymbol{x}_s)-(\rho\boldsymbol{u})_b(\boldsymbol{x}_s)\right)\ , \tag{44}$$

  where $(\rho\boldsymbol{u})_b(\boldsymbol{x}_s)$ is the desired boundary condition at the solid sample $\boldsymbol{x}_s$.
- *Spreading*: Once we computed all the forces at solid boundary samples, we can spread them to the nearby fluid nodes within a neighborhood region (e.g., $2\times2\times2$ or $3\times3\times3$ in 3D), using the same kernel:

$$\begin{aligned}\boldsymbol{g}^{s\to f}(\boldsymbol{x}_f) = \mathcal{S}[\boldsymbol{g}_{s\to f}(\boldsymbol{x})]_s &= \int\boldsymbol{g}^{s\to f}(\boldsymbol{x})\delta(\boldsymbol{x}-\boldsymbol{x}_f)d\boldsymbol{x}\\ &\approx \sum_{j\in D_f}\boldsymbol{g}^{s\to f}(\boldsymbol{x}_{s,j})K(\boldsymbol{x}_{s,j}-\boldsymbol{x}_f)\Delta v,\end{aligned} \tag{45}$$

where D_f is a set of indices containing the solid samples around a fluid node at $\boldsymbol{x}_f$, and $\boldsymbol{x}_{s,j}$ are solid sample locations within the set. These forces are added to the kinetic solver to update the fluid momentum.

The specific form of $K(\boldsymbol{x}_j - \boldsymbol{x}_i)$ is a smoothed approximation to the Dirac delta function. In 2D case, it is written as:

$$K(\boldsymbol{x}_j - \boldsymbol{x}_i) = \frac{1}{\Delta x}\hat{\delta}\left(\frac{|x_j - x_i|}{\Delta x}\right)\frac{1}{\Delta y}\hat{\delta}\left(\frac{|y_j - y_i|}{\Delta y}\right), \tag{46}$$

where x and y are components of $\boldsymbol{x}$ in 2D, and similarly in 3D case, it is written as:

$$K(\boldsymbol{x}_j - \boldsymbol{x}_i) = \frac{1}{\Delta x}\hat{\delta}\left(\frac{|x_j - x_i|}{\Delta x}\right)\frac{1}{\Delta y}\hat{\delta}\left(\frac{|y_j - y_i|}{\Delta y}\right)\frac{1}{\Delta z}\hat{\delta}\left(\frac{|z_j - z_i|}{\Delta z}\right). \tag{47}$$

where x, y and z are components of $\boldsymbol{x}$ in 3D. $\hat{\delta}$ is a function defined as [14]:

$$\hat{\delta}(r) = \begin{cases} \frac{1}{3}(1+\sqrt{-3r^2+1}), & 0 \le r \le 0.5, \\ \frac{1}{6}(5-3r-\sqrt{-3(1-r)^2+1}), & 0.5 \le r \le 1.5, \\ 0, & \text{otherwise}, \end{cases} \tag{48}$$

with an interpolation kernel size 3. To preserve flow sharpness around solid boundaries, we can also we adopt a kernel size of 2, which we use in our paper:

$$\hat{\delta}(r) = \begin{cases} 1-r, & 0 \le r \le 1, \\ 0, & \text{otherwise}. \end{cases} \tag{49}$$

Note that due to normalization in LBE, $\Delta x = \Delta y = \Delta z = \Delta t = 1$, leading to $\Delta v = 1$.

VII. Implementation Details

In summary, we can implement the kinetic method with ACM-MRT model in the following steps:

a) Initialization: Given initial fields for ρ and $\boldsymbol{u}$, we can initialize all f_i based on $f_i^{eq}(\rho, \boldsymbol{u})$ at each grid node. The initial distribution of ρ and $\boldsymbol{u}$ could be arbitrary, and if large gradients exist in initialization of $\boldsymbol{u}$, initial compression waves will appear, but will soon be dampened, and the whole solution will converge to a weakly compressible setting to approximate incompressibility in many fluid simulation used in graphics. Due to use of immersed boundary method, the solid surfaces are sampled uniformly for handling boundary condition by penalty forces in later stage.

b) Streaming: We perform streaming of f_i based on Eq.(29) by accessing the corresponding f_i of the nearby fluid nodes.

c) Computing macroscopic quantities: We obtain the new macroscopic quantities by Eq.(24) (mainly for ρ and $\boldsymbol{u}$) to proceed the computation in boundary treatment and collision.

d) Boundary treatment by immersed boundary method: We apply immersed boundary method around solid boundaries to enforce the required condition, which basically involves an interpolation step (Eq.(43)) to interpolate fluid velocities at solid surface samples where a penalty force is computed at each sample (Eq.(44)) and a spreading step (Eq.(45)) to spread the penalty force over the nearby fluid nodes. The interpolation and spreading are done by the same conservative kernel $K(r)$.

e) Collision: Then, we compute collision by Eq.(34), where we use central moment projection matrix for $\boldsymbol{M}$. The inverse of $\boldsymbol{M}$ can be computed analytically, but could be algebraically complex (very long expression of polynomials with monomials). In general, we would like to write explicitly for each term in $\boldsymbol{\Omega}$, but it is too long to list all the items. Here we only list the complete form of Ω_{26}, with the forms for other terms easily obtained by Matlab or Mathematica software:

$$\begin{aligned}
\Omega_{26} = {} & \tilde{f}_{20}/8 - \tilde{f}_{16}/8 + \tilde{f}_{21}/8 + \tilde{f}_{22}/8 - \tilde{f}_{23}/8 - \tilde{f}_{24}/8 - \tilde{f}_{25}/8 + \tilde{f}_{26}/8 - (\tilde{f}_5\boldsymbol{u}_y)/8 + (\tilde{f}_{10}\boldsymbol{u}_y)/16 + (\tilde{f}_{12}\boldsymbol{u}_y)/16 \\
& - (\tilde{f}_{13}\boldsymbol{u}_y)/16 + (\tilde{f}_{15}\boldsymbol{u}_y)/16 + (\tilde{f}_{16}\boldsymbol{u}_y)/4 - (\tilde{f}_{17}\boldsymbol{u}_y)/32 - (\tilde{f}_{18}\boldsymbol{u}_y)/32 + (\tilde{f}_{19}\boldsymbol{u}_y)/16 - (\tilde{f}_{20}\boldsymbol{u}_y)/4 - (\tilde{f}_{22}\boldsymbol{u}_y)/4 \\
& + (\tilde{f}_{24}\boldsymbol{u}_y)/4 - (\tilde{f}_4\boldsymbol{u}_z)/8 + (\tilde{f}_{10}\boldsymbol{u}_z)/16 + (\tilde{f}_{11}\boldsymbol{u}_z)/16 + (\tilde{f}_{13}\boldsymbol{u}_z)/16 + (\tilde{f}_{14}\boldsymbol{u}_z)/16 + (\tilde{f}_{16}\boldsymbol{u}_z)/4 - (\tilde{f}_{17}\boldsymbol{u}_z)/32 \\
& - (\tilde{f}_{18}\boldsymbol{u}_z)/32 - (\tilde{f}_{19}\boldsymbol{u}_z)/16 - (\tilde{f}_{20}\boldsymbol{u}_z)/4 - (\tilde{f}_{21}\boldsymbol{u}_z)/4 + (\tilde{f}_{25}\boldsymbol{u}_z)/4 + (\tilde{f}_6{\boldsymbol{u}_x}^2)/8 - (\tilde{f}_{11}{\boldsymbol{u}_x}^2)/16 - (\tilde{f}_{12}{\boldsymbol{u}_x}^2)/16 \\
& + (\tilde{f}_{14}{\boldsymbol{u}_x}^2)/16 + (\tilde{f}_{15}{\boldsymbol{u}_x}^2)/16 + (\tilde{f}_{17}{\boldsymbol{u}_x}^2)/16 - (\tilde{f}_{18}{\boldsymbol{u}_x}^2)/16 + (\tilde{f}_5{\boldsymbol{u}_y}^2)/8 - (\tilde{f}_{10}{\boldsymbol{u}_y}^2)/16 - (\tilde{f}_{12}{\boldsymbol{u}_y}^2)/16 \\
& + (\tilde{f}_{13}{\boldsymbol{u}_y}^2)/16 - (\tilde{f}_{15}{\boldsymbol{u}_y}^2)/16 + (\tilde{f}_{17}{\boldsymbol{u}_y}^2)/32 + (\tilde{f}_{18}{\boldsymbol{u}_y}^2)/32 - (\tilde{f}_{19}{\boldsymbol{u}_y}^2)/16 + (\tilde{f}_4{\boldsymbol{u}_z}^2)/8 - (\tilde{f}_{10}{\boldsymbol{u}_z}^2)/16 \\
& - (\tilde{f}_{11}{\boldsymbol{u}_z}^2)/16 - (\tilde{f}_{13}{\boldsymbol{u}_z}^2)/16 - (\tilde{f}_{14}{\boldsymbol{u}_z}^2)/16 + (\tilde{f}_{17}{\boldsymbol{u}_z}^2)/32 + (\tilde{f}_{18}{\boldsymbol{u}_z}^2)/32 + (\tilde{f}_{19}{\boldsymbol{u}_z}^2)/16 - (\tilde{f}_6\boldsymbol{u}_x)/8 \\
& + (\tilde{f}_{11}\boldsymbol{u}_x)/16 + (\tilde{f}_{12}\boldsymbol{u}_x)/16 - (\tilde{f}_{14}\boldsymbol{u}_x)/16 - (\tilde{f}_{15}\boldsymbol{u}_x)/16 + (\tilde{f}_{16}\boldsymbol{u}_x)/4 - (\tilde{f}_{17}\boldsymbol{u}_x)/16 + (\tilde{f}_{18}\boldsymbol{u}_x)/16 - (\tilde{f}_{21}\boldsymbol{u}_x)/4 \\
& - (\tilde{f}_{22}\boldsymbol{u}_x)/4 + (\tilde{f}_{23}\boldsymbol{u}_x)/4 + (\tilde{f}_7{\boldsymbol{u}_x}^2{\boldsymbol{u}_y}^2)/24 - (\tilde{f}_8{\boldsymbol{u}_x}^2{\boldsymbol{u}_y}^2)/12 + (\tilde{f}_9{\boldsymbol{u}_x}^2{\boldsymbol{u}_y}^2)/24 - (\tilde{f}_7{\boldsymbol{u}_x}^2{\boldsymbol{u}_z}^2)/12 + (\tilde{f}_8{\boldsymbol{u}_x}^2{\boldsymbol{u}_z}^2)/24 \\
& + (\tilde{f}_9{\boldsymbol{u}_x}^2{\boldsymbol{u}_z}^2)/24 + (\tilde{f}_7{\boldsymbol{u}_y}^2{\boldsymbol{u}_z}^2)/24 + (\tilde{f}_8{\boldsymbol{u}_y}^2{\boldsymbol{u}_z}^2)/24 + (\tilde{f}_9{\boldsymbol{u}_y}^2{\boldsymbol{u}_z}^2)/24 + (\tilde{f}_5\boldsymbol{u}_x\boldsymbol{u}_y)/4 + (\tilde{f}_6\boldsymbol{u}_x\boldsymbol{u}_y)/4 + (\tilde{f}_7\boldsymbol{u}_x\boldsymbol{u}_y)/24 \\
& - (\tilde{f}_8\boldsymbol{u}_x\boldsymbol{u}_y)/12 + (\tilde{f}_9\boldsymbol{u}_x\boldsymbol{u}_y)/24 - (\tilde{f}_{10}\boldsymbol{u}_x\boldsymbol{u}_y)/8 - (\tilde{f}_{11}\boldsymbol{u}_x\boldsymbol{u}_y)/8 + (\tilde{f}_{13}\boldsymbol{u}_x\boldsymbol{u}_y)/8 + (\tilde{f}_{14}\boldsymbol{u}_x\boldsymbol{u}_y)/8 - (\tilde{f}_{16}\boldsymbol{u}_x\boldsymbol{u}_y)/2 \\
& + (\tilde{f}_{22}\boldsymbol{u}_x\boldsymbol{u}_y)/2 + (\tilde{f}_4\boldsymbol{u}_x\boldsymbol{u}_z)/4 + (\tilde{f}_6\boldsymbol{u}_x\boldsymbol{u}_z)/4 - (\tilde{f}_7\boldsymbol{u}_x\boldsymbol{u}_z)/12 + (\tilde{f}_8\boldsymbol{u}_x\boldsymbol{u}_z)/24 + (\tilde{f}_9\boldsymbol{u}_x\boldsymbol{u}_z)/24 - (\tilde{f}_{10}\boldsymbol{u}_x\boldsymbol{u}_z)/8 \\
& - (\tilde{f}_{12}\boldsymbol{u}_x\boldsymbol{u}_z)/8 - (\tilde{f}_{13}\boldsymbol{u}_x\boldsymbol{u}_z)/8 + (\tilde{f}_{15}\boldsymbol{u}_x\boldsymbol{u}_z)/8 - (\tilde{f}_{16}\boldsymbol{u}_x\boldsymbol{u}_z)/2 + (\tilde{f}_{21}\boldsymbol{u}_x\boldsymbol{u}_z)/2 + (\tilde{f}_4\boldsymbol{u}_y\boldsymbol{u}_z)/4 + (\tilde{f}_5\boldsymbol{u}_y\boldsymbol{u}_z)/4
\end{aligned}$$

$$
\begin{aligned}
&+(\tilde{f}_7\boldsymbol{u}_y\boldsymbol{u}_z)/24+(\tilde{f}_8\boldsymbol{u}_y\boldsymbol{u}_z)/24+(\tilde{f}_9\boldsymbol{u}_y\boldsymbol{u}_z)/24-(\tilde{f}_{11}\boldsymbol{u}_y\boldsymbol{u}_z)/8-(\tilde{f}_{12}\boldsymbol{u}_y\boldsymbol{u}_z)/8-(\tilde{f}_{14}\boldsymbol{u}_y\boldsymbol{u}_z)/8-(\tilde{f}_{15}\boldsymbol{u}_y\boldsymbol{u}_z)/8\\
&-(\tilde{f}_{16}\boldsymbol{u}_y\boldsymbol{u}_z)/2+(\tilde{f}_{20}\boldsymbol{u}_y\boldsymbol{u}_z)/2-(\tilde{f}_5\boldsymbol{u}_x{\boldsymbol{u}_y}^2)/4-(\tilde{f}_6{\boldsymbol{u}_x}^2\boldsymbol{u}_y)/4-(\tilde{f}_7\boldsymbol{u}_x{\boldsymbol{u}_y}^2)/24-(\tilde{f}_7{\boldsymbol{u}_x}^2\boldsymbol{u}_y)/24+(\tilde{f}_8\boldsymbol{u}_x{\boldsymbol{u}_y}^2)/12\\
&+(\tilde{f}_8{\boldsymbol{u}_x}^2\boldsymbol{u}_y)/12-(\tilde{f}_9\boldsymbol{u}_x{\boldsymbol{u}_y}^2)/24-(\tilde{f}_9{\boldsymbol{u}_x}^2\boldsymbol{u}_y)/24+(\tilde{f}_{10}\boldsymbol{u}_x{\boldsymbol{u}_y}^2)/8+(\tilde{f}_{11}{\boldsymbol{u}_x}^2\boldsymbol{u}_y)/8-(\tilde{f}_{13}\boldsymbol{u}_x{\boldsymbol{u}_y}^2)/8\\
&-(\tilde{f}_{14}{\boldsymbol{u}_x}^2\boldsymbol{u}_y)/8-(\tilde{f}_4\boldsymbol{u}_x{\boldsymbol{u}_z}^2)/4-(\tilde{f}_6{\boldsymbol{u}_x}^2\boldsymbol{u}_z)/4+(\tilde{f}_7\boldsymbol{u}_x{\boldsymbol{u}_z}^2)/12+(\tilde{f}_7{\boldsymbol{u}_x}^2\boldsymbol{u}_z)/12-(\tilde{f}_8\boldsymbol{u}_x{\boldsymbol{u}_z}^2)/24-(\tilde{f}_8{\boldsymbol{u}_x}^2\boldsymbol{u}_z)/24\\
&-(\tilde{f}_9\boldsymbol{u}_x{\boldsymbol{u}_z}^2)/24-(\tilde{f}_9{\boldsymbol{u}_x}^2\boldsymbol{u}_z)/24+(\tilde{f}_{10}\boldsymbol{u}_x{\boldsymbol{u}_z}^2)/8+(\tilde{f}_{12}{\boldsymbol{u}_x}^2\boldsymbol{u}_z)/8+(\tilde{f}_{13}\boldsymbol{u}_x{\boldsymbol{u}_z}^2)/8-(\tilde{f}_{15}{\boldsymbol{u}_x}^2\boldsymbol{u}_z)/8-(\tilde{f}_4\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/4\\
&-(\tilde{f}_5{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/4-(\tilde{f}_7\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/24-(\tilde{f}_7{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/24-(\tilde{f}_8\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/24-(\tilde{f}_8{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/24-(\tilde{f}_9\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/24-(\tilde{f}_9{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/24\\
&+(\tilde{f}_{11}\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/8+(\tilde{f}_{12}{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/8+(\tilde{f}_{14}\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/8+(\tilde{f}_{15}{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/8+(R\boldsymbol{u}_x\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/8+(R\boldsymbol{u}_x{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/8\\
&+(R{\boldsymbol{u}_x}^2\boldsymbol{u}_y\boldsymbol{u}_z)/8+(\tilde{f}_4\boldsymbol{u}_x\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/2+(\tilde{f}_5\boldsymbol{u}_x{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/2+(\tilde{f}_6{\boldsymbol{u}_x}^2\boldsymbol{u}_y\boldsymbol{u}_z)/2-(R\boldsymbol{u}_x{\boldsymbol{u}_y}^2{\boldsymbol{u}_z}^2)/8-(R{\boldsymbol{u}_x}^2\boldsymbol{u}_y{\boldsymbol{u}_z}^2)/8\\
&-(R{\boldsymbol{u}_x}^2{\boldsymbol{u}_y}^2\boldsymbol{u}_z)/8-(R\boldsymbol{u}_x\boldsymbol{u}_y\boldsymbol{u}_z)/8-(\tilde{f}_4\boldsymbol{u}_x\boldsymbol{u}_y\boldsymbol{u}_z)/2-(\tilde{f}_5\boldsymbol{u}_x\boldsymbol{u}_y\boldsymbol{u}_z)/2-(\tilde{f}_6\boldsymbol{u}_x\boldsymbol{u}_y\boldsymbol{u}_z)/2+\tilde{f}_{16}\boldsymbol{u}_x\boldsymbol{u}_y\boldsymbol{u}_z\\
&+(R{\boldsymbol{u}_x}^2{\boldsymbol{u}_y}^2{\boldsymbol{u}_z}^2)/8\ ,
\end{aligned}
$$

where $\tilde{f}_i = [\boldsymbol{DM}(\boldsymbol{f}-\boldsymbol{f}^{eq})]_i$, and the operator $[\cdot]$ picks the i-th element out of a vector. When translating these expressions into actual codes in CUDA kernel threads, they occupy a lot of local resources (registers and local memories), thus we have done some simplification by grouping terms with the basic monomial basis, which reduces the computation by around 10%. After collision calculation, we can add force term G_i, which is computed by Eq.(42).

f) Treating domain boundary: Finally, we will treat domain boundary by bounce-back schemes [15]. There could be three kinds of domain boundaries: inlets, outlets and walls. For inlets and outlets, we can use Zou-He boundary treatment [16], while for walls, we use standard bounce-back scheme. In our main paper, we do not consider Zou-He boundary, and use standard bounce-back scheme instead.

REFERENCES

[1] J. A. McLennan, *Introduction to nonequilibrium statistical mechanics*. Prentice Hall Englewood Cliffs, 1989.

[2] B. Dünweg, U. D. Schiller, and A. J. Ladd, "Statistical mechanics of the fluctuating lattice Boltzmann equation," *Physical Review E*, vol. 76, no. 3, p. 036704, 2007.

[3] S. Chen and G. D. Doolen, "Lattice Boltzmann method for fluid flows," *Annual Review of Fluid Mechanics*, vol. 30, no. 1, pp. 329–364, 1998.

[4] X. Shan, X.-F. Yuan, and H. Chen, "Kinetic theory representation of hydrodynamics: a way beyond the Navier-Stokes equation," *Journal of Fluid Mechanics*, vol. 550, pp. 413–441, 2006.

[5] D. d'Humieres, "Multiple–relaxation–time lattice Boltzmann models in three dimensions," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 360, no. 1792, pp. 437–451, 2002.

[6] M. Geier, A. Greiner, and J. G. Korvink, "A factorized central moment lattice Boltzmann method," *The European Physical Journal Special Topics*, vol. 171, no. 1, pp. 55–61, 2009.

[7] K. N. Premnath and S. Banerjee, "On the three-dimensional central moment lattice Boltzmann method," *Journal of Statistical Physics*, vol. 143, no. 4, pp. 747–794.

[8] A. De Rosis, "Nonorthogonal central-moments-based lattice Boltzmann scheme in three dimensions," *Physical Review E*, vol. 95, no. 1, p. 013310, 2017.

[9] W. Li, K. Bai, and X. Liu, "Continuous-scale kinetic fluid simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 9, pp. 2694–2709, Sep. 2019.

[10] W. Li, Y. Chen, M. Desbrun, C. Zheng, and X. Liu, "Fast and scalable turbulent flow simulation with two-way coupling," *ACM Transactions on Graphics (SIGGRAPH 2020)*, vol. 39, no. 4, 2020.

[11] Z. Guo, C. Zheng, and B. Shi, "Discrete lattice effects on the forcing term in the lattice Boltzmann method," *Physical Review E*, vol. 65, no. 4, p. 046308, 2002.

[12] J. Latt, B. Chopard, O. Malaspinas, M. Deville, and A. Michler, "Straight velocity boundaries in the lattice Boltzmann method," *Physical Review E*, vol. 77, no. 5, p. 056703, 2008.

[13] C. Yuksel, "Sample elimination for generating Poisson disk sample sets," in *Computer Graphics Forum*, vol. 34, no. 2. Wiley Online Library, 2015, pp. 25–32.

[14] Z. Li, J. Favier, U. D'Ortona, and S. Poncet, "An immersed boundary-lattice Boltzmann method for single-and multi-component fluid flows," *Journal of Computational Physics*, vol. 304, pp. 424–440, 2016.

[15] S. Girimaji, "Lattice Boltzmann method: Fundamentals and engineering applications with computer codes," 2012.

[16] Q. Zou and X. He, "On pressure and velocity boundary conditions for the lattice Boltzmann BGK model," *Physics of Fluids*, vol. 9, no. 6, pp. 1591–1598, 1997.