Memory-efficient Lattice Boltzmann Method for low Reynolds number flows

Maciej Matyka[a,*], Michał Dzikowski[b]

[a] *Faculty of Physics and Astronomy, University of Wrocław, pl. M. Borna 9, 50-205 Wrocław, Poland, tel.: +48713759357, fax: +48713217682*
[b] *Interdisciplinary Centre for Mathematical and Computational Modelling, UW, ul. Tyniecka 15/17, 02-630, Warsaw, Poland*

Abstract

The Lattice Boltzmann Method algorithm is simplified by assuming constant numerical viscosity (the relaxation time is fixed at $\tau = 1$). This leads to the removal of the distribution function from the computer memory. To test the solver the Poiseuille and Driven Cavity flows are simulated and analyzed. The error of the solution decreases with the grid size L as L^{-2}. Compared to the standard algorithm, the presented formulation is simpler and shorter in implementation. It is less error-prone and needs significantly less working memory in low Reynolds number flows. Our tests showed that the algorithm is less efficient in multiphase flows. To overcome this problem, further extension and the moments-only formulation was derived, inspired by the Multi-Relaxation Time (MRT) approach for single component multiphase flows.

Keywords: Lattice Boltzmann method, LBM, CFD, memory

1. Introduction

Computational fluid dynamics (CFD) is useful in many branches of science and technology, including those related to main civilization challenges of the utmost importance for the whole society e.g. weather forecast, climate, sport, medicine, oil recovery, and food industry [1, 2, 3]. The most popular computational methods for CFD simulations are based on the direct discretization of the Navier-Stokes equations using appropriate numerical methods, e.g. finite differences, finite volumes, or finite elements [4]. They are usually difficult to implement and require large computer resources as well as some tedious preprocessing of the input data (e.g. generation and storage of complex computational grids). In this context, the mesoscopic Lattice Boltzmann Method (LBM), based on the kinetic theory of gases, has recently been gaining more and more attention as a versatile and simple fluid solver that offers a wide range of potential applications [5]. One of the main limitations of the original LBM algorithm is a relatively high computer memory demand, as one has to store the distribution function for all fluid nodes. This limits the size of the samples that can be simulated in a single machine. Also, an increased number of memory accesses and complex memory access patterns in the propagation of distribution function may form a bottleneck for the parallel acceleration of the LBM [6]. It was shown that in a GPU implementation the efficiency of the LBM solver saturates with the filling memory fraction [7]. Therefore, much research has been focused on improving memory efficiency of the LBM algorithm, including modifications of the main LBM algorithm [8, 9] or data format and algorithms for sparse environments where most of the cells are getting fully blocked by obstacles [10, 11].

In the standard LBM BGK algorithm the relaxation time τ may range from nearly 1/2 up to 1 (highly viscous flows), however, the choice $\tau = 1$ is often made in the single relaxation time BGK approximation [12, 13, 14]. For example, $\tau = 1$ was chosen in the gray LBM model used for porous media flows [15], to compute the first predictor step and fictitious viscosity solution in the simulation of the mold filling process [16], in the LBM multicomponent flow simulation with comparison to Finite Volume Methods [17] or in the immersed-boundary LBM for particles suspended in fluids [18]. It was shown that the value of τ influences the accuracy of the LBM solver in flow through narrow pores [13], and, $\tau = 1$ case was in the best agreement with advanced multi-relaxation time schemes [19]. This is also a special case for multiphase flows where EDM (Exact Difference Method) agrees well with the Shan-Chen method in terms of measured gas density error (which is minimum at $\tau = 1$ for the Shan-Chen model) [20]. Moreover, it was reported as the best choice for the Shan-Chen two immiscible fluid simulation [21]. Also, it was shown that setting $\tau = 1$ in BGK LBM provides optimal accuracy in time if solutions are compared to direct Navier-Stokes equations [22]. Setting $\tau = 1$ is also crucial for the fractional step formulation of LBM for high Reynolds flows [23].

Here we fix the viscosity of the model and set the relaxation time to $\tau = 1$ (from now We will use the codename LBM-Tau1) and modify the original LBM algorithm to a simpler, more compact and memory-efficient (an approach found i.e. in [24, 25]). we provide a complete algorithm and first test it against Poiseuille flow with error scaling analysis. Then, we continue the with the flow tests at varying Reynolds number and formulate criteria to calculate grid size necessary for stable simulations. We further develop the solver and use the multi relaxation time (MRT) version of the model at $\tau = 1$ and compare its efficiency to the standard algorithm. we show, that this

*Corresponding author
Email address: maciej.matyka@uwr.edu.pl (Maciej Matyka)

approach leads to a significant memory drop and analyze this effect for various conditions and LBM models.

2. The Model

The Lattice Boltzmann Method use the multi-dimensional velocity distribution function $f_k(\mathbf{x}, t)$ to describe the state of the fluid. Function $f_k(\mathbf{x}, t)$ corresponds to the probability that a molecule at position $\mathbf{x}$ at time t, is moving with velocity $\mathbf{e}_k$. The original LBM algorithm consists of two steps: propagation and relaxation of the distribution function. It may be written as a discrete analogon to the Boltzmann transport equation (here with a linear approximation for the collision term) [26]:

$$f_{\mathrm{k}}(\mathbf{x} + \mathbf{e}_{\mathrm{k}}, t + 1) = f_{\mathrm{k}}(\mathbf{x}, t) - \frac{f_{\mathrm{k}}(\mathbf{x}, t) - f_{\mathrm{k}}^{eq}(\mathbf{x}, t)}{\tau}, \quad (1)$$

where k is the direction on the lattice, f^{eq} is the equilibrium distribution function, $\mathbf{e}_{\mathrm{k}}$ is the lattice vector and τ is the relaxation time. By varying τ, the kinematic viscosity of the fluid may be controlled [26]

$$v = c_s^2 \, (\tau - 0.5) \, , \quad (2)$$

where c_s is the sound speed (dependent on the variant of the model, e.g. $c_s = 1/\sqrt{3}$ for two dimensional D2Q9 model [26]). The equilibrium f^{eq} is expressed in terms of macroscopic density ϱ and velocity $\mathbf{u}$ of the flow field:

$$f_k^{eq} = \omega_k \varrho \left(1 + 3\mathbf{e}_k \cdot \mathbf{u} + \frac{9}{2}(\mathbf{e}_k \cdot \mathbf{u})^2 - \frac{3}{2}\mathbf{u}^2 \right), \quad (3)$$

where ω_k are direction weights [27]. To include body force we may modify directly the momentum used for calculation of equilibrium (see e.g. [28]). The following sums over the distribution function let us compute density and velocity:

$$\varrho(\mathbf{x}, t) = \sum_k f_k(\mathbf{x}, t), \quad (4)$$

$$\mathbf{u}(\mathbf{x}, t) = \sum_k \mathbf{e}_k f_k(\mathbf{x}, t). \quad (5)$$

To save computer memory and eliminate the distribution function, we will first fix the relaxation time at $\tau = 1$ [29, 25]. With this assumption the transport equation (see Eq. (1)) simplifies to:

$$f_{\mathrm{k}}(\mathbf{x} + \mathbf{e}_{\mathrm{k}}, t + 1) = f_{\mathrm{k}}^{eq}(\mathbf{x}, t). \quad (6)$$

Now, instead of keeping the values of f_{k} in memory, we plug Eq. (6) into (4) and (5). Thus, a new formulation will consist of computing macroscopic fields from the equilibrium distribution only

$$\varrho(\mathbf{x}, t)|_{\tau=1} = \sum_k f_{ik}^{eq}(\mathbf{x} + \mathbf{e}_{\mathrm{k}}, t), \quad (7)$$

$$\mathbf{u}(\mathbf{x}, t)|_{\tau=1} = \sum_k f_{ik}^{eq}(\mathbf{x} + \mathbf{e}_{\mathrm{k}}, t) \cdot \mathbf{e}_k. \quad (8)$$

where f^{eq} is the Maxwell-Boltzmann distribution defined by Eq. 3. With the above two equations, we can write down an algorithm in which step by step the equilibrium distribution is computed from the macroscopic velocity and density and then use these values to make another iteration. In this way, the storage of f_{k} is eliminated.

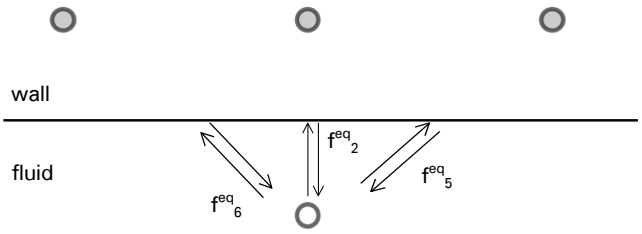

Figure 1: Calculation of the equilibrium function next to the no-slip wall.

For simplicity, we restrict our discussion to the D2Q9 model [26] (a two dimensional LBM model with nine lattice velocities $\mathbf{e}_{\mathrm{k}}$), where

$$\mathbf{e}_{\mathrm{k}} \in \{(0,0), (1,0), (0,1), (-1,0), (0,-1), (1,1), (-1,1), (-1,-1), (1,-1)\}, \quad (9)$$

for $k = 0 \ldots 8$ respectively.

To complete the picture, we need to account for the boundary conditions at the no-slip (zero tangent velocity) walls. For fluid nodes located next to a no-slip wall, the normal components of the equilibrium distribution function must be reversed (see Fig. 1) and used in equations (7) and (8). If we are at fluid node at $\mathbf{x}$ and the node $\mathbf{x} + \mathbf{e}_{\mathrm{k}}$ is of the no-slip type (a solid wall) we must use the following expression for f_{ik}^{eq} in Eqs. (7) and (8)

$$f_{\mathrm{ik}}^{eq}(\mathbf{x} + \mathbf{e}_{\mathrm{k}}) = f_{\mathrm{k}}^{eq}(\mathbf{x}, \mathrm{u} = 0, \mathrm{v} = 0) = \omega_{\mathrm{k}} \cdot \varrho(\mathrm{x}). \quad (10)$$

For example, for the wall located at the north, we need to reverse three populations that move towards the wall: $f_6^{eq}(\mathbf{x})$, $f_2^{eq}(\mathbf{x})$ and $f_5^{eq}(\mathbf{x})$ (see Fig. 1). Thus, for the north wall being no-slip we will have

$$\varrho_{\tau_1}(\mathbf{x}, t) = f_3^{eq}(\mathbf{x} + \mathbf{e}_1) + f_1^{eq}(\mathbf{x} + \mathbf{e}_3) + f_2^{eq}(\mathbf{x} + \mathbf{e}_4) + f_6^{eq}(\mathbf{x} + \mathbf{e}_8) + f_5^{eq}(\mathbf{x} + \mathbf{e}_7) + f_6^{eq}(\mathbf{x}) + f_2^{eq}(\mathbf{x}) + f_5^{eq}(\mathbf{x}). \quad (11)$$

The first five terms on the right-hand side in the above equation are standard incoming populations from neighboring nodes, whereas the three last terms are the populations reflected from the northern wall and computed in-place at the node $\mathbf{x}$. This procedure is used for all nodes adjacent to the walls. However, we do not need to write down an explicit expression for each orientation of the wall - it may be implemented by a simple expression in the algorithm. One has to check, if the neighboring node is a wall or not and choose Eq. (3) or (10) for equilibrium, accordingly.

2.1. The LBMTau1 algorithm

Using the procedure introduced in the previous section and summarized by Eqs. (7) and (8), we formulate a complete algorithm for the LBMTau1 solver. Here R_c, U_c and V_c are the macroscopic density, velocity (x) and velocity (y) fields at even

(c=0) or odd (c=1) time steps. Thus, subscript c is equal to 0 or 1 and denotes the grid number (we keep two copies of the grid to ping-pong data in the memory). Variables i, j and i_p, j_p are grid coordinates. Within the algorithm, we use the macroscopic density, velocity, lattice vector components $e_{k,x}$ and $e_{k,y}$ and the grid direction weights ω_{ik} to calculate the equilibrium distribution function f_{ik}^{eq}.

Algorithm 1 Complete time step of the LBMTau1 algorithm.

```
    initialize flags and fields for fluid and solid nodes
    for all fluid nodes (i, j) do
      U_c(i, j) ← 0
      V_c(i, j) ← 0
 5:   R_c(i, j) ← 0
      for all directions k do
        i_p ← i + e_{k,x} (neighbour in the direction k)
        j_p ← j + e_{k,y}
        i_k ← direction inverse to k
10:     if (i_p, i_p) is fluid node then
          r ← R_{1−c}(i_p, j_p)
          u ← (U_{1−c}(i_p, j_p) + f_x) / r
          v ← (V_{1−c}(i_p, j_p) + f_y) / r
          f_ik^eq ← ω_ik r·
15:       (1 − 3/2(u^2 + v^2) + 3(e_ik^x u + e_ik^y v) + 9/2(e_ik^x u + e_ik^y v)^2)
        else
          f_ik^eq ← ω_k R_{1−c}(i, j)
        end if
      end for
20:   R_c(i, j) ← R_c(i, j) + f_ik^eq
      U_c(i, j) ← U_c(i, j) + e_ik^x f_ik^eq
      V_c(i, j) ← V_c(i, j) + e_ik^y f_ik^eq
    end for
    Visualization (optional)
```

The algorithm starts with the initialization of macroscopic fields (line 1). At this point, we set up the flags for each node (flags denote if the node is occupied by fluid or solid). Also, the initial velocity and density fields are set up here (we start from zero velocity condition and density set to one). Next, we start the main loop over all fluid nodes (line 2) and for each of them compute the equilibrium distribution function from the local velocity and density. We also include the body force (lines 12-13 of the Algorithm 1). In the case of solid walls, we compute the equilibrium function by reflecting its normal components and assume zero velocity (no-slip boundary condition, line 17). Finally, we update the density and velocity by adding the populations that are incoming or being reflected from neighboring cells (lines 20-22). Implementation of this algorithm is straightforward - it contains $D + 1$ tables (where D is the dimension of the model), two loops, and one conditional (see the exemplary C/C++ implementation in Appendix A).

3. Validation and Results

To verify the solver we run the steady-state flow in a straight rectangular two-dimensional channel and two-dimensional lid-driven cavity flows (standard tests performed in computational fluid dynamics). First, for the channel flow, we use the periodic conditions at the left and right system edges and the no-slip at the top and bottom walls. The external body force $f = 2.5 \cdot 10^{-5}$ (lattice units) was used to generate a steady flow along the channel axis. we used a 200×100 grid. To verify if the steady-state was reached we monitored the changes of velocity in the middle of the channel and used the convergence condition:

$$\frac{|u_{n-1} - u_n|}{|u_n|} < \varepsilon, \tag{12}$$

where $\varepsilon = 10^{-7}$. The resulting velocity profile along the channel crossection is given in Fig. 2. we find an excellent agreement between the numerical and analytical solutions. To quantify the agreement we repeat the simulations at varying grid size L and calculate the percentage error of the solution $e = 100 \cdot \frac{|u-\tilde{u}|}{|\tilde{u}|}$, where $\tilde{u}$ is the analytical value of velocity in the middle of the channel. we find that the error follows the power law and scales with the grid size as L^{-2} (see Fig. 3).

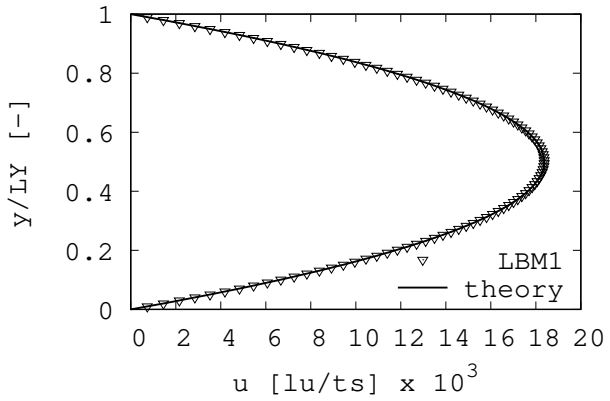

Figure 2: The velocity profile in the Poiseuille Flow simulated using the LBMTau1 algorithm (points) compared to the analytical solution (solid line). The profile was taken along the y-axis perpendicular to the flow direction.

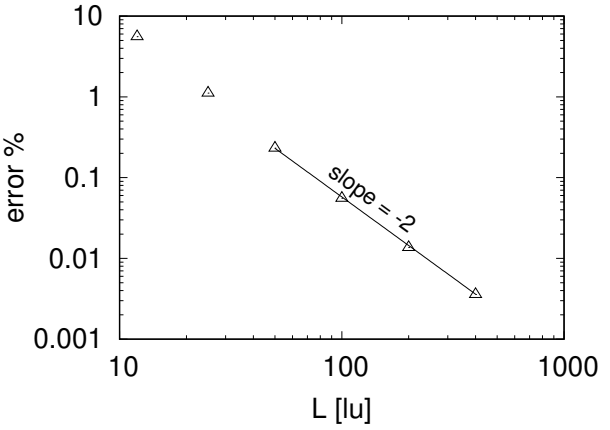

Figure 3: Scaling of the relative error between LBMTau1 and analytical solution (Poiseuille Flow). The solid line represent best fit to L^{-2} scaling taken at $L >= 40$.

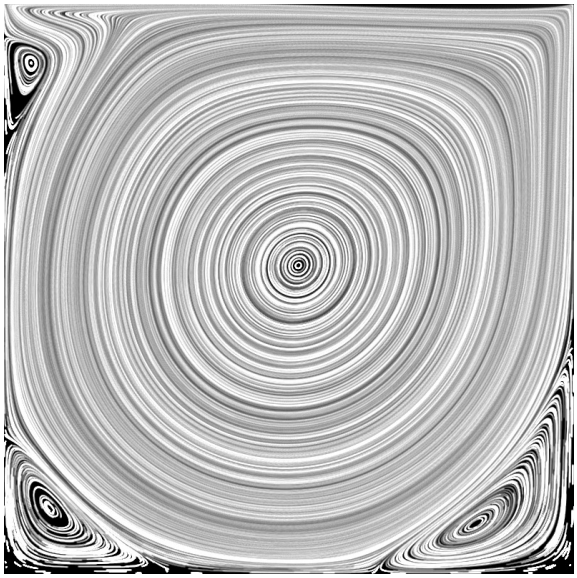

Figure 4: The lid-driven cavity test run at Re=3200 calculated using the LBMTau1 code (see Appendix A) on a 1100×1100 grid. Visualization was made using massless tracers advected on top of the velocity field.

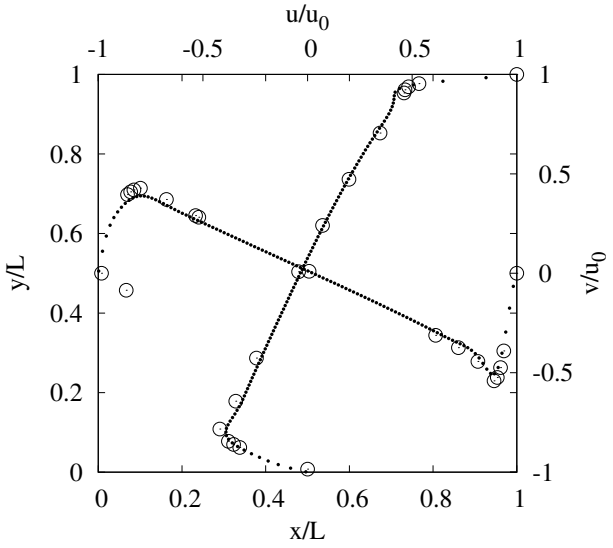

Figure 5: Velocity profiles in lid-driven cavity at $Re = 3200$. LBMTau1 results (solid dots) are compared to the literature benchmark data [31] (open circles).

Next, we run the standard lid-driven cavity flow in which the fluid is enclosed in a rectangular cavity with a top lid moving at a constant velocity [30]. The Dirichlet boundary condition $\mathbf{v}_{\text{lid}} = (u_0, 0)$ at the top boundary is applied. The no-slip condition is applied at the left, right, and bottom boundaries. We performed the simulation on a 1100×1100 grid and velocity $u_0 = 0.4844$ at Re=3200. The simulation was continued as long as relative changes in the volumetric flux across vertical cross-section located at half of the system were larger than 1%. We checked the change between two timesteps at $\Delta t = 1000$ interval. In Fig. 4 we draw the streamlines on top of the final velocity field. The main vortex in the middle of the cavity, as well as vortex structures in corners of the cavity, are visible. The quantitative comparison with the multigrid method is given in Fig. 5.

The final test of a time-dependent multiphase flow is the multiphase Shan-Chen model [32] in the LBMTau1 solver with the random initial conditions. The implementation of the Shan-Chen model was straightforward and we observed an expected phase separation effect (see Fig. 6). However, due to the two-fold loop over all neighbours of each computational node (once we need to go over neighbours and then over neighbours of each neighbour), we observed a significant drop in efficiency in the multiphase algorithm, if compared to standard LBM.

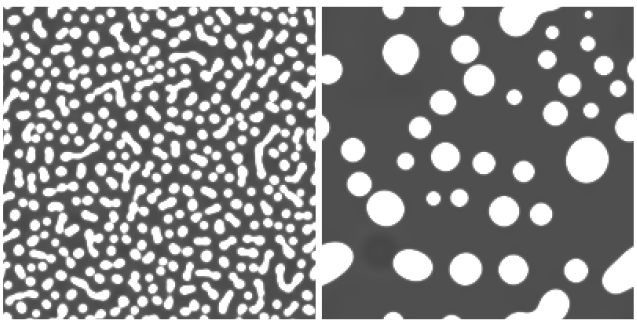

Figure 6: LBMTau1 implementation of the multiphase simulation using the Shan-Chen model.

4. Moments-only multirelaxation time LBMTau1

The significant bottleneck of LBMTau1 in terms of performance is an unnecessary evaluation of f^{eq} in each lattice node, which is mostly visible in the multiphase flows as found in the previous section. Let us look at evolution equation Eq. (6) at $\tau = 1$ (or $\omega = 1$), which could be rewritten as

$$f_j(\mathbf{x}, t + \Delta t) = f_j^{eq}(\rho(\mathbf{x} - \mathbf{e}_j, t), \mathbf{u}(\mathbf{x} - \mathbf{e}_j, t)),$$

to underline dependence on macroscopic variables. In the Multi-Relaxation Time (MRT) scheme, if we take the raw moments matrix $\mathbf{M} = M_{ij}$, and multiply the evolution equation, we obtain

$$M_{ij} f_j(\mathbf{x}, t + \Delta t) = M_{ij} f_j^{eq}(\rho(\mathbf{x} - \mathbf{e}_j, t), \mathbf{u}(\mathbf{x} - \mathbf{e}_j, t)).$$

Here, by definition the density is defined as:

$$\rho = M_{1j} f_j,$$

whereas momentum reads:

$$\rho \mathbf{u} = M_{kj} f_j \, , k = 2, 3[, 4].$$

As a consequence, the evolution equation is reduced to:

$$\rho(\mathbf{x}, t + \Delta t) = M_{1j} f_j^{eq}(\rho(\mathbf{x} - \mathbf{e}_j, t), \mathbf{u}(\mathbf{x} - \mathbf{e}_j, t)),$$

$$\rho \mathbf{u}(\mathbf{x}, t + \Delta t) = M_{kj} f_j^{eq}(\rho(\mathbf{x} - \mathbf{e}_j, t), \mathbf{u}(\mathbf{x} - \mathbf{e}_j, t)) \, , k = 2, 3[, 4].$$

Now we can derive closed formulas, and skip explicit evaluation of f^{eq} entirely. The main reason for speedup in such a case is that we do not evaluate unnecessary degrees of freedom of the system (higher-order moments) as they are relaxed to equilibrium either-way. Nevertheless, the number of arithmetic operations involved in the evaluation of density and momentum using those closed statements could be significant. To optimize further, those statements could be simplified using heuristic approaches and the computer algebra system of choice. Statements for the evolution of D2Q9 lattice, optimized by using PolyAlgebra package (part of TCLB software [33]) are given in supplementary material.

This approach could be used to optimize the Shan-Chen type multiphase models. If we consider interaction potential force in form :

$$\mathbf{F}_{SC} = \psi(\mathbf{x}) \sum_i \alpha_i \psi(\mathbf{x} + \mathbf{e}_i) G(\mathbf{e}_i), \tag{13}$$

then for $\tau = 1$, evolution equation could be once again written as: as

$$f_j(\mathbf{x}, t + \Delta t) = f_j^{eq}(\rho(\mathbf{x} - \mathbf{e}_j, t), \mathbf{u}(\mathbf{x} - \mathbf{e}_j, t), \mathbf{F}_{SC}(\mathbf{x} - \mathbf{e}_j, t)).$$

which gives similar final result:

$$\mathbf{M} f_j(\mathbf{x}, t + \Delta t) = \mathbf{M} f_j^{eq}(\rho(\mathbf{x} - \mathbf{e}_j, t), \mathbf{u}(\mathbf{x} - \mathbf{e}_j, t), \mathbf{F}_{SC}(\mathbf{x} - \mathbf{e}_j, t)).$$

For a single component multiphase model, two approaches to evaluate $\mathbf{F}_{SC}$ could be used. In the first, one stores ψ for latter use in neighbouring nodes in Eq. (13) loops. In the second, one recalculates ψ of neighbouring nodes in-place anytime it is necessary. We found, that the first method significantly speeds up the algorithm at memory cost of additional scalar field.

4.1. Performance evaluation

To evaluate the computational performance of presented models, we used a model of fluid drop enclosed in a periodic domain. For an inter-particle force, we used the equation proposed by Kupershtokh [34], which provide superior stability compared to original Shan-Chen scheme remaining identical from the implementation point of view, especially when $\tau = 1$. Details of both methods could be found in [34, 32].

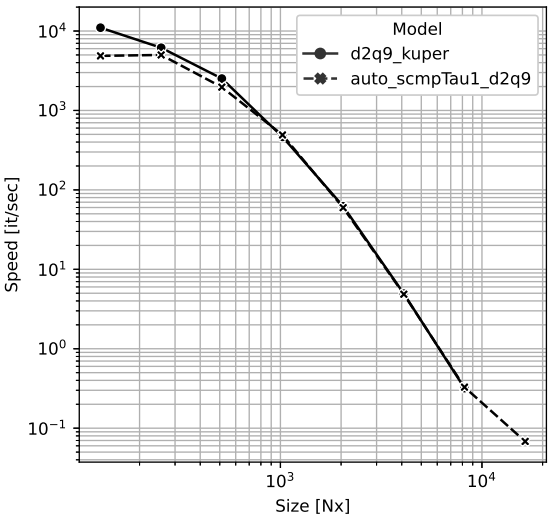

Figure 7: Iteration speed expressed in iteration per second for two types of solvers: LBMTau1 (crosses) and standard LBM implementation (filled circles). Both solvers were implemented in the TLCB framework [33].

A drop of liquid is placed in a periodic domain filled with vapor. We used $L \times L$ grids at L from 128 to 8192. The final, largest grid used has filled almost completely the GPU memory of the NVIDIA GPU V100 card used for all tests. Thus, it was impossible to calculate final, largest grid using standard LBM (which clearly showed the advantage of LBMTau1 formulation). The simulation was run for several iterations until average velocity reached a steady state.

To quantify solver efficiency we plot the drop test iteration speed in function of the size of the system (in lattice units) in Fig. 7. We compare the two types of solver [33, 35]: classical "d2q9_kuper" model where standard SCMP multiphase model is implemented (see [36] for details) and the LBMTau1 variant "auto_scmpTau1_d2q9" where no distribution function is used. Both simulations are carried out at the same viscosity and in the same two-dimensional square domain. The memory usage of both solvers is compared in Fig. 8.

4.2. Convergence and accuracy of LBMTau1 and standard LBM

The LBM method converges to the Navier-Stokes equation in terms of the small parameter ϵ used in the expansion, e.g. Chapman-Enskog procedure. Two scalings are possible: acoustics one $\epsilon = \Delta x = \Delta t$ and diffusive one $\epsilon = \Delta t = \Delta x^2$. To investigate theoretical memory usage at preserved accuracy we restrict ourselves to cases belonging to the same viscous scaling series. For two viscosities, ν_{LB} and $\nu_{LB1} = 1/6$ the Reynolds number puts a restriction on a both Δx and Δt. For two grids, denoted by LB (variable viscosity) and LB1 ($\tau = 1$), it is re-

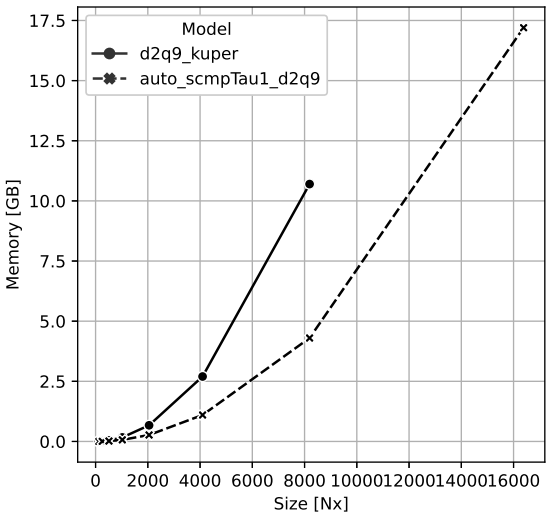

Figure 8: Memory overhead for two types of solvers (see caption in Fig. 7 for reference).

quired to

$$\frac{U_{LB1}L_{LB1}}{\nu_{LB1}} = \frac{U_{LB}L_{LB}}{\nu_{LB}},$$

where additionally diffusive scaling is defined as

$$\epsilon = \Delta t = \Delta x^2 \to 0.$$

We could now consider theoretical memory usage concerning non-fixed viscosity LBM. We define N_{LB} as grid resolution and T_{LB} as the number of iterations (simulation time). By comparing Reynolds number for LB and LB1 cases, one gets

$$\frac{T_{LB}N_{LB}^2}{T_{LB1}N_{LB1}^2} = 6\nu_{LB}.$$

On the other hand, from diffusive scaling time and spatial resolution are constrained by

$$T_{LB}N_{LB}^2 = \left(\Delta t \cdot \Delta x^2\right)^{-1} = \epsilon^{-2},$$

which after substitution into Eg. 4.2 gives

$$\left(\frac{\epsilon_{LB1}}{\epsilon}\right)^2 = 6\nu_{LB}.$$

From that we recover the spatial resolution ratio as

$$\frac{N_{LB}}{N_{LB1}} = \frac{\Delta x_{LB1}}{\Delta x_{LB}} = \sqrt{\frac{\epsilon_{LB1}}{\epsilon_{LB}}} = (6\nu_{LB})^{1/4}.$$

The ratio of the number of volume elements for LB case in relation to LB1 is equal to

$$N_{LB}^D = N_{LB1}^D (6\nu_{LB})^{D/4},$$

where D denotes the number of spatial dimensions. If we now consider floating-point variables for D2Q9 LB case (we do not consider constant factors in front of both expressions, e.g. double buffering, AA or SSS buffers):

$$N_{LB}^D Q,$$

and for D2Q9 LBMTau1 case

$$N_{LB1}^D (D+1).$$

Memory ratio is equal to

$$\frac{N_{LB1}^D (D+1)}{N_{LB}^D Q} = \frac{D+1}{Q(6\nu_{LB})^{D/4}}.$$

For the D2Q9 model it is, thus, beneficial to have a larger grid with $\tau = 1$ as long as the second grid has a viscosity

$$\frac{1}{54} < \nu_{LB}, (\tau > 0.55).$$

Similarly, for D3Q19 case, it is beneficial to have larger grids and $\tau = 1$ as long as we compare with the standard algorithm at viscosity

$$\frac{1}{6}\left(\frac{4}{19}\right)^{4/3} (\approx 0.021) < \nu_{LB}, (\tau > 0.563).$$

Those limits show that LBMTau1 is particularly efficient at low Reynolds number flows, where high spatial resolution is required, i.e. in porous media. The larger grid likely will require a larger number of time steps (approximately square root of the spatial divisions ratio). This drawback could be partially compensated by faster iterations in LBMTau1. This, however is implementation, and model, dependent and rather hard to estimate theoretically.

5. Discussion

In this paper, a memory-saving algorithm for a simplified (fixed viscosity) LBM method is formulated and tested for flows with the relaxation time $\tau = 1$. This results in an immediate relaxation of the local distribution function [29] and put some limitations on the range of parameters that may be used in the model. The Reynolds number is defined as

$$Re = u_0 L/\mu, \tag{14}$$

where the viscosity $\mu = 1/6$ (see Eq. 2). By changing u_0 or L we control the Reynolds number which is now limited by the resolution of the grid only. To understand the limit we may estimate the Courant-Friedrichs-Lewy (CFL) condition. For velocity measured in the lattice units per time step we require, that u_0 (velocity of the top lid) fulfill $u_0 << 1$. Taking small u_0 stabilizes simulation, but at the same time slows down computation and require more memory as larger grids are required (u_0 and L are the only parameters that may be changed in Eq. 14). If we write the CFL condition as

$$u_0 = (Re/L) \cdot \mu << 1, \tag{15}$$

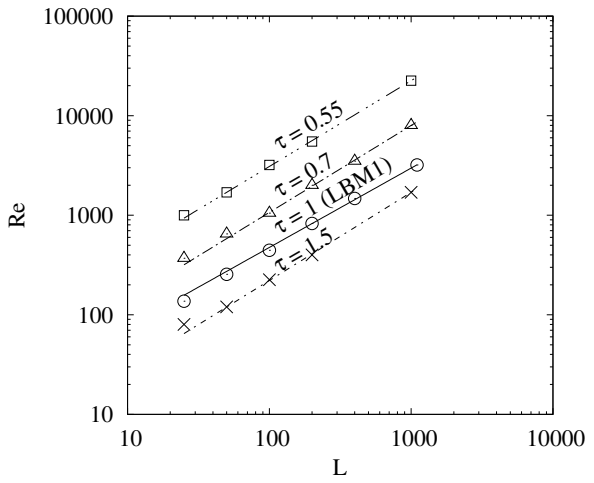

Figure 9: The maximum Reynolds number achieved at grid size L at various τ. The data were obtained empirically by running several simulations for a given grid size. The least square fits (solid and dashed lines) to functions of the type $Re(L) = aL^b$ gave: a=3.7, b=0.89 for $\tau = 1.5$, a=12, b=0.8 for $\tau = 1$, a=19, b=0.87 for $\tau = 0.7$ and a=43, b=0.93 for $\tau = 0.55$.

and because $\mu = 1/6$, to fulfill CFL criteria we should keep $Re/L << 6$. To check and validate this condition we run a series of simulations for the lid-driven cavity at L from 50 to 1000 for increasing Reynolds number. If the simulation becomes unstable (and the solver crashed) after at least 1000 time steps, then the previous Re is taken as the maximum possible for the given lattice size. We repeated the procedure for various τ and collect the data in Fig. 9. The data for the smallest viscosity and $L < 100$ agrees with [37]. We notice that the lower relaxation time τ is, the higher Reynolds number may be achieved. However, for the relaxation time, $\tau = 0.55$ the low resolution of the lattice leads to an inaccuracy in the solutions, especially in the regions where small vortices appear and in the center of the main vortex (data not shown). We found that if we keep $\tau = 1$ then all converged solutions are of acceptable accuracy (see e.g. Fig. 5). This finding agrees with the conclusions based on the linear stability theory where $\tau = 1$ was suggested too [22]. In practice, one could estimate the maximum Reynolds number using the grid size L directly from the plot in Fig. 9 or an empirical function fit to $Re(L) = aL^b$ given in the figure caption. The results for Re=3200 (see Fig. 5) confirmed the stability of the solver for larger Reynolds numbers and large grids. There is only one outlier point for u velocity component at $x \approx 0.46$ that has probably been a typo in the original tables provided in [31].

The main advantage of a new formulation is its relatively low memory consumption. For example, if we use the AB lattice access pattern in standard LBM (where an additional copy of the main lattice is kept in memory) the memory consumption is estimated from [38]

$$M_{\mathrm{LBM}} = (2Q + N_f)c, \tag{16}$$

where Q represents lattice velocity directions (i.e. Q=9 in the standard D2Q9 model), N_f is the number of macroscopic fields (density, velocity, etc.) and c are bytes per single number ($c = 4$ for float, $c = 8$ for double-precision data). We may write that $N_f = D + 1$, where D is the dimension of the model (D components of velocity plus density). Thus, in our case, if we eliminate the distribution function in the LBMTau1 algorithm, it will need only

$$M_{\mathrm{LBM_1}} = 2N_f c \tag{17}$$

bytes of the memory (the factor 2 appears because we store two copies of the macroscopic fields - one from the current and one from the previous time step). In D2Q9 model $N_f = 3$ and $Q = 9$. Thus, using equations (16) and (17) we have $M_{\mathrm{LBM}} = 21c$ and $M_{\mathrm{LBM_1}} = 6c$ respectively. This means the LBMTau1 algorithm needs $\Delta M \approx 76\%$ less memory than the original implementation. A similar calculation for the three-dimensional D3Q27 model gives $\Delta M \approx 86\%$. In practice, for the 2D $1000x1000$ lid-driven cavity flow in Fig. 4 we need $m = 1000 \cdot 1000 \cdot 84 = 84$ MB (megabytes) of memory in the standard LBM to store all simulation data. In LBMTau1, however, for the same grid size, we used only $m = 1000 \cdot 1000 \cdot 24 = 24$ MB. One should keep in mind, however, that in the basic LBMTau1 implementation this memory drop is true for low Reynolds number flows only (see Fig. 9) as higher Reynolds number may be achieved at smaller grids in the standard LBM. This problem, however, may be solved using i.e. fractional step approach for viscosity boost [23], which we leave for future research.

The actual memory gain measured from solver statistics for SCMP model (see Fig. 8) is lower than in theoretical discussions (theoretical $\approx 44\%$, averaged $\approx 40\%$). This is due to the solver internal buffering designed for multi GPU communication. Additionally, to speed up computations LBMTau1 variant of SCMP uses one additional global variable for inter-particle potential which sets a higher theoretical limit on memory gain. The theoretical limit could be lowered to 30% at expense of additional computations and speed loss.

Apart from reduction of memory consumption, LBMTau1 could be optimized in terms of floating-point operations per lattice update as well. In the case of Shan-Chen type models, the proposed approach could outperform a classical model for the same parameter set. Such properties render such rewritten LBM a good candidate for low Re number flows with boundaries that could benefit from high grid resolution - for example, porous media flows.

We suggest that the LBMTau1 algorithm may provide a good starting point for fast and memory-efficient implementations of a solver in parallel environments, including graphics processors (GPUs), as the number of memory accesses decreases with decreasing memory demand of the main algorithm. However, to provide complete parallel implementation, one would need to consider memory access patterns used to compute macroscopic fields, which may not be the most efficient in the basic LBMTau1 implementation. To improve the parallel efficiency, we suggest using one of the improved memory layout algorithms and data exchange algorithms used for the standard Lattice Boltzmann implementations. That includes an AA pattern in which one, instead of two buffers is used (thus, the two-factor

reduction is achieved) and leads to 20% performance gain compared to standard layout [39]. Recently, the structure of arrays shifts and swap (SSS pattern) method based on the AA memory layout was also introduced [40]. It comprises of an additional, separated array of directions for density function on the grid and has confirmed improved parallel efficiency as data access pattern is conserved between odd and even time steps [40]. The approach studied in this paper may be directly compared to the swap algorithm [41], where speedup is less than 1.5 with a memory drop around 2 times if compared to the standard two-lattice approach. The main advantage of using LBMTau1 if compared to these schemes is the memory reduction achieved by removing the distribution function.

6. Conclusions

The presented LBMTau1 version of the LBM algorithm outperforms the original algorithm in terms of memory usage and is useful in large scale, low Reynolds number flows. This is important especially in systems where memory storage matters. This includes multiscale media e.g. porous and artery systems, where the flow at the microscale correlates with macroscopic properties of the medium.

Finally, it is rather surprising, how simple it is to implement a basic version of the LBMTau1 solver. The main function consists of a few lines of a simple C code (see Appendix A). The ratio of the work needed to achieve useful results is relatively low, especially compared to any standard CFD solver. Thus, we believe, the solution provided in this paper may be also attractive in computational physics education. From a practical point of view, the LBMTau1 algorithm discussed here should be useful in applications where the original BGK Lattice Boltzmann was combined with the relaxation time $\tau = 1$. For example, in [12, 14, 15, 16, 18, 20, 21, 22, 23, 42, 43, 44, 45, 46] it is possible to save more than 75% of the memory by using LBMTau1 described here.

7. Acknowledgments

We would like to thank Jonas Latt from the University of Geneva for discussions. Also, many remarks and first reading comments from Remigiusz Durka and Zbigniew Koza from the University of Wrocław were very handful.

Appendix A. The LBMTau1 C code for lid-driven cavity

```
float U[2][L][L], V[2][L][L], R[2][L][L];
int F[L][L];
const int ex[9]={0,1,0,−1,0,1,−1,−1,1};
const int ey[9]={0,0,1,0,−1,1,1,−1,−1};
const int inv[9]={0,3,4,1,2,7,8,5,6};
const float w[9]={4/9.,1/9.,1/9.,1/9.,1/9.,1/36.,1/36.,1/36.,
1/36.};
float U0=0.5;

void init()
{
 for(int i=0; i<L ; i++)
 for(int j=0; j<L ; j++)
 {
  U[0][i][j]=V[0][i][j]=0;
  U[1][i][j]=V[1][i][j]=0;
  R[0][i][j]=R[1][i][j]=1;
  F[i][j]=0;

  if(j==0 or i==0 or i==L−1) F[i][j] = 1;
  if(j==L−1) U[0][i][j] = U[1][i][j] = U0;
 }
}

void LBMTau1(int c)
{
 float r,u,v,f;

 for(int i=0; i<L; i++)
 for(int j=0; j<L−1; j++)
 if(F[i][j]==0)
 {
  U[c][i][j]=V[c][i][j]=R[c][i][j]=0;

  for(int k=0; k<9; k++)
  {
   int ip=i+ex[k], jp=j+ey[k], ik=inv[k];

   if(F[ip][jp]==0)
   {
    r=R[1−c][ip][jp];
    u=U[1−c][ip][jp]/r;
    v=V[1−c][ip][jp]/r;

    f=w[ik]∗r∗(1−(3/2.)∗(u∗u+v∗v)+3.∗(ex[ik]∗u+ey[ik]∗v)
    +(9/2.)∗(ex[ik]∗u+ey[ik]∗v)∗(ex[ik]∗u+ey[ik]∗v));
   }
   else
     f=w[ik]∗R[1−c][i][j];

   R[c][i][j] += f;
   U[c][i][j] += ex[ik]∗f;
   V[c][i][j] += ey[ik]∗f;
  }
 }
}
```

References

[1] F. J. Zajaczkowski, S. E. Haupt, K. J. Schmehl, A preliminary study of assimilating numerical weather prediction data into computational fluid dynamics models for wind prediction, Journal of Wind Engineering and Industrial Aerodynamics 99 (2011) 320–329.

[2] Y. Toparlar, B. Blocken, B. Maiheu, G. Van Heijst, A review on the cfd analysis of urban microclimate, Renewable and Sustainable Energy Reviews 80 (2017) 1613–1640.

[3] B. Xia, D.-W. Sun, Applications of computational fluid dynamics (cfd) in the food industry: a review, Computers and electronics in agriculture 34 (2002) 5–24.

[4] J. Peiró, S. Sherwin, Finite Difference, Finite Element and Finite Volume Methods for Partial Differential Equations, Springer Netherlands, Dordrecht, 2005, pp. 2415–2446.

[5] S. Succi, S. Succi, The Lattice Boltzmann Equation: For Complex States of Flowing Matter, Oxford University Press, 2018.

[6] T. Tomczak, R. G. Szafran, A new gpu implementation for lattice-boltzmann simulations on sparse geometries, Computer Physics Communications 235 (2019) 258–278.

[7] M. Januszewski, M. Kostur, Sailfish: A flexible multi-gpu implementation of the lattice boltzmann method, Computer Physics Communications 185 (2014) 2350–2368.

[8] R. Argentini, A. Bakker, C. Lowe, Efficiently using memory in lattice boltzmann simulations, Future Generation Computer Systems 20 (2004) 973–980.

[9] M. Sheida, M. Taeibi-Rahni, V. Esfahanian, A new approach to reduce memory consumption in lattice boltzmann method on gpu, Journal of Applied Fluid Mechanics 10 (2017) 55–67.

[10] T. Tomczak, R. G. Szafran, Sparse geometries handling in lattice boltzmann method implementation for graphic processors, IEEE Transactions on Parallel and Distributed Systems 29 (2018) 1865–1878.

[11] P. Valero-Lara, Reducing memory requirements for large size lbm simulations on gpus, Concurrency and Computation: Practice and Experience 29 (2017) e4221.

[12] Y. Wei, Lattice boltzmann simulations for thermal vapor-liquid two-phase flows, The Journal of Computational Multiphase Flows 4 (2012) 103–109.

[13] P. Rao, L. Schaefer, Lattice boltzmann models for micro-tomographic pore-spaces, Computers & Fluids 193 (2019) 104294.

[14] S. Khajepor, J. Cui, M. Dewar, B. Chen, A study of wall boundary conditions in pseudopotential lattice boltzmann models, Computers & Fluids 193 (2019) 103896.

[15] Y. Chen, K. Zhu, A study of the upper limit of solid scatters density for gray lattice boltzmann method, Acta Mechanica Sinica 24 (2008) 515–522.

[16] M. Szucki, J. Suchy, J. Lelito, P. Malinowski, J. Sobczyk, Application of the lattice boltzmann method for simulation of the mold filling process in the casting industry, Heat and Mass Transfer 53 (2017) 3421–3431.

[17] O. Shardt, Comparison of finite volume and lattice boltzmann methods for multicomponent flow simulations, The Canadian Journal of Chemical Engineering 98 (2020) 44–53.

[18] L. Mountrakis, E. Lorenz, A. Hoekstra, Revisiting the use of the immersed-boundary lattice-boltzmann method for simulations of suspended particles, Physical Review E 96 (2017) 013302.

[19] C. Pan, L.-S. Luo, C. T. Miller, An evaluation of lattice boltzmann schemes for porous medium flow simulation, Computers & fluids 35 (2006) 898–909.

[20] D. Lycett-Brown, K. H. Luo, Multiphase cascaded lattice boltzmann method, Computers & Mathematics with Applications 67 (2014) 350–362.

[21] F. Hai-Ping, W. Rong-Zheng, F. Le-Wen, Lattice boltzmann method simulation on the flow of two immiscible fluids in complex geometry, Chinese Physics 9 (2000) 515.

[22] R. A. Worthing, J. Mozer, G. Seeley, Stability of lattice boltzmann methods in hydrodynamic regimes, Physical Review E 56 (1997) 2243.

[23] C. Shu, X. Niu, Y.-T. Chew, Q. Cai, A fractional step lattice boltzmann method for simulating high reynolds number flows, Mathematics and Computers in Simulation 72 (2006) 201–205.

[24] Z. Chen, C. Shu, Y. Wang, L. Yang, D. Tan, A simplified lattice boltzmann method without evolution of distribution function, Advances in Applied Mathematics and Mechanics 9 (2017) 1–22.

[25] J. G. Zhou, Macroscopic lattice boltzmann method (maclab), CoRR abs/1901.02716 (2019). URL: `http://arxiv.org/abs/1901.02716`. `arXiv:1901.02716`.

[26] Z. Guo, C. Shu, Lattice Boltzmann Method and Its' Applications in Engineering, Advances in Computational Fluid Dynamics, World Scientific Publishing Company Incorporated, 2013.

[27] J. Buick, J. Cosgrove, Investigation of a lattice boltzmann model with a variable speed of sound, Journal of Physics A: Mathematical and General 39 (2006) 13807.

[28] M. C. Sukop, D. T. J. Thorne, Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers, 1 ed., Springer Publishing Company, Incorporated, 2007.

[29] a. K. G. E. Shiyi Chen, Gary D. Doolen, Lattice-boltzmanna fluid dynamics versatile tool for multiphase and other complicated flows, Los Alamos Science (1994) 98–111.

[30] O. Botella, R. Peyret, Benchmark spectral results on the lid-driven cavity flow, Computers & Fluids 27 (1998) 421–433.

[31] U. Ghia, K. N. Ghia, C. Shin, High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method, Journal of computational physics 48 (1982) 387–411.

[32] X. Shan, H. Chen, Lattice boltzmann model for simulating flows with multiple phases and components, Phys. Rev. E 47 (1993) 1815–1819. URL: `https://link.aps.org/doi/10.1103/PhysRevE.47.1815`. doi:10.1103/PhysRevE.47.1815.

[33] Ł. Łaniewski-Wołłk, M. Dzikowski, D. Sashko, T. Mitchell, G. Gruszczyński, PabloOb, R. M., R. W., G. T., bhill23, M. J., de Waard C., franjesus, CFD-GO/TCLB: Version 6.5, 2020. URL: `https://github.com/CFD-GO/TCLB`. doi:10.5281/zenodo.4074541.

[34] A. Kupershtokh, D. Medvedev, D. Karpov, On equations of state in a lattice Boltzmann method, Computers & Mathematics with Applications 58 (2009) 965–974. doi:10.1016/j.camwa.2009.02.024.

[35] Ł. Łaniewski-Wołłk, J. Rokicki, Adjoint lattice boltzmann for topology optimization on multi-gpu architecture, Computers & Mathematics with Applications 71 (2016) 833 – 848. doi:https://doi.org/10.1016/j.camwa.2015.12.043.

[36] M. Dzikowski, Ł. Łaniewski-Wołłk, J. Rokicki, Single component multiphase lattice boltzmann method for taylor/bretherton bubble train flow simulations, Communications in Computational Physics 19 (2016) 1042–1066. doi:10.4208/cicp.220115.110915a.

[37] A. Montessori, G. Falcucci, P. Prestininzi, M. La Rocca, S. Succi, Regularized lattice bhatnagar-gross-krook model for two-and three-dimensional cavity flow simulations, Physical Review E 89 (2014) 053317.

[38] Sailfish, Sailfish manual, `http://sailfish.us.edu.pl/simulation`, 2019. Accessed: 2019-10-04.

[39] P. Bailey, J. Myre, S. D. Walsh, D. J. Lilja, M. O. Saar, Accelerating lattice boltzmann fluid flow simulations using graphics processors, in: 2009 international conference on parallel processing, IEEE, 2009, pp. 550–557.

[40] M. Mohrhard, G. Thäter, J. Bludau, B. Horvat, M. J. Krause, Auto-vectorization friendly parallel lattice boltzmann streaming scheme for direct addressing, Computers & Fluids 181 (2019) 1–7.

[41] K. Mattila, J. Hyväluoma, T. Rossi, M. Aspnäs, J. Westerholm, An efficient swap algorithm for the lattice boltzmann method, Computer Physics Communications 176 (2007) 200–210.

[42] R. Blaak, P. M. Sloot, Lattice dependence of reaction-diffusion in lattice boltzmann modeling, Computer Physics Communications 129 (2000) 256–266.

[43] I. Halliday, S. Lishchuk, T. Spencer, G. Pontrelli, C. Care, Multiple-component lattice boltzmann equation for fluid-filled vesicles in flow, Physical Review E 87 (2013) 023307.

[44] M. Matyka, A. Khalili, Z. Koza, Tortuosity-porosity relation in porous media flow, Phys. Rev. E 78 (2008) 026306. doi:10.1103/PhysRevE.78.026306.

[45] M. Mendoza, H. J. Herrmann, S. Succi, Lattice boltzmann model for electronic structure simulations, in: Journal of Physics: Conference Series, volume 640(1), IOP Publishing, 2015, p. 012018.

[46] O. Shardt, Comparison of finite volume and lattice boltzmann methods for multicomponent flow simulations, The Canadian Journal of Chemical Engineering 98 (2020) 44–53.