High Performance Computing with Python

Final Report

Tanya Dora

5651996

td167@students.uni-freiburg.de

August 27, 2023

Contents

1

Overview

In modern scientific research and engineering, fluid dynamics simulations and high-performance computing are essential instruments. Simulations are a desirable alternative to large-scale physics experiments because they are often expensive and unfeasible. One example of simulation's usefulness is in chemical systems, where complex reactions and interactions can be accurately modeled to gain valuable insights.[1]
Engineers may thoroughly understand aerodynamic drag and optimize car designs without the need for several physical prototypes by precisely modeling fluid flow around the car body forms.
In fluid flow simulations, a scheme is required to approximate the physical states of the fluid at each time step. The lattice Boltzmann method (LBM) is a well-known and powerful scheme used for fluid flow simulations.[2]
The lattice Boltzmann method (LBM) is a mesoscopic simulation technique that efficiently models complex fluid flow behaviors by simulating discrete particles on a lattice grid. It avoids direct solutions of the Navier-Stokes equations, instead using iterative simulations of particle collisions and propagation to compute physical states. The method's accuracy is bolstered by the use of the Maxwell velocity distribution function. LBM's versatility and efficiency make it a valuable tool in diverse research and engineering applications.[3] The advantages of LBM are as follows:
Simplified Execution Equations for each moment are straightforward, leading to a simple implementation of the LBM algorithm.
Collision Handling LBM limits collision handling to adjacent lattices, enhancing efficiency and reducing computational intensity.
Excellent Parallel Scaling LBM exhibits strong parallel scaling when implemented in Python efficiently utilizing parallel computational resources.
Local Dynamics Nature LBM's local dynamics contribute to its robust parallelization, enhancing scalability with computational resources.[4]

2

Lattice Boltzmann Fundamentals

In this Chapter, It is explained how the equations utilized in the LBM are generated. More particularly, we discuss the Boltzmann transport equation (BTE) which is one of the fundamental equations of the kinetic theory of fluids, as well as how to deal with boundary conditions.

2.1 The Boltzmann transport equation (BTE)

Ludwig Boltzmann established the Boltzmann transport equation (BTE) as a statistical model for molecular movement in flow within the kinetic gas theory framework. The Lattice Boltzmann method (LBM), introduced by McNamara and Zanetti in 1988[5], is a numerical approach based on a discretized form of the BTE. LBM models have found extensive use in fluid dynamics research over the past three decades, spanning multiphase flow, porous media, and microfluidics applications. A comprehensive evaluation of LBM's applications in fluid dynamics and beyond is provided by [6].
As postulated by Bhatnagar, Gross, and Krook (BGK) in 1954, one regular approach is to assume relaxation of f towards f_{eq} with a single characteristic time τ. The Boltzmann transport equation (BTE) which formulates the time evolution of the particle probability density function (PDF) $f(x, v, t)$ given the microscopic velocity v and position x of particles is as follows:

$$\frac{df(\mathbf{x}, \mathbf{v}, t)}{dt} = -\frac{f(x, v, t) - f_{\text{eq}}(v; \rho(x, t), u(x, t), T(x, t))}{\tau} \tag{2.1}$$

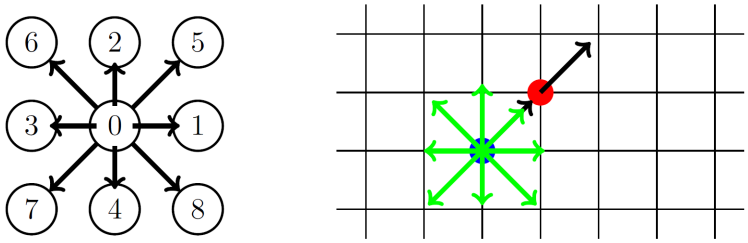
where f_{eq} represents the statistical equilibrium, $T(x, t)$ represents the temperature at x of time step t, τ represents a characteristic time, $\rho(x, t)$ represents the macroscopic density, and $u(x, t)$ represents the macroscopic velocity.
The characteristic time governs how fast.the fluid approaches equilibrium.The

higher the τ, the more slowly the convergence to the equilibrium.Eq. (2.1) Furthermore, $f(x, v, t)$ is utilized to compute fluid physical properties including density and velocity. The instant updates are carried out using [7]:

$$\rho(x,t) = \int f(x,v,t)dv \text{ and } u(x,t) = \frac{1}{\rho(x,t)} \int vf(x,v,t)dv. \tag{2.2}$$

2.2 Discretization of Boltzmann Equation

Due to the problem's complexity and intrinsic multidimensionality, solving the Boltzmann Equation analytically is challenging. Thus, numerical methods, particularly discretization techniques, have become essential[8].In this paper, our emphasis is on discretization within a two-dimensional space.



(a) (b)

Figure 2.1: Discretization of the BTE. (a) Discretization on the velocity space according to D2Q9. (b) Uniform 2D grid for the discretization in the physical space.

These methods discretize the continuous distribution function onto a grid, allowing numerical solutions through computer resources. This paper offers an overview of these procedures for solving the Boltzmann equation and their varied applications. The review showcases advancements in computational approaches to studying transport phenomena and foresees future progress in this evolving field by tracing the evolution of Boltzmann equation discretization techniques.[9]
The discretization for space and time is carried out in such a way that the equality requirement of the following inequality:

$$c_i \Delta t \leq \|\Delta x_i\|$$

where Δt is the time step size and Δx_i is the distance between the closest grid in the direction of c_i that is defined by:

$$c = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}^\top. \tag{2.3}$$

Given the condition that $\Delta x = \Delta y = \Delta t = 1$, the discretization is tailored to meet this equality requirement. Notably, this approach is used within two-dimensional space and encompasses nine discrete directions., as visually depicted in Fig. 2.1
This discretization scheme, often referred to as the D2Q9 lattice Boltzmann method, involves discretizing the particle probability density function across nine directions. This is accomplished by using subscript notation to represent the particle probability density function as $f(x,t)$.[9, 10]

$$\rho(\mathbf{x}_j, t) = \sum_i f_i(\mathbf{x}_j, t) \tag{2.4}$$

$$\mathbf{u}(\mathbf{x}_j, t) = \frac{1}{\rho(\mathbf{x}_j, t)} \sum_i \mathbf{c}_i f_i(\mathbf{x}_j, t) \tag{2.5}$$

The major difference between f_I and the continuous distribution function f is that all of the argument variables of f_I are discrete. In Eq. 2.4, the density is treated as a unit molecular mass. Furthermore, the equilibrium in 2.1 is calculated as follows:

$$\underbrace{f_i(\mathbf{x}_j + \mathbf{c}_i \cdot \Delta t, t + \Delta t) - f_i \mathbf{x}_j, t}_{\text{streaming}} = \underbrace{-\omega \left(f_i(\mathbf{x}_j, t) - f_i^{\text{eq}}(\mathbf{x}_j, t) \right)}_{\text{collision}} \tag{2.6}$$

where $\omega = t/\tau$ is the relaxation parameter. The equilibrium is computed as [11]:

$$f_i^{\text{eq}}(x,t) = w_i \rho(x,t) \left[1 + 3c_i \cdot u(x,t) + \frac{9}{2}(c_i \cdot u(x,t))^2 - \frac{3}{2}\|u(x,t)\|^2 \right] \tag{2.7}$$

where the index i corresponds to Figure 2.1 and $w = [\frac{4}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}]$. The grid receives the particle flow $f_i(x + c_i dt, cdot)$ from its nine neighboring grids during the streaming stage. We relax the probability density function towards the equilibrium f_i^{eq} in the collision phase by taking into account the impacts of the particle collision.

3

Computation and Design Methodology

3.1 Boundary and Initial Considerations

In the next section, we'll explore how we handle particle collisions with boundaries. Symmetry requirements are addressed through periodic boundary conditions, creating a cyclic flow where fluid leaving one boundary reenters the domain from the opposite side. This condition is often implicitly applied during the streaming process, with the population $f_i(x+L,t)$ leaving one border automatically rewritten on the opposite border $f_i(x,t)$.

3.1.1 Dry and Wet Nodes

LB boundary schemes fall into two main categories: *Dry Node* and *Wet Node*. This paper exclusively covers the dry-node group, chosen for its ease of creation and ability to maintain second-order precision when the physical wall aligns precisely midway between nodes.[12]

- **Dry Nodes/Link-Wise** These lattice points correspond to areas where fluid presence is minimal or absent, commonly indicating solid boundaries or void regions within the computational lattice.
- **Wet Nodes** denote lattice sites containing fluid particles, embodying the dynamic behavior of the fluid system.

3.1.2 The Time Step: Streaming

In the lattice Boltzmann method (LBM), the streaming phase is a crucial step where particle distributions move from their current lattice node to neighboring nodes through predetermined discrete velocity vectors. This mimics fluid particle advection and transfers information about particle distributions across the lattice. During the streaming step, each lattice node

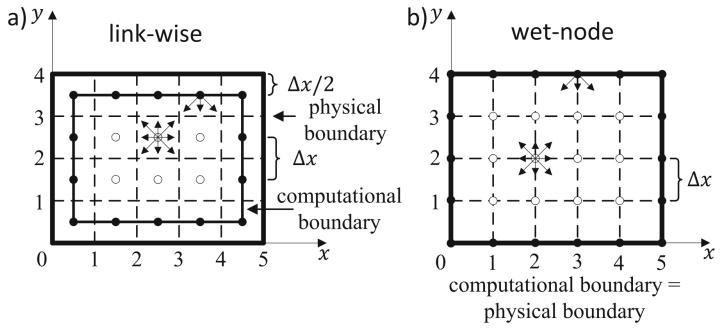


Figure 3.1: Two discretizations of the same domain with (a) dry-node/link-wise and (b) wet-node boundary conditions.[11]

updates its particle distribution values by shifting them along the discrete velocity vectors. For a D2Q9 lattice (nine discrete velocities), the streaming step can be mathematically expressed as:

$$\underbrace{f_i(\mathbf{x}_j + \mathbf{c}_i \cdot \Delta t, t + \Delta t) - f_i \mathbf{x}_j, t}_{\text{streaming}} \tag{3.1}$$

Here, f_i is the particle distribution function for the $i_t h$ velocity, x is the lattice node position, c_i is the velocity vector, and Δt is the time step. The streaming process moves fluid particles through the lattice over discrete time intervals, simulating fluid flow and transport.

3.1.3 The Time Step: Collision

To incorporate the collision operator in the lattice Boltzmann equation (LBE), a simplified BGKT (Bhatnagar-Gross-Krook-Twersky) model is used, assuming local relaxation of the distribution function towards equilibrium[13, 14]. With this approximation, the Boltzmann transport equation (BTE) becomes:

$$\underbrace{-\omega \left(f_i(\mathbf{x}_j, t) - f_i^{\text{eq}}(\mathbf{x}_j, t)\right)}_{\text{collision}} \tag{3.2}$$

where $\omega = t/\tau$ is the relaxation parameter. The equilibrium is computed as [11]:

$$f_i^{\text{eq}}(x,t) = w_i \rho(x,t) \left[1 + 3c_i \cdot u(x,t) + \frac{9}{2}(c_i \cdot u(x,t))^2 - \frac{3}{2}\|u(x,t)\|^2\right] \tag{3.3}$$

where the index i corresponds to Figure 2.1 and $w = [\frac{4}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}]$. These equations capture how the BGKT approximation influences the lattice Boltzmann method, facilitating distribution function relaxation toward equilibrium[13, 14].

3.2 Periodic Boundary Conditions

Periodic boundary conditions are essential in LBM simulations to mimic infinite domains and minimize edge effects[15]. Three common types are used: rigid walls, moving walls, and pressure gradient walls, each simulating different scenarios.

3.2.1 Rigid Wall

The hard wall boundary condition simulates impenetrable solid walls, reflecting incoming particles as outgoing ones on the opposite side of the domain. To achieve this, distribution functions are exchanged between border nodes. Fluid particles "bounce back" from the wall, adhering to a no-slip condition at the boundary. The equation at the boundary is computed as [16]:

$$f_i(x_b, t + \Delta t) = f_{i^\star}(x_b, t). \tag{3.4}$$

3.2.2 Moving Wall

The moving wall boundary condition is used for moving boundaries or objects within the fluid. Incoming distribution functions are propagated to the outgoing side, like the rigid wall condition, but with an additional shift to accommodate boundary movement. This shift ensures accurate interaction between fluid particles and the moving boundary. When the boundary moves at velocity U_w, adjustments are made to account for changes in particle moments, modifying the equation as follows: [16]:

$$f_i(x_b, t + \Delta t) = f_{i^\star}(x_b, t) - 2w_i\rho_w \frac{c_i \cdot U_w}{c_s^2} \tag{3.5}$$

where $c_\mathrm{s} = \frac{1}{\sqrt{3}}$ is the speed of sound and ρ_w is the density at the wall. It is important to mention that we utilize the value $U_w = [U_\mathrm{w}, 0]^\top$ throughout this study. The quantity ρ_w is typically determined through one of the following approaches [17, 18]:

1. Calculate the average density $\bar{\rho}$ of the simulated region.
2. Estimate ρ_w through extrapolation using the particle probability density function within the actual physical domain, as described by Equation 19 in reference [17].

3.2.3 Pressure Gradient Wall

Pressure gradient wall simulates pressure-driven fluid flow across domains, useful for channels or pipes. Inlet distribution functions change for the pressure gradient, propelling particles.As they move, distribution functions

adapt to pressure-driven motion.
We consider a scenario where boundaries are positioned at $x = 0$ (inlet) and $(X-1)\Delta x$ (outlet), where X represents the lattice grid count along the x-axis. The fundamental periodic boundary condition (PBC) assumes a flow continuity from outlet to inlet, effectively expressing that $f(0, y, t) = f((X-1)\Delta x, y, t)$ [16]. Implicitly, this condition is incorporated during the streaming operation .
An alternative PBC accounts for pressure variation, denoted as Δp, between the inlet and outlet. Given that density ρ is determined as $\rho = \frac{p}{c_s^2}$, where p signifies pressure, densities at the inlet and outlet, $\rho_{\text{in}} = \frac{p_{\text{out}}+\Delta p}{c_s^2}$ and $\rho_o ut = \frac{p_{\text{out}}}{c_s^2}$, can be evaluated using a constant outlet pressure p_{out} and the pressure variation Δp. Consequently, pre-streaming distributions $f^\star$ at the inlet and outlet are determined as follows [16]:

$$f_i^\star(-\Delta x, y, t) = f_i^{\text{eq}}(\rho_{in}, u((X-1)\Delta x, y, t)) + (f_i^\star((X-1)\Delta x, y, t) - f_i^{\text{eq}}((X-1)\Delta x, y, t)),$$
$$f_i^\star(X\Delta x, y, t) = f_i^{\text{eq}}(\rho_{\text{out}}, u(0, y, t)) + (f_i^\star(0, y, t) - f_i^{\text{eq}}(0, y, t)) \tag{3.6}$$

Here, $x = -\Delta x$ and $x = X\Delta x$ indicate additional layers of nodes introduced for implementation purposes. These additional layers at $-\Delta x$ (inlet) and $X\Delta x$ (outlet) correspond respectively to $(X-1)\Delta x$ and 0 in the physical domain. Importantly, since the pressure-based PBC computes the pre-streaming distributions $f^\star$, it necessitates execution before the streaming operation, unlike the bounce-back method.
These boundary conditions ensure particle periodicity, allowing seamless domain exit and entry. This permits simulating large domains without explicit full-scale modeling.

3.3 Implementation in Python

Python's popularity for lattice Boltzmann method (LBM) implementations thrives due to its user-friendly syntax and powerful libraries. Its versatility facilitates modeling intricate fluid dynamics phenomena. NumPy and similar libraries optimize data manipulation, boosting LBM computational efficiency. A helpful resource is "Lattice Boltzmann Method with Python"[19], offering practical insights and examples. Implementations assume D2Q9 discretization, where x and y denote horizontal and vertical axes, respectively.`Numpy` [1] and `mpi4py` [2].

[1]Numpy: https://numpy.org/
[2]mpi4py: https://mpi4py.readthedocs.io/en/stable/

The `numpy.roll` [3] function is a valuable tool in the lattice Boltzmann method for handling the streaming step, which involves the movement of particle distributions to adjacent lattice nodes.
Code for Streaming Operation Move particles to neighboring cells

```
def streaming(f):
for i in np.arange(1, 9):
    f[i] = np.roll(f[i],shift=c_ai.T[i],axis=(0,1))
    #Shift particles by the lattice velocity vector
return f
```

Here, f signifies particle distribution functions for various lattice velocities. c_{ai} defines velocity vectors for particle shifting in the streaming step.
Code for Collision Operation:Update particle distribution based on equilibrium

```
def collision(f_inm, omega):
    # Update density and velocity arrays
    rho_ij = compute_density(f_inm)
    u_cij = compute_velocity(f_inm, rho_ij)
    # Calculate the equilibrium distribution function
    f_eqm = equilibrium_distribution(rho_ij, u_cij)
    # Update particle using collision model
    f_new = f_inm + omega * (f_eqm - f_inm)
    return f_new
```

Collision updates f_{inm} with local density and velocity, aiming for equilibrium distribution f_{eqm} via BGK model and relaxation parameter ω.

3.4 Parallel Computation by MPI4

Parallel computation of the lattice Boltzmann method (LBM) in fluid mechanics using `MPI` [4] (Message Passing Interface) is a powerful approach that enables efficient simulation of complex fluid flows on high-performance computing (HPC) systems.
Lattice Boltzmann Method divides the domain into cells, each representing a fluid particle distribution. Parallel LBM distributes these cells across processors in an HPC cluster. MPI, a standard communication library, aids communication and synchronization between processors.[20]
MPI-based parallelization in LBM employs domain decomposition and workload distribution. The computational domain is split into subdomains, assigned to processors. Processors handle streaming and collision operations independently. Boundary information communication ensures result accuracy.[14]

[3] numpy.roll :https://numpy.org/doc/stable/reference/generated/numpy.roll.html
[4] MPI: //www.mpi-forum.org/

4

Statistical Findings and Results

In the preceding section, we delved into the technical intricacies of applying the Lattice Boltzmann Method (LBM) to different parameters. In the forthcoming chapter, we will elucidate the process of assessing these implementations. This evaluation will be conducted prior to presenting the visual representations and numerical findings derived from the series of experiments.

4.1 Shear wave decay

The simulation centers around two core shear wave experiments, each characterized by distinct initial circumstances that enable a detailed investigation into the intricate dynamics of shear wave propagation and fluid behavior.

Case 1: Initial Density Perturbation

Initial Density: $\rho(\mathbf{r}, 0) = 1$
Initial Velocity: $u_x(\mathbf{r}, 0) = \varepsilon \cdot \sin\left(\frac{2\pi y}{L_y}\right)$
(where L_y represents the length of the domain in the y-direction)

Case 2: Initial Velocity Perturbation

Initial Density: $\rho(\mathbf{r}, 0) = \rho_0 + \varepsilon \cdot \sin\left(\frac{2\pi x}{L_x}\right)$
(where L_x represents the length of the domain in the x-direction)
Initial Velocity: $u(\mathbf{r}, 0) = 0$

These two cases provide a comprehensive foundation for studying shear wave propagation and its interaction with fluid mechanics.

4.1.1 Shear Wave Decay - Python Implementation and Results

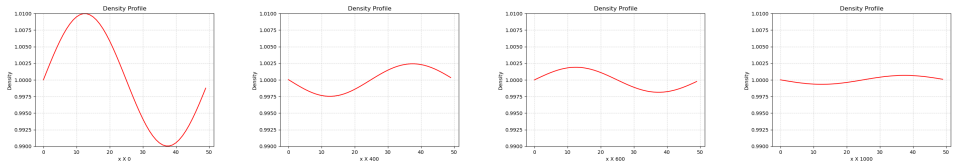
Here's a brief overview of what the script does:

- **Importing Libraries**: The script begins by importing the necessary libraries, including `os` [1], `Numpy`, `Matplotlib`, and functions from the `scipy.signal` [2] and `scipy.optimize` [3] modules.
- **Simulation Functions**:
  - shear wave viscosity: This function simulates shear wave experiments and calculates viscosity for either density or velocity profiles. It takes various parameters such as experiment type, grid dimensions, relaxation parameters, and others.
  - shear wave simulation: This function performs the shear wave simulation and plots the results. It sets up initial conditions, updates the lattice using streaming and collision steps, and generates plots for density and velocity profiles.

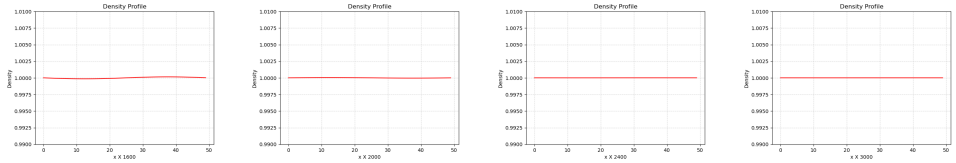
Algorithm

```
import os, scipy, numpy, matplotlib, Helper files
# Define paths and directories
def shear_wave_viscosity():
    # Initialize meshgrid and initial conditions
    # Set initial conditions based on experiment_type
    Loop for num_steps:
        Update distribution, density, and velocity
        Calculate shear wave amplitude and append to
        quantity
    Calculate simulated viscosity using curve fitting
    Return simulated_viscosity, analytical_viscosity
def shear_wave_simulation():
    # Initialize meshgrid and viscosity
    # Set initial conditions based on experiment_type
    Loop for num_steps:
        Update distribution, density, and velocity
        of the lattice using streaming and
        collision steps
    Plot simulated and analytical results
```

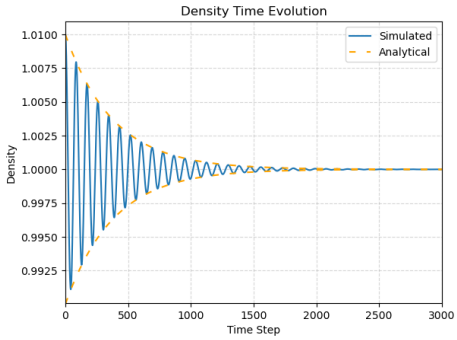
[1] os:https://docs.python.org/3/library/os.html
[2] https://docs.scipy.org/doc/scipy/reference/signal.html
[3] https://docs.scipy.org/doc/scipy/reference/optimize.html



(a) $t = 0$ (b) $t = 400$ (c) $t = 600$ (d) $t = 1000$

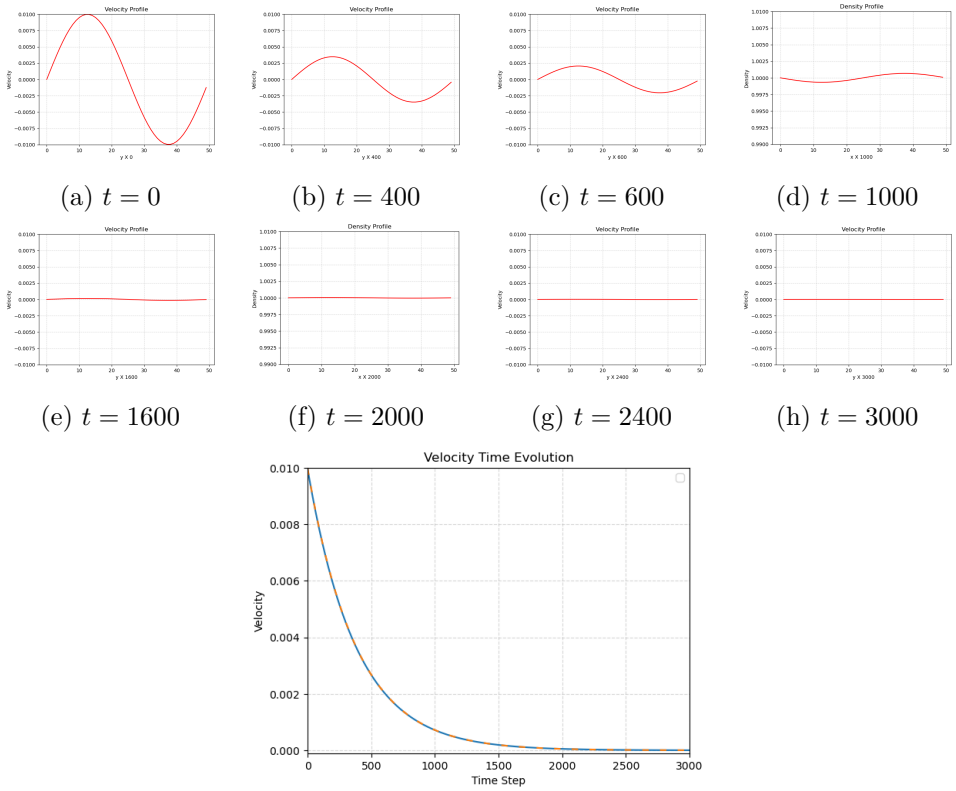


(e) $t = 1600$ (f) $t = 2000$ (g) $t = 2400$ (h) $t = 3000$



(a) Time evolution of density at $x = \arg\max_{x'} \rho(x', t = 0)$

Figure 4.2: The progression over time of the sinusoidal density, within a lattice grid of dimensions $(50, 50)$.

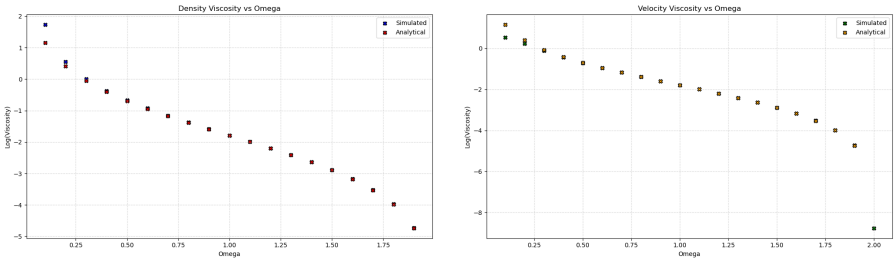


(a) $t = 0$ (b) $t = 400$ (c) $t = 600$ (d) $t = 1000$

(e) $t = 1600$ (f) $t = 2000$ (g) $t = 2400$ (h) $t = 3000$

(i) Time evolution of velocity at $x = \arg\max_{x'} \rho(x', t = 0)$

Figure 4.3: The progression over time of the sinusoidal velocity, is depicted at a specific point, namely $y = 25$, within a lattice grid of dimensions $(50, 50)$.



(a) Sinusoidal density (b) Sinusoidal velocity

Figure 4.4: The simulated viscosity value over various relaxation values ω. The analytical solution uses $\nu = c_s^2(\frac{1}{\omega} - \frac{1}{2})$ and the simulated viscosity ν is approximated from an exponential decay curve.

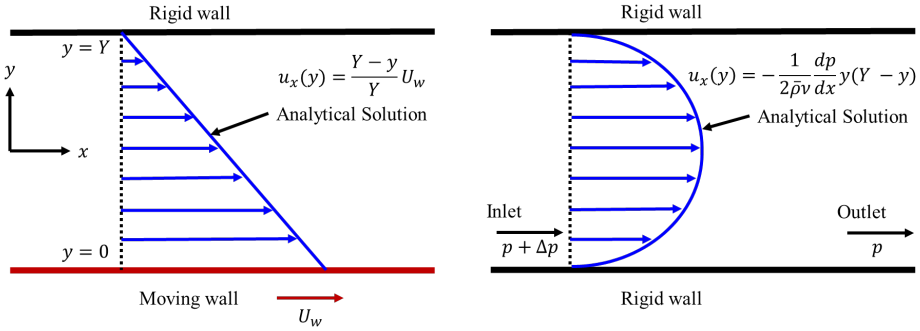


Figure 4.5: The conceptual visualizations of the Couette flow (Left) and Poiseuille flow (Right).[11]

4.2 Couette Flow

Couette flow is a fundamental fluid dynamics scenario that involves the flow of a viscous fluid between two parallel plates, where one plate is stationary (fixed wall) and the other plate is moving with a constant velocity (moving wall). This setup leads to the generation of a shear flow between the two plates, resulting in velocity gradients and fluid motion.

4.2.1 Couette Flow - Python Implementation and Results

Here's a brief overview of what the script does:

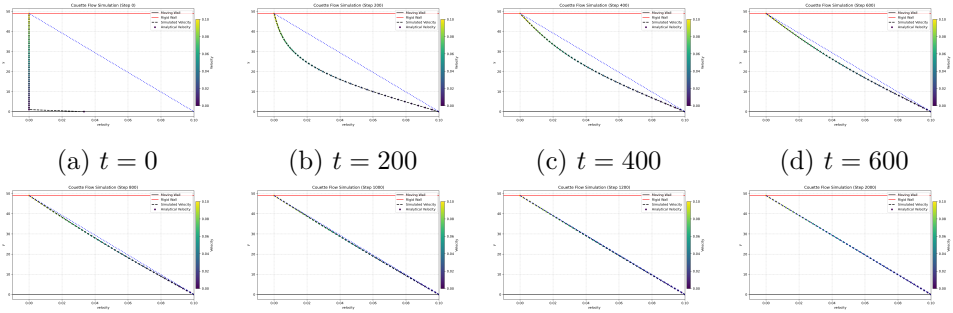
- Import necessary libraries: NumPy, Matplotlib, and custom modules for Lattice Boltzmann Method and Boundary Conditions.
- Boundary Conditions:
  - Implement periodic boundary conditions in the x-direction, allowing fluid particles leaving one side of the domain to re-enter

from the opposite side.
  - Implement bounce-back boundary conditions on the Rigid wall.
  - Apply the appropriate boundary conditions to the moving wall to account for its prescribed velocity.

- **plot couette flow** to visualize Couette flow velocity profiles.
- **couette flow** to simulate Couette flow using Lattice Boltzmann Method.

Algorithm

```
import os, scipy, numpy, matplotlib,Helper files
# Define paths and directories
def couette_flow_simulation():
    # Initialize meshgrid and viscosity
    # Set initial conditions based on experiment_type
    # Set Moving and Rigid wall Boundary
    # Loop for num_steps:
        # Cache boundary conditions
        boundaries_cache(f_inm,f_eq,u_cij)
        # Streaming step
        f_inm = lbm.streaming(f_inm)
        # Apply boundary conditions
        apply_boundaries(f_inm)
        # Collision step
        f_inm = lbm.collision(f_inm, omega)
        # Update density and velocity arrays
        compute_density(f_inm)
        compute_velocity(f_inm, latticeBoltzmann.rho_ij)
        # Plot profiles at regular intervals
        # Append deviations to profile
    # Plot simulated and analytical results
```



(a) $t = 0$ (b) $t = 200$ (c) $t = 400$ (d) $t = 600$

(e) $t = 800$ (f) $t = 1000$ (g) $t = 1200$ (h) $t = 2000$

Figure 4.6: Time evolution of Couette Flow

4.3 Poiseuille Flow

In Poiseuille flow, fluid motion is propelled by a pressure disparity between the pipe's entrance and exit. The velocity profile follows a parabolic pattern, with maximum speed at the pipe's center, tapering linearly towards the walls. This flow adheres to the Navier-Stokes equation, and assuming steady, laminar conditions yields a simplified solution. Mathematically, the velocity profile is expressed as:

$$u(y) = -\frac{1}{2\mu}\frac{dp}{dx}y(h-y)$$

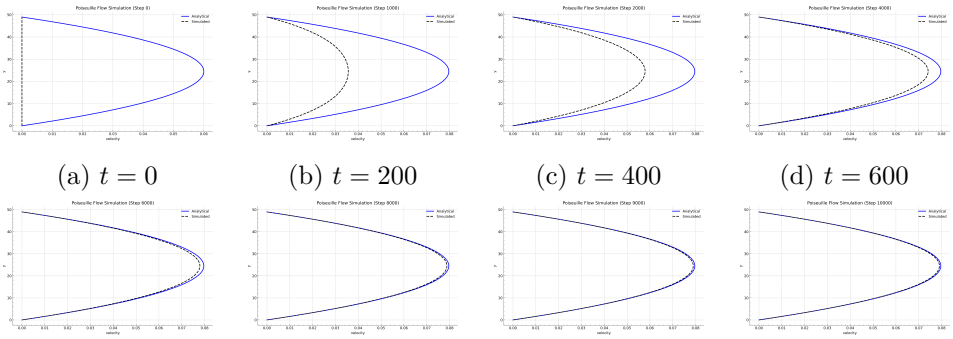
where $u(y)$ is the velocity at a distance y from the pipe axis, μ is the dynamic viscosity, $\frac{dp}{dx}$ is the pressure gradient, and h is the diameter of the pipe.

4.3.1 Poiseuille Flow - Python Implementation and Results

Algorithm

```
import os, scipy, numpy, matplotlib, Helper files
# Define paths and directories
def poiseuille_flow_simulation():
    # Initialize meshgrid and viscosity
    initialize_meshgrid()
    viscosity = calculate_viscosity()
    # Set initial conditions based on experiment_type
    # Set Left, Right Pressure and Rigid wall Boundary
    # Loop for num_steps:
        # Cache boundary conditions
        boundaries_cache(f_inm, f_eq, u_cij)
        # Streaming step
        f_inm = lbm.streaming(f_inm)
```

```
        # Apply boundary conditions
        apply_boundaries(f_inm)
        # Collision step
        f_inm = lbm.collision(f_inm, omega)
        # Update density and velocity arrays
        compute_density(f_inm)
        compute_velocity(f_inm, rho_ij)
        # Plot profiles at regular intervals
        # Append deviations to profile
# Plot simulated and analytical results
plot_results()
```



(a) $t = 0$ (b) $t = 200$ (c) $t = 400$ (d) $t = 600$

(e) $t = 800$ (f) $t = 1000$ (g) $t = 1200$ (h) $t = 2000$

Figure 4.7: Time evolution of Poiseuille Flow

4.4 Sliding Lid-driven cavity

The sliding lid phenomenon offers a captivating glimpse into fluid dynamics, revealing the intricate interplay of motion and vortices within a confined liquid space. As a lid gracefully glides along a box's boundary, it initiates a captivating dance of fluid motion, accompanied by mesmerizing vortices. Central to this phenomenon is the Reynolds number (Re), a key dimensionless parameter representing the balance between inertial and viscous forces in the fluid. Leveraging the Lattice Boltzmann method, we embark on a computational expedition to simulate and unveil the dynamics of this mesmerizing process.

4.4.1 Sliding Lid - Python Implementation and Results

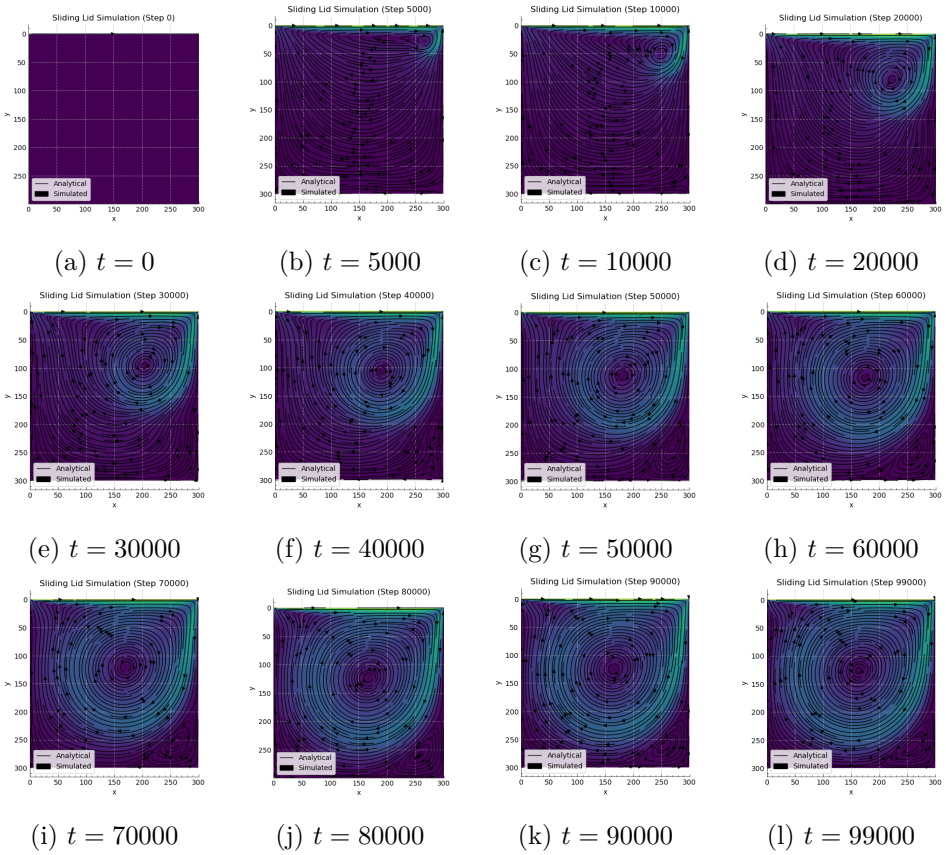
Here's a breakdown of what the script does

- Import Libraries: Import necessary libraries for image processing, calculations, and visualization: os, PIL, numpy, and matplotlib..
- plotting function: Create the plot function to visualize fluid flow by displaying velocity magnitude images and streamlines.
- animation function: Develop the animation function to generate a GIF animation from saved plot images.
- The sliding lid simulation function: Implement the sliding lid function which conducts the main simulation. Initialize fluid properties, establish simulation boundaries, and execute streaming and collision steps over a specified number of iterations. Periodically call the plot function for visualizations.

Algorithm

```
import os, scipy, numpy, matplotlib, Helper files
# Define paths for storing results
def plot(velocities, step, nx, ny):
    # Plot sliding lid simulation results
def animation():
    # Create animation of sliding lid simulation
    # (Code for animation function)
def sliding_lid:
    # Initialize meshgrid and viscosity
    # Set initial conditions based on experiment_type
    # Set Moving and Rigid walls Boundary
    # Loop for num_steps:
        # Cache boundary conditions
```

```
            boundaries_cache(f_inm,f_eq,u_cij)
            # Streaming step
            f_inm = lbm.streaming(f_inm)
            # Apply boundary conditions
            apply_boundaries(f_inm)
            # Collision step
            f_inm = lbm.collision(f_inm, omega)
            # Update density and velocity arrays
            compute_density(f_inm)
            compute_velocity(f_inm, rho_ij)
            # Plot profiles at regular intervals
            # Append deviations to profile
    # Plot simulated and analytical results
# Main block to run the sliding lid simulation
```



(a) $t = 0$ (b) $t = 5000$ (c) $t = 10000$ (d) $t = 20000$

(e) $t = 30000$ (f) $t = 40000$ (g) $t = 50000$ (h) $t = 60000$

(i) $t = 70000$ (j) $t = 80000$ (k) $t = 90000$ (l) $t = 99000$

Figure 4.8: Time evolution of Sliding Lid-Driven Cavity

5

Conclusion

In conclusion, our comprehensive exploration of the Lattice Boltzmann Equation (LBE) and its streaming operator, within the realm of high-performance computing using Python, has yielded valuable insights into computational fluid dynamics.

We began by delving into the essence of the Boltzmann Transport Equation (BTE) and its significance in describing particle movement within fluids. This understanding allowed us to discretize the BTE, transforming complex equations into manageable computations for efficient fluid simulations.

The implementation of periodic boundary conditions expanded our capabilities, enabling the study of diverse scenarios such as interactions with boundaries, pressure gradient-driven flows, and moving walls.

The collision and streaming operator, a core component of the LBE, emerged as a pivotal element for simulating fluid behavior by capturing particle interactions. This exploration deepened our grasp of discretizing complex fluid phenomena, paving the way for accurate and scalable simulations.

The LBE demonstrated its proficiency in simulating wave propagation and decay through the analysis of shear wave decay. This showcased its accuracy in capturing wave behaviors and highlighted its applicability in various fields, from acoustics to material science.

Couette flow and Poiseuille flow investigations underscored the LBE's prowess in simulating a broad range of fluid flows, from laminar to turbulent. Its versatility in modeling fluid motion across diverse regimes empowers researchers to explore fluid dynamics under varied conditions.

The Sliding Lid experiment provided a glimpse into the LBE's application for lid-driven cavity flows, revealing complex vortical structures. This experiment exemplified the method's adaptability in capturing intricate flow patterns and understanding fluid flow physics.

The results and methodologies presented contribute to fluid mechanics and computational science, paving the way for future advancements in this captivating field.

Bibliography

[1] David F. Fletcher. *The future of computational fluid dynamics (CFD) simulation in the chemical process industries*, volume 187. Official Journal of the European Federation of Chemical Engineering, 2022.

[2] Xiaoyi He and Li-Shi Luo. *Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation*, volume 56. American Physical Society, 1997.

[3] Wiley Huang K. *Statistical Mechanics*, volume 2. Chemical Engineering Research and Design, 1987.

[4] Dierk Raabe. *Overview of the lattice Boltzmann method for nano- And microscale fluid dynamics in materials science and engineering*, volume 12. Modelling and Simulation in Materials Science and Engineering, 09 2004.

[5] Guy R. McNamara and Gianluigi Zanetti. *Use of the Boltzmann Equation to Simulate Lattice-Gas Automata*, volume 61. American Physical Society, 1988.

[6] Jean Boon. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, volume 22. European Journal of Mechanics - B/Fluids, 2003.

[7] B. Caroli, C. Caroli, and B. Roulet. *Non-equilibrium thermodynamics of the solidification problem*, volume 66. Journal of Crystal Growth, 1984.

[8] Roger Peyret and Thomas D Taylor. *Computational methods for fluid flow*. Springer Series in Computer Physics, 1980.

[9] X. Shan and H. Chen. *Lattice Boltzmann model for simulating flows with multiple phases and components*, volume 47(3). Physical Review E, 1993.

[10] S. Chen and G.D. Doolen. *Lattice Boltzmann method for fluid flows.*, volume 30(1). Annual Review of Fluid Mechanics, 1998.

[11] Halim Kusumaatmaja Timm Krüger. *The Lattice Boltzmann Method*, volume 1. Springer Cham, 2017.

[12] H Liu and JG Zhou. *Lattice Boltzmann approach to simulating a wetting–drying front in shallow flows*, volume 1. Journal of fluid mechanics, 2014.

[13] A. A. Mohamad. *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes.* Springer: London, 2011.

[14] Andreas Greiner. *Lecture notes in High-Performance Computing: Fluid Mechanics with Python.* University of Freiburg, Summer Semester 2023.

[15] Clausen Aidun, C. K. and J. R. *Lattice-Boltzmann Method for Complex Flows*, volume 42. . Annual Review of Fluid Mechanics, 2010.

[16] Sauro Succi. *The lattice Boltzmann equation: for complex states of flowing matter.* Oxford University Press, 2018.

[17] Qisu Zou and Xiaoyi He. *On pressure and velocity boundary conditions for the lattice Boltzmann BGK model*, volume 9. American Institute of Physics, 1997.

[18] Sorush Khajepor, Jing Cui, Marius Dewar, and Baixin Chen. *A study of wall boundary conditions in pseudopotential lattice Boltzmann models*, volume 193. Elsevier, 2019.

[19] S Frijters. *Lattice Boltzmann Method with Python.* Springer., 2018.

[20] A. N. Gorban, M. Kazachkov, and D. A. Kuzmin. *Parallelization of lattice Boltzmann simulations for fluid flow problems.* Parallel Computational Fluid Dynamics, 2003.