High-Performance Computing: Fluid Mechanics with Python

Final Report

Wardi Mohamed Amine
Matriculation number : 5652579
mw776@students.uni-freiburg.de

Computer Science Department
Summer Semester - 2023

Contents

1 Introduction

Fluid mechanics is a foundational discipline for understanding the behavior of fluids in various scientific and engineering domains. Computational methods, particularly the lattice Boltzmann method (LBM), have revolutionized fluid flow simulations, providing insights into complex fluid dynamics. The LBM is derived from kinetic theory and enables the modeling of fluid flow by dividing the computational domain into a lattice grid. It involves the streaming operator, which governs particle movement, and the collision operator, which models particle interactions.

In the context of high-performance computing (HPC) for fluid mechanics, the implementation of the lattice Boltzmann equation (LBE) is of paramount importance. HPC systems provide the necessary computational power to handle the complex simulations efficiently. By utilizing parallel computing techniques and distributing the workload across multiple processors or computing nodes, HPC enables faster and more accurate simulations. This allows researchers and engineers to study complex fluid flows, such as turbulent flows, multiphase flows, and flows in complex geometries.

To validate and refine the lattice Boltzmann model, numerous simulations replicating well-known fluid mechanics experiments are conducted. These simulations serve as benchmarks to confirm the accuracy and reliability of the model while enhancing our understanding of fluid mechanics phenomena. The behavior of shear waves, Couette flow, Poiseuille flow, and the sliding lid experiment are investigated, providing insights into the strengths and limitations of the model and opportunities for refinement.

While the initial focus is on the serial implementation of the lattice Boltzmann equation, later stages involve the application of parallelization techniques to advance the study. This comprehensive approach includes presenting the principles of the lattice Boltzmann equation, and the collision operator. Details of the implementation and simulation results are discussed.

2 Methods

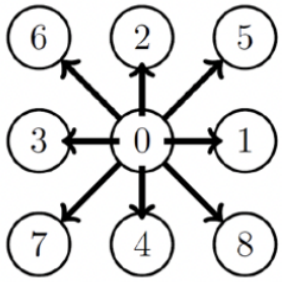
The Boltzmann Transport Equation (BTE) [1] can be expressed as :

$$\frac{\partial f(\mathbf{r},\mathbf{v},t)}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{r}} f(\mathbf{r},\mathbf{v},t) + \mathbf{a}\cdot\nabla_{\mathbf{v}} f(\mathbf{r},\mathbf{v},t) = C(f(\mathbf{r},\mathbf{v},t)) \tag{1}$$

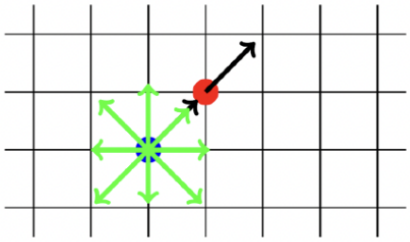
The streaming term is represented by the left-hand side (l.h.s.), while the right-hand side (r.h.s.) corresponds to the collision term C(f). The purpose of the equation 1 is to determine the probability density function f(r, v, t), also known as the distribution function.

2.1 Discretization of BTE

The probability density function $f(r,v,t)$ depends on r, v and t. Hence, discretization is necessary for f in terms of space r, velocity v, and time t. We discretize f in space using a structured mesh that consists of equidistant grid elements in an n-dimensional setup. For the discretization of velocity space, we impose constraints on particle movement along predefined directions denoted as c_i. In the case of 2D simulations, we utilize the D2Q9 scheme as shown in Figure 1a, which encompasses nine directions (c_i). Consequently, we represent f using the notation $f_i(r,t)$, where the index i corresponds to the specific directions considered.



(a) D2Q9 Discretization



(b) Uniform 2D grid for the discretization

Figure 1: Discretization of the BTE [1]

Our grid is the superposition of Figure 1a and Figure 1b.

The density and velocity field can be calculated using these two equations respectively [1] :

$$\rho(r,t) = \sum_i f_i(r,t) \tag{2}$$

$$u(r,t) = \frac{1}{\rho(r,t)} \sum_i c_i \cdot f_i(r,t) \tag{3}$$

2.2 Streaming Operator

In this section, to solely focus on the streaming component of the equation, we will be setting the collision term to zero, we will disregard any interactions between particles. This approach allows us to derive a transport equation that describes the movement of particles in a vacuum, where mutual particle interaction is not taken into consideration. The LBE can now be written as :

$$f_i(r + c_i\Delta t, t + \Delta t) = f_i(r,t) \tag{4}$$

In accordance with equation 4, it is demonstrated that the function f can be advanced from one time step to the next by propagating it along each direction represented by c_i.

2.3 Collision Operator

The collision term on the $r.h.s$ of the equation 1 accounts for particle interactions, typically involving a complex two-particle scattering integral. To simplify this term, we employ a relaxation time approximation. This approximation assumes that the distribution function $f(r,v,t)$ locally relaxes towards an equilibrium distribution $f_eq(r,v,t)$. With this approximation, the BTE can be written as follows [1]:

$$f_i(\mathbf{r} + \Delta t\mathbf{c}_i, t + \Delta t) = f_i(\mathbf{r},t) + \omega(f_i^{eq}(\mathbf{r},t) - f_i(\mathbf{r},t)) \tag{5}$$

Where ω is the relaxation term.

By using the density and velocity function shown in equations 2 and 3, we can calculate the equilibrium distribution function [1] :

$$f_{\text{eq}i}(\rho(\mathbf{r}), \mathbf{u}(\mathbf{r})) = w_i\rho(\mathbf{r}) \left[1 + 3c_i \cdot \mathbf{u}(\mathbf{r}) + \frac{9}{2}(c_i \cdot \mathbf{u}(\mathbf{r}))^2 - \frac{3}{2}|\mathbf{u}(\mathbf{r})|^2\right] \tag{6}$$

Where $w_i = \left(\frac{4}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}\right)$.

2.4 Shear Wave Decay

The shear wave decay experiment in the lattice Boltzmann equation involves simulating fluid flow with shear forces. A homogeneous fluid is initialized with a shear perturbation. The simulation evolves the fluid using the LBE, accounting for collisions and streaming operations. The propagation and decay of the shear wave are observed over time. Analyzing the wave's dispersion and attenuation provides insights into the accuracy and stability of the LBE model. The shear wave decay is widely used in computational physics to measure the kinematic viscosity of a fluid. This method involves setting a sinusoidal velocity profile in the simulation domain and measuring the rate at which the wave decays, providing a means to estimate the fluid's viscosity.

As a first experiment, we select the initial distribution of density, $\rho(r,0)$, as $\rho_0 + \epsilon \sin(\frac{2\pi y}{L_x})$, and the initial velocity component in the x-direction, $u_x(r,0)$, as 0. We then proceed to observe and analyze the outcomes of the simulation.

For the second experiment, we select the initial distribution of density, $\rho(r,0)$, as 1, and the initial velocity component in the x-direction, $u_x(r,0)$, as $\epsilon \sin(\frac{2\pi y}{L_y})$, where ϵ represents a small amplitude and L_y is the length of the domain in the y-direction. In other words, we introduce a sinusoidal variation in the x-velocity component with respect to the y-position.

We assume that these initial conditions follow the Stokes flow condition [1]:

$$\frac{\partial \mathbf{u}}{\partial \mathbf{t}} = \nu \cdot \Delta \mathbf{u} \tag{7}$$

The velocity has only a component in the x-direction and only depends on the y variable, so we can write the previous equation 7 like the following :

$$\frac{\partial \mathbf{u_x}(\mathbf{y},\mathbf{t})}{\partial \mathbf{t}} = \nu \cdot \frac{\partial^2 u_x(y,t)}{\partial y^2} \tag{8}$$

The solution to the equation 8 can be easily found, which is :

$$u_x(y,t) = a(t) \cdot \sin(\frac{2\pi y}{L_y}) \tag{9}$$

By replacing $u_x(y,t)$ found in the equation 9 in the equation 8, we can find the relation between the amplitude a, which is easily calculated in our experiment, and the viscosity ν.

$$a(t) = a_0 e^{-\nu k t} \tag{10}$$

Where $k = \frac{2\pi}{L_y}$.

The theoretical way of calculating the viscosity is [1]:

$$\nu = \frac{3}{2}(\frac{1}{\omega} - \frac{1}{2}) \tag{11}$$

As we can see, it depends on the relaxation term ω.

2.5 Moving and Fixed Walls

Previously, our simulations were conducted under the assumption of periodic boundary conditions (PBC). PBC allows for a seamless continuation of the simulation domain, treating opposite boundaries as if they were connected. This approach eliminates the need for explicitly defining walls or boundaries within the simulation. However, in the current study, we are introducing fixed and moving walls to the system. This modification enables us to investigate the behavior of the fluid in the presence of specific boundary conditions.

A fixed wall refers to a boundary condition where the velocity of the fluid at the wall is set to zero. This condition represents a rigid, immovable wall that restricts fluid motion. At a fixed wall, the fluid particles adjacent to the wall experience a no-slip condition, meaning their velocity relative to the wall is zero. In a broader sense, we can state that the

population departing from the boundary node x at time t will encounter the wall at $t+\Delta t$ and eventually arrive at the corresponding channel on the opposite side of the boundary node at t [1]. As a result, for the populations leaving the boundary node, the streaming operation is substituted with the equation 12.

$$f_{\bar{i}}(x, t+\Delta t) = f_i^*(x, t) \tag{12}$$

Where the index $\bar{i}$ denotes the opposite channel of i (see Table 1) and f^* denotes the value of the probability density function before the streaming step.

Channel	Opposite Channel
0	0
1	3
2	4
3	1
4	2
5	7
6	8
7	5
8	6

Table 1: Channels and their opposing channels

In contrast to a fixed wall, a moving wall is a boundary condition where the wall has a specified velocity. This condition simulates a boundary that can either accelerate or decelerate with time, affecting the flow of the fluid. At a moving wall, the fluid particles in contact with the wall move with the same velocity as the wall itself. To account for the alteration in momentum, we incorporate an additional term into the equation representing the simple rigid wall scenario, as shown in the equation 13. This extra term is directly related to the velocity of the wall, denoted as u_w [1].

$$f_{\bar{i}}(x, t+\Delta t) = f_i^*(x, t) - 2w_i\rho_w\frac{c_i u_w}{c_s^2} \tag{13}$$

Where ρ_w is the average density and $c_s^2 = \frac{1}{3}$.

Using the three wall types mentioned, we conducted two experiments. The initial experiment involved studying the Couette flow, employing a moving wall for the top boundary, a fixed wall for the bottom boundary, and periodic boundary conditions (PBC) for the right and left boundaries. In the second experiment, known as the sliding lid, we used a fixed wall for the bottom, right, and left boundaries, while the top boundary was represented by a moving wall. To characterize the behavior of the sliding lid experiment, we employed the Reynolds number ($Re = \frac{Lu}{\nu}$) [1], a dimensionless quantity that relates the effects of inertia to viscosity, serving as a ratio between these two terms.

2.6 Poiseuille Flow

In the Poiseuille Flow experiment, a steady laminar flow is induced by applying a pressure gradient along the channel or pipe's length. The flow is characterized by a parabolic velocity profile, with the maximum velocity occurring at the center of the channel and decreasing towards the walls. Periodic boundary conditions can also be used in the presence of a prescribed pressure variation $\Delta p = p_{out} - p_{in}$ between the outlet and the inlet. The relationship between pressure and density is governed by the ideal gas equation of state [1] :

$$p = \rho \cdot c_s^2 \tag{14}$$

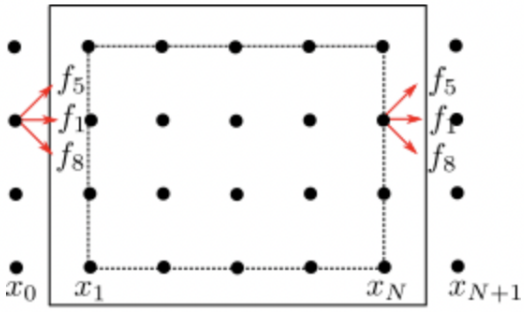


Figure 2: Representation of the periodic boundary conditions with extra nodes [1]

To establish periodic conditions, an additional layer of nodes beyond the physical domain needs to be defined, as depicted in Figure 2. Conceptually, if we envision the periodic conditions as multiple connected pipes, it becomes evident that the extra node x_0 at the inlet corresponds to the last node x_N within the previous pipe's physical domain. Consequently, by relating x_N to its corresponding image node x_0, and likewise, x_1 to its image node x_{N+1}, the periodic conditions can be effectively implemented [1].

To ensure periodicity, such as in the x-direction, it is necessary to enforce equality between variables at x_0 and x_N, as well as between x_{N+1} and x_1. In terms of the non-equilibrium component, the value of the corresponding image node can be copied to maintain consistency across the periodic boundaries [1].

The boundary conditions can be defined by using the equations 15 and 16 [1].

$$f_i^*(x_0, y, t) = f_i^{eq}(\rho_{in}, u_N) + (f_i^*(x_N, y, t) - f_i^{eq}(x_N, y, t)) \tag{15}$$

$$f_i^*(x_{N+1}, y, t) = f_i^{eq}(\rho_{out}, u_1) + (f_i^*(x_1, y, t) - f_i^{eq}(x_1, y, t)) \tag{16}$$

Where $\rho_{out} = \frac{p_{out}}{c_s^2}$ and $\rho_{in} = \frac{p_{in}+\Delta p}{c_s^2}$

To verify our experiment, we incorporate the analytical solution of the velocity field for a Hagen-Poiseuille flow in a pipe. By assuming laminar flow, we can simplify the Navier-Stokes equations and focus solely on the streamwise component in the direction of the pipe axis [1].

$$\frac{\partial p(x)}{\partial x} = \frac{1}{\mu}\frac{\partial^2 u_x(y)}{\partial y^2} \tag{17}$$

Using the boundary conditions $u(0) = 0$ and $u(h) = 0$, we obtain the velocity profile :

$$u(y) = \frac{1}{2\mu}\frac{dp}{dx}y(y-h) \tag{18}$$

Where $\mu = \rho\nu$

2.7 Parallelism

In computer science, parallelism refers to the technique of simultaneously executing multiple tasks, resulting in significantly faster and more efficient programs. This is achieved by breaking down the problem into smaller tasks that can be executed independently or interdependently, and then running them simultaneously. Modern central processing units (CPUs) found in computers, phones, and other hardware often possess multiple cores capable of executing instructions in parallel. Additionally, there are massively parallel computing systems where multiple CPUs are grouped into nodes that share a common memory. These nodes are then interconnected via a network to create a comprehensive computing system [2]. In our case, parallelism is employed to speed up the runtime of the sliding lid experiment.

3 Implementation

In this section, we will demonstrate the implementation of the methods discussed in the previous section using Python and NumPy arrays. Before we proceed, it is crucial to define the velocity set c, as it plays a vital role in understanding the subsequent algorithms. It is defined as follows [1]:

$$c = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}$$

3.1 Streaming Operator

To implement the streaming operator, we iterate through each velocity channel and utilize the np.roll function in the following manner :

```
for i in range(number_channels) :
    res[i, :, :] = np.roll(res[i, :, :], shift=(c[0, i], c[1, i]),
                           axis=(0, 1))
```

After each streaming step, we verify whether mass conservation is maintained within our grid. If it is not the case, we print an error message stating that the mass is not conserved.

3.2 Collision Operator

To account for the collision term, we initially devised a procedure to compute the equilibrium distribution function by utilizing a grid of density and velocity values. Subsequently, we employed this calculated equilibrium distribution function along with a specified ω value to incorporate the collision term into our model.

```
def equilibrium_distribution(rho_grid, u_grid) :
    product = np.einsum('ji, jkl -> ikl', c, u_grid)
    norm = np.linalg.norm(u_grid, axis=0)**2
    return w_i[:, np.newaxis, np.newaxis] * \
           rho_grid[np.newaxis, :, :] * \
           (1 + 3 * product + (9/2) * \
           product**2 - (3/2) * norm[np.newaxis, :, :])
```

```
def collision_term(grid, omega) :
    return grid + omega*(equilibrium_distribution(rho(grid),
                         u(grid)) - grid)
```

3.3 Shear Wave decay

The shear wave decay experiment was implemented following these steps :

- Create a function, that initializes $\rho(r, 0)$ to 1, and $u_x(r, 0)$ to $\epsilon \sin(\frac{2\pi y}{L_y})$.
- Calculate the probability density grid by using the equilibrium distribution on the initial ρ and u.
- We can repeatedly apply the streaming function, which performs the shift operation, followed by the collision function, to observe and analyze the behavior of the system. By carrying out these steps multiple times, we can track the changes in the system and calculate the amplitude. This allows us to determine the viscosity.

- We do the previous steps for different omegas to compare the results to the theoretical viscosity.

3.4 Couette Flow

To simulate the Couette flow experiment, we adopted a method that involved incorporating ghost points to represent the walls concerned by the changes. We created these additional lines, effectively reversing the flow so that the top channels became the bottom channels and vice versa. This approach allowed us to emulate the presence of walls within the simulation.

The boundary conditions were applied after each streaming step (In the Streaming function). During this process, we iterated through the cells of the two walls and reversed the channels for each cell.

```
def couette_flow_conditions(grid, copy) :
    c_s_square = 1/3
    for x in range(grid.shape[1]) :
        top = np.copy(copy[:, x, -1])
        bottom = np.copy(copy[:, x, 0])
        rho_n = density(grid)
        avg_density = rho_n.mean()
        grid[bottom_boundary, x, -1] = [top[5] -
            2*w_i[5]*avg_density*np.dot(c[:, 5], u_w)/c_s_square,
                                        top[2] -
            2*w_i[2]*avg_density*np.dot(c[:, 2], u_w)/c_s_square,
                                        top[6] -
            2*w_i[6]*avg_density*np.dot(c[:, 6], u_w)/c_s_square]

        grid[top_boundary, x, 0] = bottom[bottom_boundary]

    return grid
```

3.5 Sliding Lid

To simulate the sliding lid experiment, we utilized a similar approach by introducing ghost lines to emulate the presence of walls. These additional lines allowed us to establish fixed walls at the bottom, right, and left boundaries. Meanwhile, the top boundary was represented by a moving wall. This configuration facilitated the desired sliding lid effect in the simulation.

Similar to the Couette flow experiment, the boundary conditions were applied after each streaming step. During this process, we iterated through the cells of the fixed walls and appropriately adjusted the channels.

```
    # Boundaries for the right and left
    for y in range(grid.shape[2]) :
        right = np.copy(copy[:, -1, y])
        left = np.copy(copy[:, 0, y])

        grid[left_boundary, -1, y] = right[right_boundary]
        grid[right_boundary, 0, y] = left[left_boundary]
```

3.6 Poiseuille Flow

Like in the previous experiments, we define our new boundary conditions with the ghost points :

```
rho_out = np.full((shape[1], shape[2]), p_out/c_s_squared)
rho_in = np.full((shape[1], shape[2]),
                 (p_out+delta_p)/c_s_squared)

f_eq = equilibrium_distribution(rho, u)

f_eq_out = equilibrium_distribution(rho_out, u_1)[:, 1, :]
f_eq_in = equilibrium_distribution(rho_in, u_N)[:, -2, :]

grid[channels, -1, :] = (f_eq_out +  grid[:, 1, :]
                         - f_eq[:, 1, :])[channels, :]
grid[channels, 0, :] = (f_eq_in +  grid[:, -2, :]
                        - f_eq[:, -2, :])[channels, :]
```

Subsequently, we conduct a series of streaming steps until we obtain the desired plot. The number of steps required may vary depending on the specific objectives of the experiment.

3.7 Parallel Sliding Lid

To implement the parallel sliding lid experiment, we are going to use to `mpi4py` library. It is a Python library that enables Message Passing Interface (MPI) communication for parallel computing.

To implement this approach, the 2D domain was divided into smaller parts through decomposition. Initially, the number of processors was determined, followed by the computation of nodes along the x and y axes. The subgrid shapes were then derived using equations 19 and 20.

$$subgrid\_shape\_x = grid\_shape\_x//2 + 2 \tag{19}$$

$$subgrid\_shape\_y = grid\_shape\_y//2 + 2 \tag{20}$$

To incorporate borders into the subgrids, we include additional cells known as ghost cells (added by `+2` in the shape of the subgrid), which facilitate communication between processes. For communication purposes, we utilize the **MPI.COMM_WORLD** communicator, from which we construct the Cartesian communicator.

```
cartcomm = comm.Create_cart(dims=[node_shape_x, node_shape_y],
                            periods=(False, False), reorder=False)
```

Next, for every process, we obtain the source and destination information for all communications in every direction (right, left, up, down) through the Cartesian communicator.

```
right_src, right_dst = cartcomm.Shift(1, 1)
left_src, left_dst = cartcomm.Shift(1, -1)
up_src, up_dst = cartcomm.Shift(0, -1)
down_src, down_dst = cartcomm.Shift(0, 1)
```

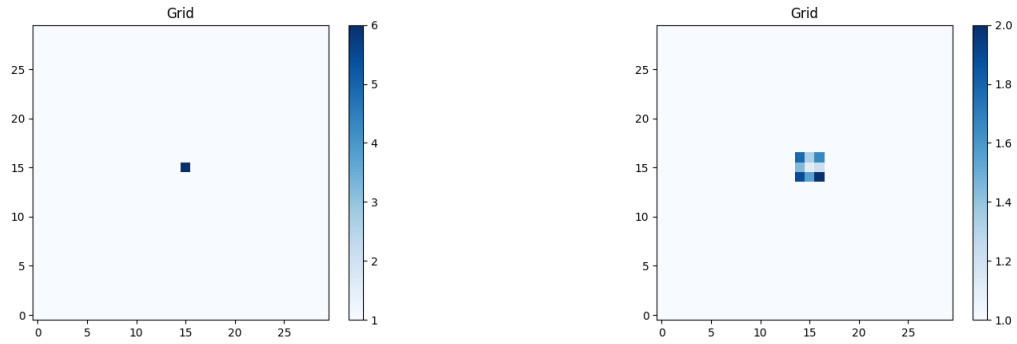
Subsequently, each process exchanges the required values of the ghost cells with other processes using the `Sendrecv` method of `mpi4py`, enabling them to proceed with computations within their respective subgrids.

In order to execute our code with multiple processes, we utilized BwUniCluster, an HPC cluster commonly employed by universities and research institutions for its high-performance computing capabilities.

4 Results

4.1 Streaming Operator

To validate our streaming implementation, we generated a 30x30 grid where only one point had a value of $\frac{i}{9}$ for each velocity direction represented by c_i. As depicted in Figure 3b, we observe that the streaming process aligns with the theoretical explanation provided in the earlier sections of this report.



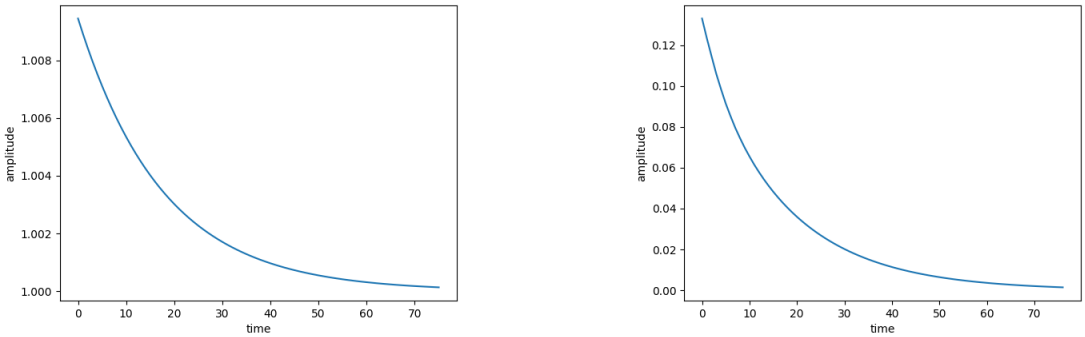
(a) Before Streaming

(b) After Streaming

Figure 3: Streaming Operator

4.2 Shear Wave decay

In the theoretical portion of the shear wave decay experiment, we conducted two experiments to evaluate the decay of sinusoidal wave amplitudes. To simulate these experiments, we performed 20000 streaming steps on a grid with dimensions of 300x300. The simulation was carried out with a parameter value of $\omega = 0.5$, which influenced the behavior of the wave propagation. These details were taken into account to accurately capture the decay phenomenon seen in Figure 4. The ϵ was equal to 0.01 for the first experiment where the density was sinusoidal, and equal to 0.1 for the second experiment.



(a) Density Amplitude

(b) Velocity Amplitude

Figure 4: Plot of amplitude for $\omega = 0.5$

An alternative method to illustrate the decay of the velocity's amplitude over time is by projecting the density profile along the X-axis instead of time, as shown in Figure 5. This approach provides insights into the rate at which the decay occurs and allows us to examine the amplitudes both as a function of time and position.

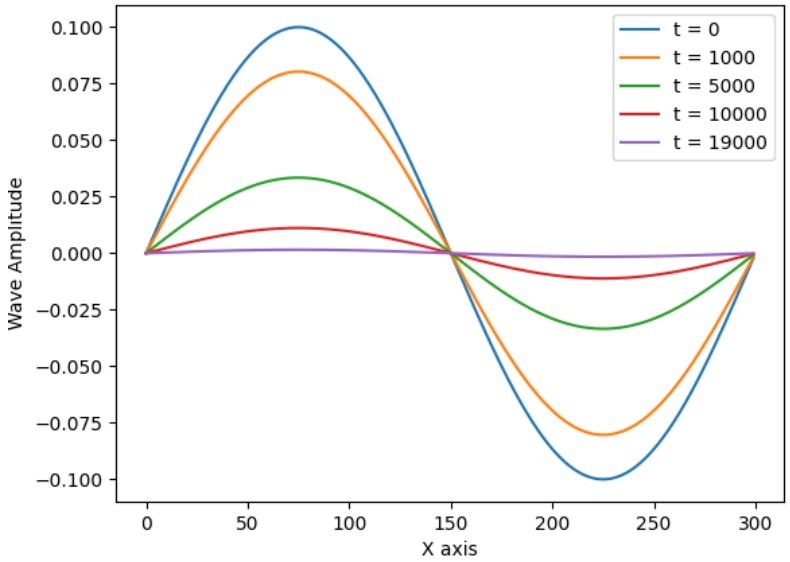


Figure 5: Shear Wave Decay with sinusoidal velocity for $\omega = 0.5$

With the identical simulation parameters discussed earlier, we computed the viscosity for various values of ω (ranging from 0.1 to 1.9). Figure 6 presents a comparison between the calculated viscosity and the theoretical viscosity

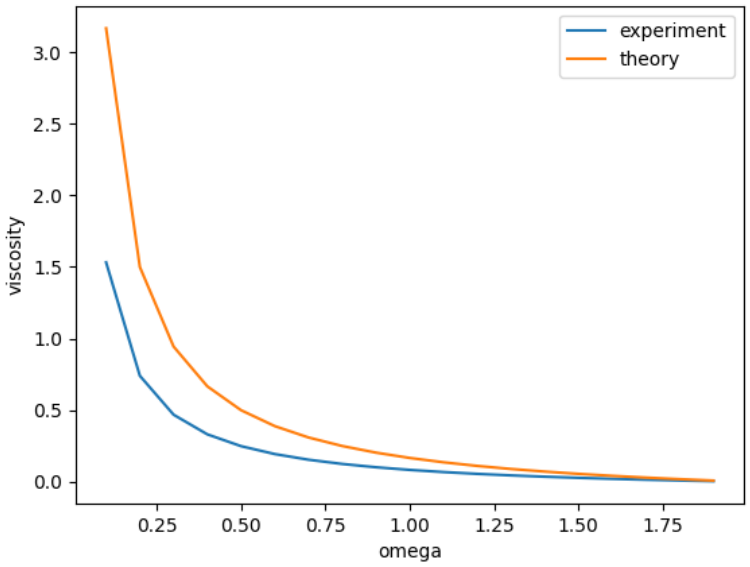


Figure 6: Theoretical Viscosity vs Experiment Viscosity

As the relaxation parameter ω decreases in lattice Boltzmann simulations, the departure from Stokes flow becomes more pronounced. Inertial effects become significant, making the simple Stokes flow assumption inadequate. The dynamics of the fluid become more complex, potentially including turbulence and nonlinear behavior. To accurately capture these effects, more advanced modeling techniques or numerical methods are required. Adjusting ω allows for a trade-off between computational efficiency and capturing inertial effects.

4.3 Couette Flow

Figure 7 illustrates the evolution of velocity at a specific cross-section of the lattice, perpendicular to the x-axis, captured at various simulation time steps. Notably, when examining the points located at the fixed wall, the velocity is null on the horizontal axis, it can be observed that the velocity remains consistently zero at the bottom wall throughout all simulation steps. Furthermore, a notable observation is that the velocity of the points associated with the moving boundary (horizontal axis value of 100) is equivalent to the velocity of the boundary itself, in our case 0.1. The simulation was done on 100x100 grid with $\omega = 1$, $u_w = [0.1, 0]$, $\rho(r, 0) = 1$, $u(r, 0) = 0$, and in 20000 time steps.

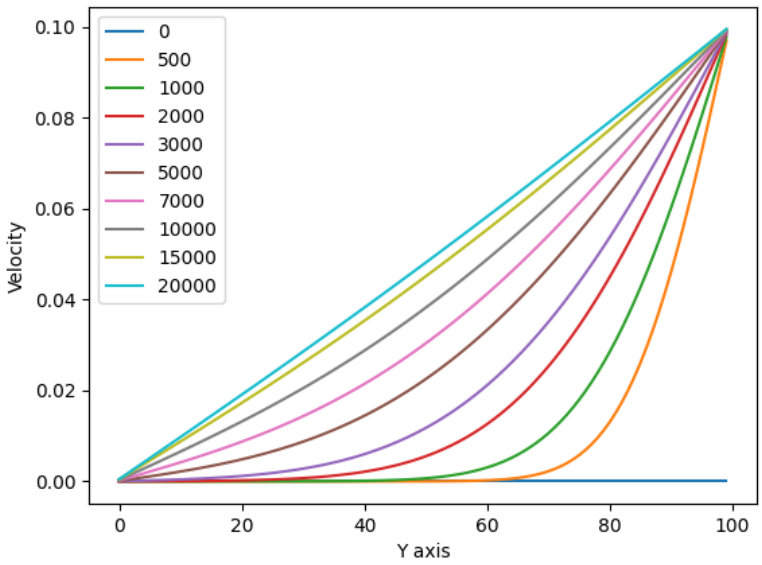
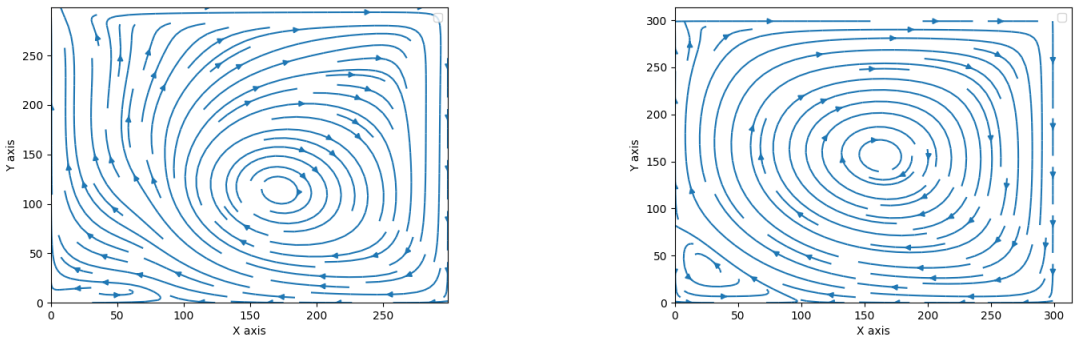


Figure 7: Velocity in Couette Flow

Each color in Figure 7 represents a different time step.

4.4 Sliding Lid

Figure 8 illustrates the outcomes of the sliding lid simulation, conducted within the same environment as the Couette flow experiment, except for a different value of ω, set to 1.7 to yield a Reynolds number (Re) of 1000. The depicted results indicate the accuracy of the simulation, as they clearly demonstrate a left-to-right flow pattern and the emergence of circular motion. This circular motion arises due to the presence of fixed walls on the right and left sides of the simulation domain.



(a) Time step = 20000

(b) Time step = 100000

Figure 8: Sliding Lid for $Re = 1000$

4.5 Poiseuille Flow

Figure 9 depicts the velocity profile at various time steps and compares them to the theoretical velocity. The simulation was conducted on a 300x100 grid utilizing the equilibrium distribution function with a density of $\rho_0 = 1$ and an initial velocity of $u_0 = 0$. The parameters employed were $\omega = 1$, $p_{in} = 0.03$, and $p_{out} = 0.3$.

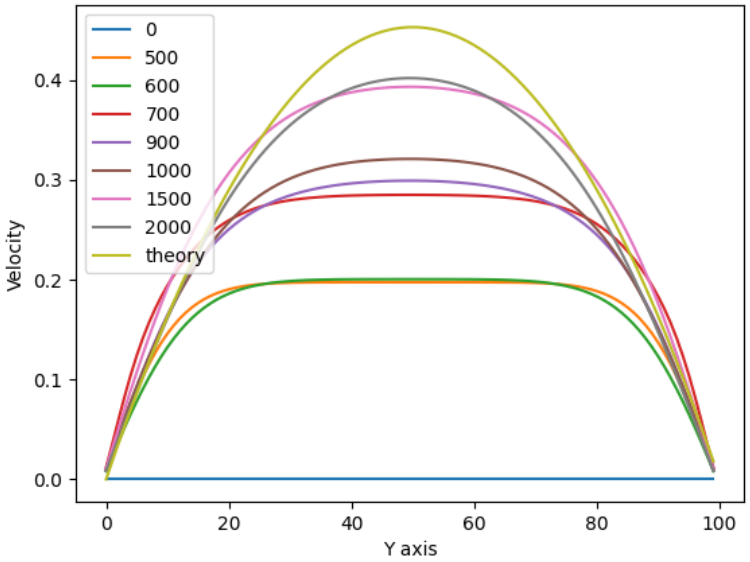


Figure 9: Velocity in Poiseuille Flow

4.6 Parallel Sliding Lid

To validate our parallel code, we opted to display the velocity streamplot, similar to the standard sliding lid experiment, using identical parameters. Figure 10 demonstrates the outcome, reflecting the typical velocity behavior in the sliding lid experiment.

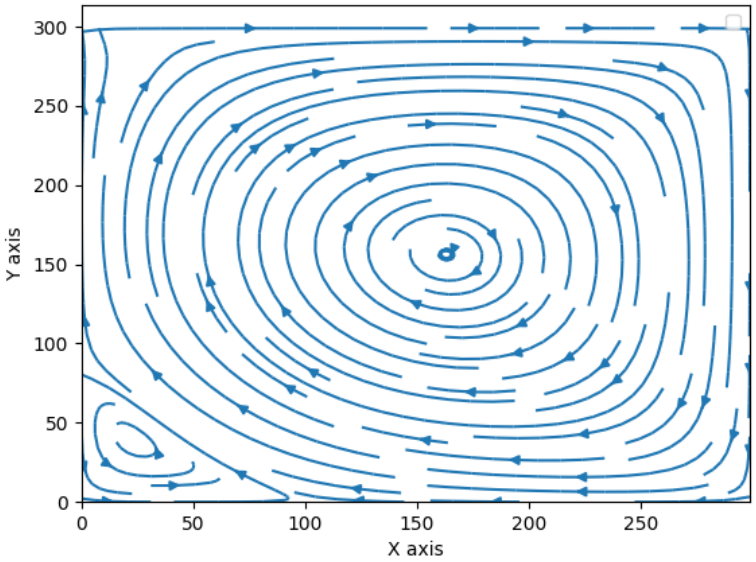


Figure 10: Sliding Lid for parallel method where $Re = 1000$ and Time step $= 100000$

We were also curious about the scalability of our code when employing more processes. As evident in Figure 11, the Million Lattice Updates Per Second (MLUPS), calculated using equation 21, exhibits a rapid increase from 1 to 25 processes, followed by a gradual rise beyond that point.

$$mlups = \frac{300 \times 300 \times timesteps}{10^6 \times time}[1] \tag{21}$$

Where in this case $timesteps = 100000$ and $time$ was the runtime of our code.

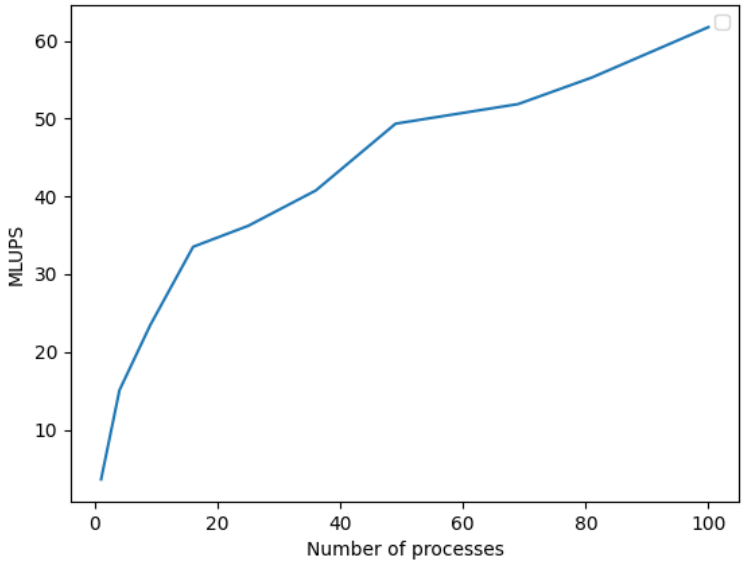


Figure 11: MLUPS for different number of MPI processes for a 300x300 grid

5 Conclusion

In conclusion, this report presents a comprehensive study of the lattice Boltzmann method (LBM) as applied to fluid mechanics simulations. By discretizing the Boltzmann Transport Equation (BTE) and employing the streaming and collision operators, the LBM successfully models particle movement and interactions in fluid flows.

The implementation of the lattice Boltzmann equation (LBE) on high-performance computing (HPC) systems is of paramount importance to handle complex simulations efficiently. Parallel computing techniques play a crucial role in achieving faster and more accurate simulations by distributing the workload across multiple processors or computing nodes.

Through the validation and refinement of the lattice Boltzmann model, various well-known fluid mechanics experiments were replicated as benchmarks. The simulations of shear wave, Couette flow, Poiseuille flow, and the sliding lid experiment have provided valuable insights into the strengths and limitations of the model. Additionally, they have offered opportunities for further improvements and refinements to enhance our understanding of fluid mechanics phenomena.

References

[1] Andreas Greine. High performance computing: Fluid mechanics with python. Lecture at the University of Freiburg, Summer Semester 2023.

[2] heavy.ai. Parallel computing, https://www.heavy.ai/technical-glossary/parallel-computing.