High Performance Computing:

Computational Fluid Dynamics with Python

Siddhant Saka

5253227

mr.siddhantsaka@gmail.com

GitHub:
https://github.com/Siddhantsaka/HPC_Python

Contents

1. Introduction: 2
2. Lattice Boltzmann Methodology: 3
2.1 Boltzmann Transport Equation (BTE)- 3
3. Implementation: 6
3.1 Streaming: 6
3.2 Collision: 6
3.3 Shear Wave Decay: 7
3.4 Couette Flow: 8
3.5 Poiseuille flow: 9
3.6 Sliding Lid: 10
3.7 Using MPI (Message Passing Interface) for Parallel execution: 11
4. Numerical Results and Validation: 13
4.1 Shear wave decay: 13
4.2 Couette Flow: 16
4.3 Poiseuille Flow: 17
4.4 Sliding Lid: 18
5. Conclusion: 22
6. References: 23

1. Introduction:

With the advancement of engineering processes, there has been a significant improvement in methodologies for experimentation. These experiments often come with high costs. Without having prior knowledge of underlying principles and expected interactions, it is unfeasible. These issues are mitigated by performing numerous simulations. Currently, there's continuous ongoing research to increase optimization for efficient utilization of computer resources. This efficiency is typically measured using Big O notation. The advancements in optimizing simulations have an added advantage of improving accuracy along with stability.

This report highlights the basic aspects of simulation in fluid flow prioritizing the usage and significance of Lattice Boltzmann Method (LBM) that was developed by Shan and Chen [1]. This methodology is proven for incorporating multi-component fluid flow along with the added benefit of stability and highly flexible nature that is not usually found in traditional CFD methodologies [2], [3].

Choosing LBM has significant advantages[4], [5], some of which are listed below.

1. Provides numerical stability to the simulation.
2. Offers flexibility to incorporate more parameters and complex fluid interactions.
3. Ability to handle macros and mesoscale simulations easily.
4. Ensures precision in applying fluidic properties.

In LBM, fluid particles are considered as distribution functions that enable the mesoscopic scale implementation of Navier-Stokes equation (NVE). This is done by utilizing the lattice automata concept for fluid simulations [6]. Unlike traditional CFD methods like Finite Volume Method (FVM) and Finite Difference Method (FDM), this process simplifies the computational process and reduces complexity [7]. There is a vast feasibility for parallel computation in LBM that enables this method to work very efficiently for big datasets [8].

This course aims to provide a comprehensive examination of LBM that covers the following-

1. the following- Theoretical concepts.
2. Mathematical formulation.
3. Various implementation techniques.
4. Detailed discussion and Results.
5. Efficiency measurement

2. Lattice Boltzmann Methodology:

This section of the report describes the usage of Boltzmann Transport Equation (BTE) for CFD using Python. Here a 2-Dimensional lattice (D2Q9 model) is utilized for fluid flow simulation. The density distribution of fluid particles is discretized across the mesh. Two main components are emphasized – streaming and collision. It also discusses the importance of updating time step in BTE that is required for maintaining the stability of the simulation.

2.1 Boltzmann Transport Equation (BTE)-

BTE is the foundation for LBM. It offers a statistical approach for fluid dynamics by utilizing the evolution of probability distribution function (PDF) $f(\vec{x}, \vec{v}, t)$. The PDF represents the probability that a particle with velocity $\vec{v}$ is encountered at position $\vec{x}$ at time t [9]. The comprehensive form of BTE is represented as:

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \nabla f + \vec{F} \cdot \nabla_v f = \Omega(f) \tag{1}$$

Here:

- $\frac{\partial f}{\partial t}$ refers to change of pdf with respect to time.
- $\vec{v} \cdot \nabla f$ refers to the inclusion of particle velocity-induced advection.
- $\vec{F} \cdot \nabla_v f$ this adds the influence of external forces on the particles.
- $\Omega(f)$ denotes the collision operator, that represents change in f due to particle collisions [10].

The collision operator $\Omega(f)$ can be simplified for computationally feasible calculations using Bhatnagar-Gross-Krook (BGK) model that can be represented as:

$$\Omega(f) = -\frac{1}{\tau}(f - f^{eq}) \tag{2}$$

In equation 2, f^{eq} refers to equilibrium distribution function, and τ is the relaxation time taken by the system to come back to equilibrium state [11].

In D2Q9 model of LBM, 9 discrete lattice velocities ($\vec{c}_i$), are used to simulate fluid dynamics on a 2D grid. Here i = 0 represents the stationary behavior. i = 1,3,5,7 represent the movement in 4 cardinal directions. i = 2,4,6,8 represent diagonal transitions. This discretization of velocity vector enables the precise modeling of particle movement in a square lattice with each vector ($\vec{c}_i$) representing the direction of travel and magnitude at every time step Δt. The uniform grid enables synchronized movement of particles by allowing them to move to adjacent nodes (or remain stationary).

The discrete lattice velocities can be represented by the following matrix:

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

where the rows denote the x component and columns denote the y component of the velocity vector.

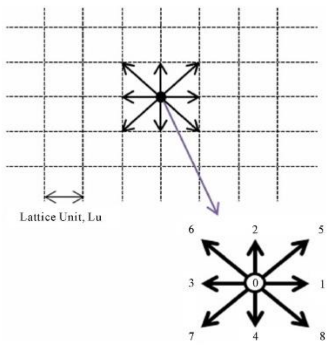


Figure 2.1- Example of discretization of velocity and the positioning of particle in the lattice [12].

Streaming in LBM involves the movement of particles (PDF: $f_i(\vec{x},t)$) across the lattice grid this can be mathematically represented as:

$$f_i(\vec{x} + \vec{c_i}\Delta t, t + \Delta t) = f_i(\vec{x},t) \tag{3}$$

Here f_i refers to the PDF in direction i. Velocity vector in direction i is $\vec{c_i}$ and $\vec{x}$ is the node position of the particle. Δt refers to the change in time step. The equation highlights the PDF at node and its propagation at time Δt [10].

Collision phase involves the use of Bhatnagar-Gross-Krook (BGK) operator that simplifies the collision dynamics by linearizing the relaxation near the equilibrium. This plays an important role in the simulation as it recreates the macroscopic behavior of the fluid. The collision can be represented by:

$$f_i(x + c_i\Delta t, t + \Delta t) = f_i(x,t) + \frac{1}{\tau}\left(f_i^{eq} - f_i(x,t)\right), \tag{4}$$

Where τ refers to relaxation time and f_i^{eq} refers to the equilibrium distribution function. BGK approximation simplifies the collision operator as it assumes a single uniform relaxation time for all particles. This models the distribution function to relax towards Maxwellian equilibrium state. f_i^{eq} can be represented by the following equation [13]:

$$f_i^{eq} = \rho w_i \left(1 + 3\mathbf{c_i} \cdot \mathbf{u} + \frac{9}{2}(\mathbf{c_i} \cdot \mathbf{u})^2 - \frac{3}{2}\mathbf{u^2}\right), \tag{5}$$

The weights have the following values:

$$W_i = \frac{4}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}, \frac{1}{36}$$

The macroscopic entities including density (ρ) and velocity (u) is computed as follows:

$$\rho = \sum_i f_i \tag{6}$$

Velocity can be calculated in the following equation:

$$u = \frac{1}{\rho} \sum_i c_i f_i \tag{7}$$

The relaxation time τ inversely controls the fluid kinematic viscosity(ν) given by $c_s^2(\tau - 0.5)\Delta t$, where c_s is the speed of sound in lattice.

Time step Δt selection is based on Courant-Friedrichs-Lewy (CFL) inequality condition to maintain numerical stability. This is represented by:

$$\Delta t \leq \frac{\Delta \mathrm{x}}{c_{\max}} \tag{8}$$

Where C_{max} refers to maximum speed of information transfer. This preserves momentum [14].

3. Implementation:

3.1 Streaming:

In LBM, the streaming step is used to propagate the fluid particles in the lattice by updating the PDF according to the respective velocity vectors that is defined in the D2Q9 model. This is very important as this considers all the directions including rest position for the particle movement. This is realized using array manipulation operations. Numpy's '**np.roll**' function is utilized for efficient roll(circular shifting) of the PDF values in a mesh grid of size N_x and N_y. This automatically adheres to periodic boundary conditions.

Pseudocode 1: Streaming Function

```
Input – PDF f and velocity vector c
Output – streamed f in time $\Delta t$
FUNCTION stream(f: array, c:array)
        FOR i from 1 to 8
                f[i] = roll (f[i], shift = c[i], axis = (0,1) )
                END FOR
        RETURN f
END FUNCTION
```

Table 3.1 displays the implementation of the streaming functionality for N_x, Ny mesh.

3.2 Collision:

In the collision step, density and velocity at each lattice locations are calculated to represent the properties of mass distribution and momentum conservation. Density is calculated by summing all the PDFs using **'calculate_density(f)'** function where **f** represents the PDFs. Velocity(u) is calculated by multiplying the PDFs with their respective directional velocities (c) and normalizing it by density using '**calculate_velocity(f, density)**' function. This calculation helps in transitioning towards the equilibrium mentioned in BGK model.

$$calculate\ \mathrm{Density}(f) = np.sum(f, axis = 0)$$

$$calculateVelocity(f, density) = np.dot(f.T, c).T/density$$

Pseudocode 2: Collision Function

```
Input – PDF f,  density array, velocity vector c, relaxation constant, weights (W)
Output – streamed f in time $\Delta t$
FUNCTION calculate_collision(f: array, density: array, velocity: array, rl: float)
        f_eq = calculate_equilibrium(density, velocity)
        f = f - rl * (f - f_eq)
        RETURN f, density, velocity
END FUNCTION

FUNCTION calculate_equilibrium(density: array, velocity: array)
```

```
        local_velocity_avg = velocity[0, :, :]^2 + velocity[1, :, :]^2
        cu = DOT_PRODUCT (TRANSPOSE(velocity), TRANSPOSE(c))
        velocity_2 = cu^2
        f_eq = TRANSPOSE((1 + 3 * cu + 4.5 * velocity_2 - 1.5 * local_velocity_avg) * density) * W
        RETURN f_eq
END FUNCTION
```

Table 3.2 displays the implementation of the Collision functionality for N_x, Ny mesh.

3.3 Shear Wave Decay:

For studying the shear wave decay in a fluid domain, the attenuation of sinusoidal velocity perturbation is focused on. This provides information on the fluid's kinematic viscosity (ν). The simulation initializes the fluid density uniformly for (N_x ,N_y) sized lattice. Then an initial sinusoidal velocity is introduced along 1 dimension to generate a shear wave.

The theoretical instantaneous velocity decay is calculated using the following equation:

$$y = e^{\left(\frac{-\nu 2\pi}{N_y}\right)^2} \tag{9}$$

The decay of perturbation is governed by this equation and the amplitude of the sine wave decreases over time due to viscosity. Kinematic viscosity is calculated as $\frac{1}{3}\left(\frac{1}{\omega} - \frac{1}{2}\right)$. The max velocity value of the theoretical and simulated shear wave is stored at each time step for comparison.

Pseudocode 3: Shear wave simulation algorithm

```
Input – PDF f,  density array, velocity vector c, relaxation constant, omega
Output - simulated_viscosity, analytical_viscosity, PDF f
FUNCTION shear_wave_simulation(x_dim = 100, y_dim = 100, omega = 0.5, ep = 0.05, save_every = 20, steps = 1000)
   FUNCTION instant_theoretical_velocity(viscosity)
      RETURN exp(-viscosity * (2 * pi / y_dim) ** 2)
   END FUNCTION
   FUNCTION decay_perturbation(time, viscosity)
      RETURN ep * exp(-viscosity * (2 * pi / y_dim) ** 2 * time)
   END FUNCTION
   // Initialize density and velocity fields
   velocity[1, :, :] = ep * sin(2 * pi / y_dim * range(y_dim))
   // Compute initial equilibrium distribution function
   f = calculate_equilibrium(density, velocity)
   // Main simulation loop
   FOR step FROM 1 TO steps
```

```
        f = stream(f) // Perform streaming
        f, density, velocity = calculate_collision (f, density, velocity, omega) //
    Perform collision
        kinematic_viscosity = 1 / 3 * (1 / omega - 0.5) // Calculate kinematic viscosity
        // Update velocity based on theoretical decay
        y_val = instant_theoretical_velocity(kinematic_viscosity)
        y_val *= vn[1, x_dim // 4, :]
        vn[1, x_dim // 4, :] = y_val
        // Store the maximum theoretical velocity for the current step
        theoretical_velocity.APPEND(y_val.max())
        // Optional: Save simulation state at intervals
        IF step MOD save_every == 0
            // Save or process simulation state as required
        END IF
      END FOR
      // Estimate simulated viscosity through curve fitting
      simulated_viscosity = curve_fit(decay_perturbation, time_data, vel_list)[0]
      analytical_viscosity = kinematic_viscosity // Directly calculated from omega
      // Return the simulated viscosity, analytical viscosity, and final state
      RETURN simulated_viscosity, analytical_viscosity, f
    END FUNCTION
```

Table 3.3 displays the implementation of the Shear wave simulation for N_x=100, Ny=100 mesh grid.

3.4 Couette Flow:

In Couette flow analysis, a viscous fluid is confined between two parallel plates, and the laminar flow characteristics of the fluid which is developed due to one plate in constant motion is analysed. This is useful to determine fluid mechanics and shear stress in LBMs. In the current setup the top lid is constantly moving with uniform velocity. The bottom plate is fixed at a point. Here bounce-back conditions are used to simulate the no-slip boundary conditions at the walls.

The bounce-back condition absolutely reflects the PDF at the boundary wall to simulate interaction with the wall. Here, there is a clear distinction between ‘wet’ nodes located inside the fluid and ‘dry’ nodes located inside the wall.

In this current setup, different boundary conditions are applied separately to two walls as one is moving relative to other:

- At the dry node of stationary bottom wall, PDFs are directly reflected to enforce the no slip criteria (fluid particles having 0 velocity at the wall node as they are in contact)
- At the top lid PDFs are reflected but a component of velocity get added to the wet node as the lid is motion and this creates a shear flow.

This bounce back is realized using the following pseudocode:

Pseudocode 4: Bounce Back Function

```
Input – PDF $f$,  Lid velocity
Output – streamed $f$ in time $\Delta t$
FUNCTION bounce_back(f: array, lid_vel: float)
    //Bounce-back on the stationary bottom wall
    f[2, 1:-1, 1] = f[4, 1:-1, 0]
    f[5, 1:-1, 1] = f[7, 1:-1, 0]
    f[6, 1:-1, 1] = f[8, 1:-1, 0]

    //Bounce-back on the moving top wall with lid velocity adjustment
    f[4, 1:-1, -2] = f[2, 1:-1, -1]
    f[7, 1:-1, -2] = f[5, 1:-1, -1] - 1/6 * lid_vel
    f[8, 1:-1, -2] = f[6, 1:-1, -1] + 1/6 * lid_vel

    RETURN f
END FUNCTION
```

Table 3.4 displays the implementation of the bounce back condition.

3.5 Poiseuille flow:

This flow model depicts the fluid flow that occurs due to pressure gradient between two non-moving parallel plates. The assumption here is that it is a infinitely long channel along with laminar flow condition of the fluid. This disregards any impact from the side walls of the channel. In these ideal conditions, the velocity profile tends to be parabolic. This shape is derived from the Navier -Stokes equation. [15]

Mathematically the profile of Velocity u(y) is calculated by Hagen – Poiseuille equation:

$$u(y) = \frac{1}{2\mu}\left(-\frac{\Delta P}{L}\right)(h^2 - y^2) \tag{10}$$

Here *u(y)* denotes the velocity of fluid at distance of *y* from the center. Viscosity is determined by μ that is dynamic in nature. ΔP is the pressure difference between two ends of channel, L denotes length of the channel. *h* denotes the channel half height.

Pseudocode 5:Modified Bounce Back Function for Poiseuille flow

```
Input – PDF f,  Lid velocity
Output – streamed f  in time  $\Delta t$
FUNCTION bounce_back(f: array, lid_vel: float)
    // bounce-back on the bottom wall
    f[2, 1:-1, 1] = f[4, 1:-1, 0]
    f[5, 1:-1, 1] = f[7, 1:-1, 0]
    f[6, 1:-1, 1] = f[8, 1:-1, 0]

    //bounce-back on the top wall with adjustments for lid velocity
    f[4, 1:-1, -2] = f[2, 1:-1, -1]
    f[7, 1:-1, -2] = f[5, 1:-1, -1] - 1/6 * lid_velocity
    f[8, 1:-1, -2] = f[6, 1:-1, -1] + 1/6 * lid_velocity

    RETURN f
END FUNCTION
```

Table 3.5 displays the implementation of the bounce back condition for Poiseuille flow.

3.6 Sliding Lid:

In Sliding Lid simulation scenario, vortex behavior in a fluid is examined. This is achieved by driving the lid (top boundary) moving at constant velocity u_{lid}. No-slipping criteria is adhered at the stationary walls that can be written as u = (u_{lid} , 0) for the top lid and u = (0,0) for the stationary walls. Using LBM for simulation and utilizing functions including collision, streaming, modified bounce back conditions
, accurate fluid dynamics can be calculated and visualized. The Sliding lid simulation performs as a benchmark for validating and evaluating the dynamics of fluid as well as fluid motion due to boundary conditions.

Pseudocode 6: Modified Bounce Back Function for Sliding Lid

```
Input – PDF f, Lid velocity
Output – Updated distribution function f after bounce-back

FUNCTION optimized_bounce_back(f: array, lid_v: float)
    // Bounce-back on the bottom wall
    f[[1, 5, 8], 1, 1:-1] = f[[3, 7, 6], 0, 1:-1]

    // Bounce-back on the top wall
    f[[3, 6, 7], -2, 1:-1] = f[[1, 8, 5], -1, 1:-1]

    // Bounce-back on the left and right walls
    f[[2, 5, 6], 1:-1, 1] = f[[4, 7, 8], 1:-1, 0]

    // Adjustments for lid-driven cavity flow at the top wall
    d_w = 2.0 * f[[2, 5, 6], 1:-1, -1].sum(axis=0) + f[[0, 1, 3], 1:-1, -1].sum(axis=0)
    f[4, 1:-1, -2] = f[2, 1:-1, -1]
    f[7, 1:-1, -2] = f[5, 1:-1, -1] - lid_v * d_w / 6
    f[8, 1:-1, -2] = f[6, 1:-1, -1] + lid_v * d_w / 6

    RETURN f
END FUNCTION
```

Table 3.6 displays the implementation of the bounce back condition for Sliding Lid.

3.7 Using MPI (Message Passing Interface) for Parallel execution:

Parallel computing plays a very critical role in the CFD simulations. An important approach for parallel computing is by using MPI, that enables communication and data exchange between processors.
MPI executes parallel processing by dividing the computational domain into multiple subdomains, which each is managed by a separate processor. This is called domain decomposition which is very important for load balancing and efficient execution.

MPI and Domain Decomposition: Division of computational domain space into multiple subdomains that is assigned to individual processors. There should be minimal overlap to decrease communication overhead and optimize parallel efficiency.
Implementation Highlights: Several key steps are involved in implementing MPI in LBM. Each processor performs operations that include collision, streaming, boundary conditions locally. Now this boundary information is exchanged to neighboring subdomains via MPI communication protocols. This ensures continuity.

Ghost Nodes: Ghost Nodes introduce buffer zones at the borders of the subdomain. These nodes receive data from the boundaries of the adjacent subdomains. Often the communication pattern involves the usage of non-blocking send and receive.

Performance Considerations: The efficiency for MPI based LBM execution is based on communication to computation ratio. Optimal performance is achieved when the domain decomposition is perfect enough that overhead for inter-processor communication is less than the computational workload.

Pseudocode 7: Modified Bounce Back Function for MPI

```
Input – PDF $f$,  Lid velocity
```

```
Output – Updated distribution function f after bounce-back

FUNCTION MPI_bounceback(f, mpi_s)
    // Apply bounce-back conditions for solid boundaries using non-equilibrium extrapolation

    IF mpi_s["boundary"]["right"] IS true THEN
        // Reflect particles hitting the right wall
        f[[3, 6, 7], -2, :] = f[[1, 8, 5], -1, :]

    IF mpi_s["boundary"]["left"] IS true THEN
        // Reflect particles hitting the left wall
        f[[1, 5, 8], 1, :] = f[[3, 7, 6], 0, :]

    IF mpi_s["boundary"]["bottom"] IS true THEN
        // Reflect particles hitting the bottom wall
        f[2, :, 1] = f[4, :, 0]
        // For diagonal directions
        f[[5, 6], :, 1] = f[[7, 8], :, 0]

    IF mpi_s["boundary"]["top"] IS true THEN
        // Reflect particles hitting the top wall
        f[4, :, -2] = f[2, :, -1]
        // For diagonal directions, adjust with lid velocity
        f[[7, 8], :, -2] = Adjust with lid velocity
    RETURN f
END FUNCTION
```

Table 3.7.1 displays the implementation of the bounce back condition for MPI Sliding Lid.

Pseudocode 8: Communication Function for MPI

```
Input – PDF f, comm, mpi_s
Output – Updated distribution function f

FUNCTION comunicate(f, comm, mpi_s)
    // Communicate boundary rows/columns with adjacent subdomains

    // Right boundary
    IF NOT mpi_s["boundary"]["right"] THEN
        Prepare send buffer for the right boundary
        Send/Receive operation with the right neighbor
        Update the rightmost column of f from the received buffer

    // Left boundary
    IF NOT mpi_s["boundary"]["left"] THEN
        Prepare send buffer for the left boundary
        Send/Receive operation with the left neighbor
        Update the leftmost column of f from the received buffer
    // Bottom boundary
    IF NOT mpi_s["boundary"]["bottom"] THEN
        Prepare send buffer for the bottom boundary
        Send/Receive operation with the bottom neighbor
        Update the bottom row of f from the received buffer

    // Top boundary
    IF NOT mpi_s["boundary"]["top"] THEN
        Prepare send buffer for the top boundary
        Send/Receive operation with the top neighbor
        Update the top row of f from the received buffer

    RETURN f
END FUNCTION
```

Table 3.7.2 displays the implementation of the communication for MPI Sliding Lid.

4. Numerical Results and Validation:

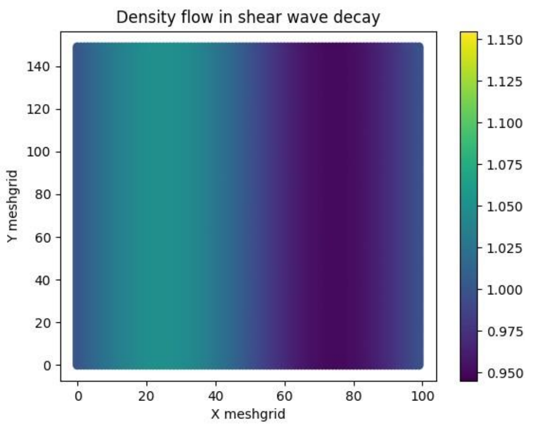
4.1 Shear wave decay:

In LBM there is an observable attenuation of velocity profile and dampening due to viscosity. Here we are analyzing the progressive decrease in density and velocity over time. There is also analysis on the change in kinematic viscosity represented by ω. The initial conditions create a minor sinusoidal shaped disturbance to the velocity and the temporal evolution of the wave decay is observed in this simulation.
The theoretical analysis for this shear wave is derived from the Navier-Stokes equation for the fluids that are incompressible. Here, we do no consider the convective and pressure gradient terms in the equation and focus solely on the viscous force. This simplification provides an expression that predicts the decay (exponential) of the velocity profile(sinusoidal) by the following equation:

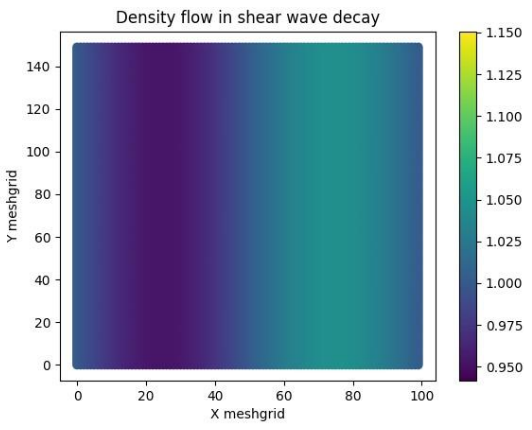
$$v(x,t) = v_0 \exp\left(-\nu\left(\frac{2\pi}{L}\right)^2 t\right)\sin\left(\frac{2\pi x}{L}\right) \quad (11)$$

Here ν refers to kinematic viscosity, L denotes channel length, t denotes time.

Density Visualization:



(a) Time =0



(b) Time= 90

Table 4.1.1 displays the flow of density in the mesh (x_dim = 100, y_dim = 150) with ω = 1). There is an observable shift in the particle density from left to right.

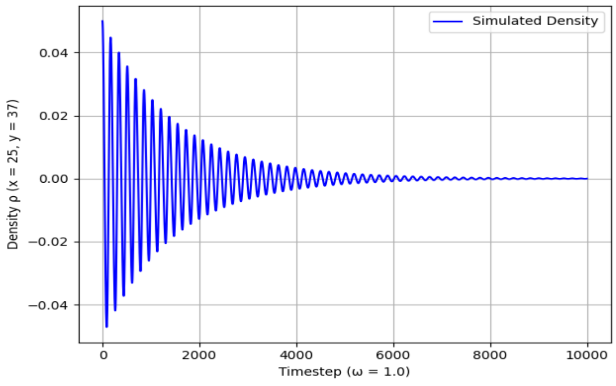
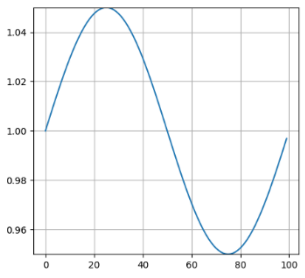
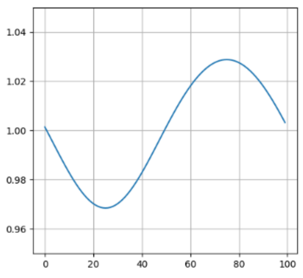
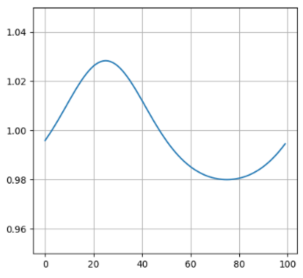
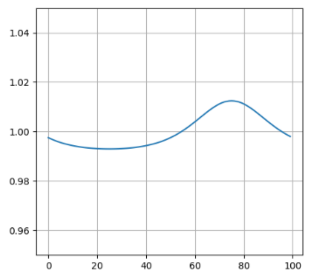


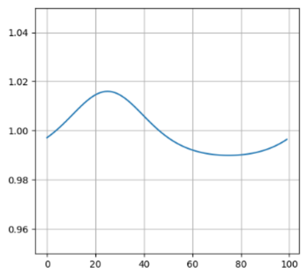
Table 4.1.2 displays the density decay of one element located at (50,75) in the lattice for time range of 10,000 steps with mesh size (x_dim = 100, y_dim = 150).


(a) T=0


(b) T= 110


(c) T=320


(d) T =570


(e) T=1000

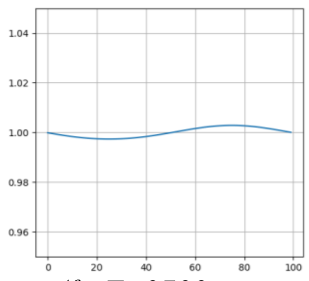
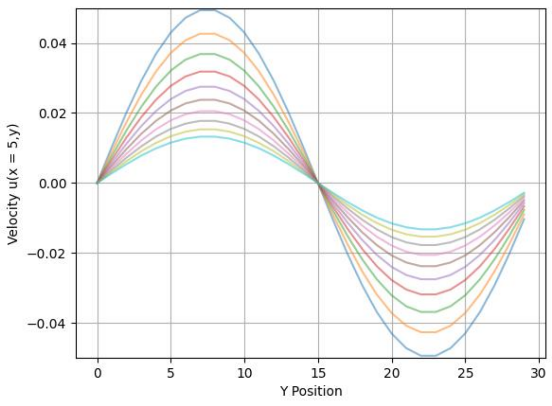
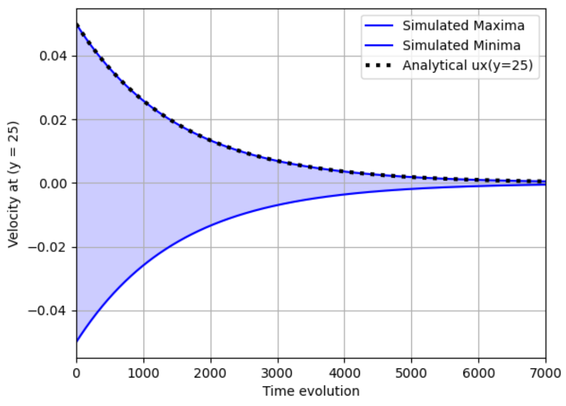

(f) T=3700

Table 4.1.3 displays the decay of density for position y = 37 across the mesh of size (x_dim = 100, y_dim = 150).

Velocity Visualization:



(a) Velocity decay of y for x = 5



(b) Velocity decay w.r.t time for 7000 steps

Table 4.1.4 (a) displays the velocity decay for y keeping x = 5 for mesh size (x_lim = 20, y_lim = 30). (b) displays the maximum decay of velocity vs time for 7000 steps for a mesh size of (x_dim = 100, y_dim = 150).

Kinematic Viscosity:

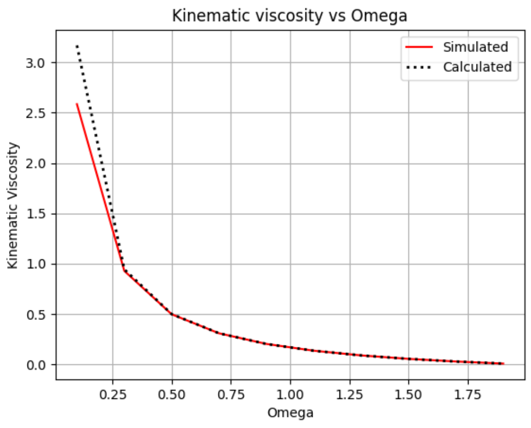


Table 4.1.5 displays the kinematic viscosity decay with time. Analytical values are calculated is a function of Omega. It is observed that the simulated result obtained from implementation of Pseudocode 3 closely follows the theoretical calculation.

4.2 Couette Flow:

Following the Pseudocode 4 bounce back along with streaming and collision, the simulation runs for 4000 steps with mesh grid size of (x_dim = 100, y_dim = 50)

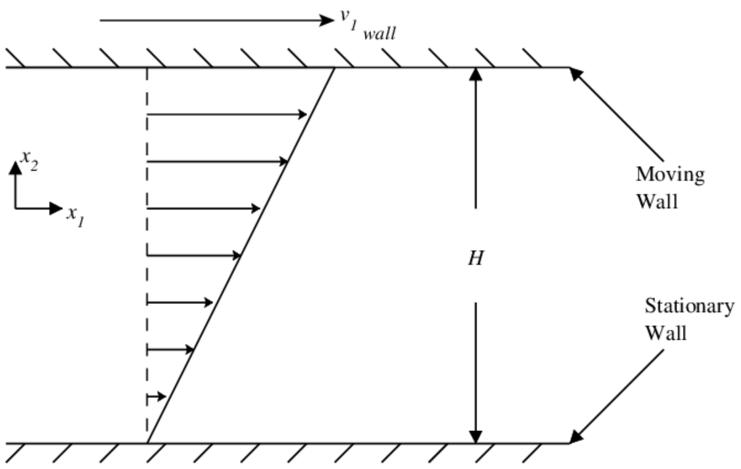
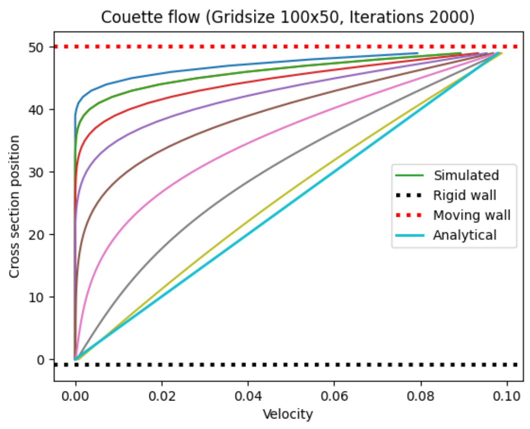
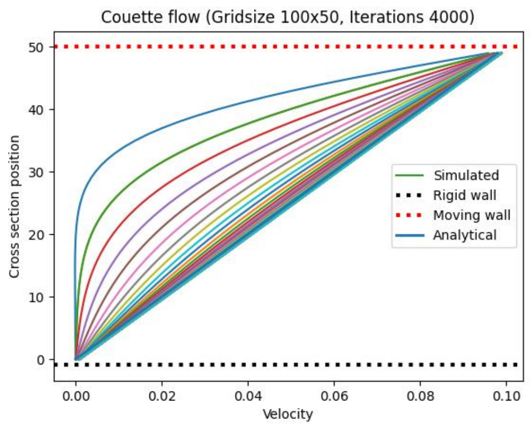


Table 4.2.1 Analytical Couette flow

Table 4.2.1 describes the theoretical Couette flow with just a straight line as the gradient when the top lid is moving and the bottom is fixed [16].



(a) *Flow at 2000 steps*

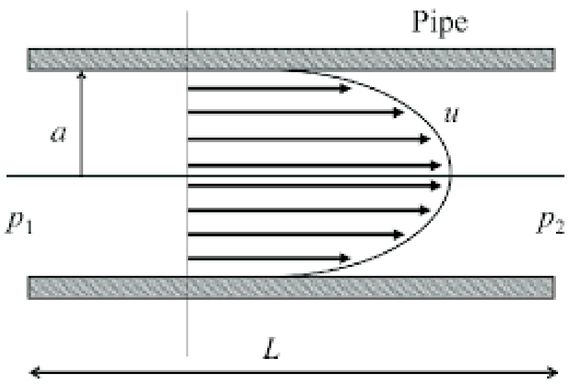


(b) *Flow at 4000 steps*

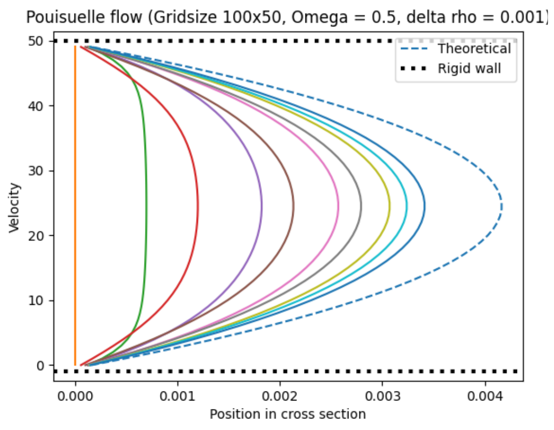
Table 4.2.2 displays the profile of the fluid flow after (a) 2000 steps and (b) 4000 steps.

4.3 Poiseuille Flow:

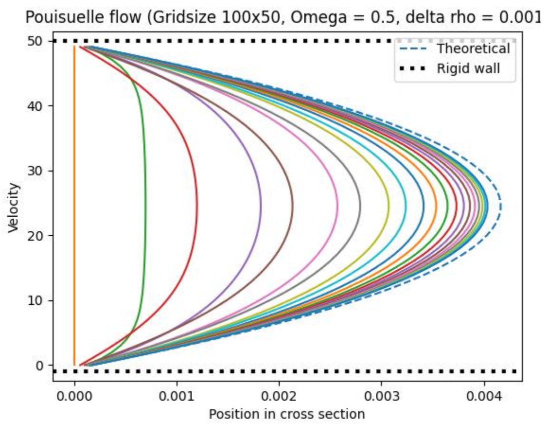
In simulation of Poiseuille LBM simulation, the behavior of fluid was observed when it is subjected to uniform pressure gradient along with no slip conditions for both the walls with the boundary conditions mentioned in Pseudocode 5. The simulation was conducted with varying time steps from 1000 to 6000 with mesh grid size of (x_dim=100, y_dim=50).



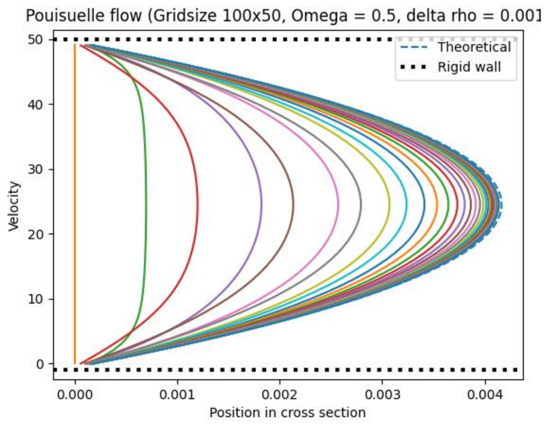
(a) Theoretical Poiseuille flow [17]



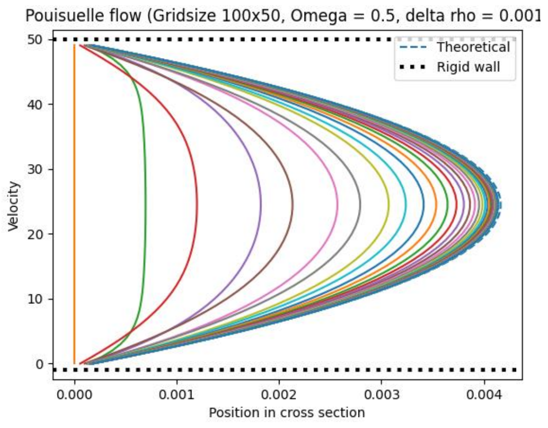
(b) T = 1000



(c) T = 2000



(d) T = 4000

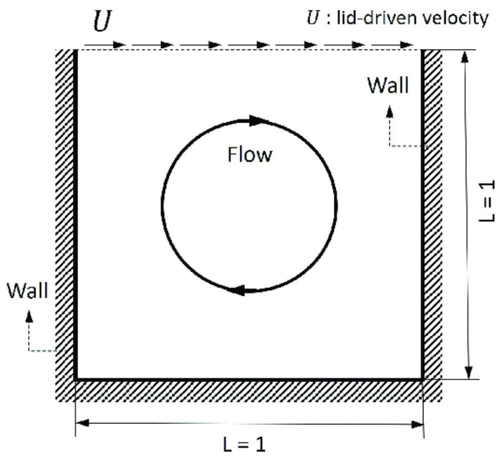


(e) T = 6000

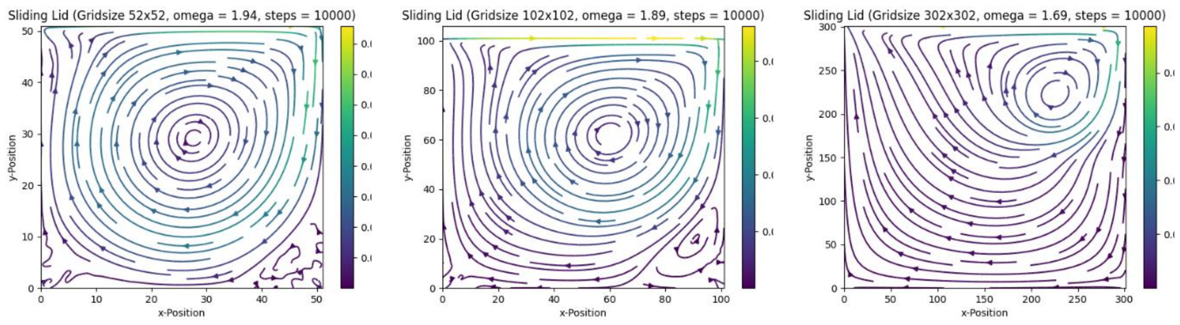
Table 4.3.1 (a) Displays the theoretical Poiseuille flow output. (b) simulated flow profile when T=1000 steps. (c) T=2000 steps (d) T=6000 steps mesh grid size is(x_dim= 100, y_dim= 50).

4.4 Sliding Lid:

In this section, LBM model's fidelity was validated for simulating moving fluid dynamics. The simulation was conducted with a 2D square box with lid constantly in motion with velocity 0.1. Bounce back conditions from pseudocode 6 were utilized. The results obtained after a large range of time steps and with varying parameters closely resembled the theoretical results.

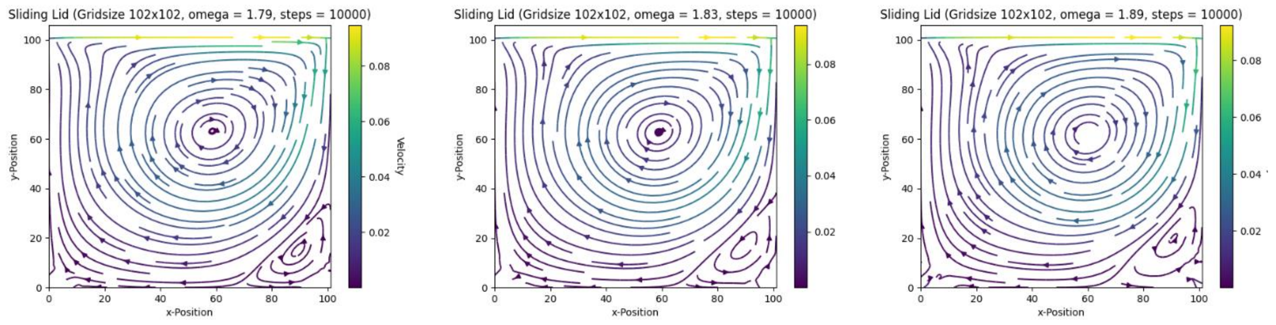


(a) Theoretical result [18]



(b) *(c)* *(d)*

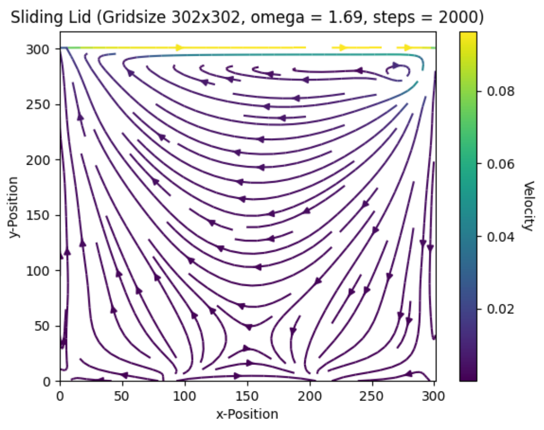
Table 4.4.1(a) provides representation of sliding lid expected outcome. (b),(c),(d) displays sliding lid simulations for different grid sizes (52 X 52), (102 X 102), (302 X 302) with 10,000-time steps.



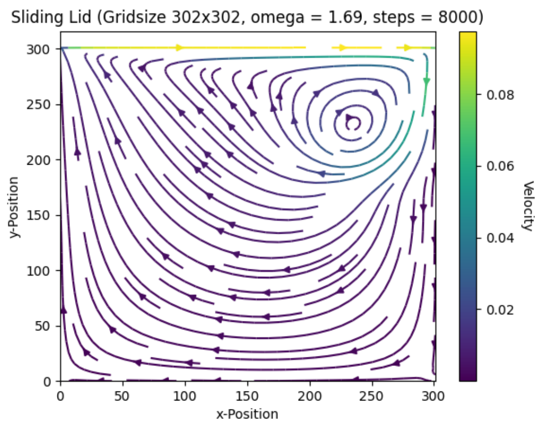
(a) *(b)* *(c)*

Table 4.4.2(a)-(c) displays simulations with relaxation time = 0.02,0.015,0.01 calculated by $Rel\text{ Time} = \frac{Length\ x\ \text{Lid Velocity}}{Re}$

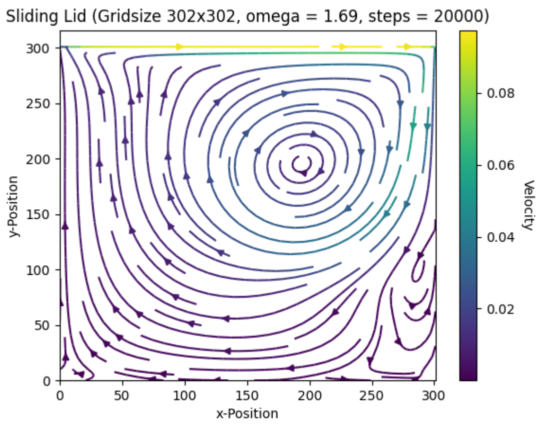
with grid size (102 X 102) and 10,000-time steps.



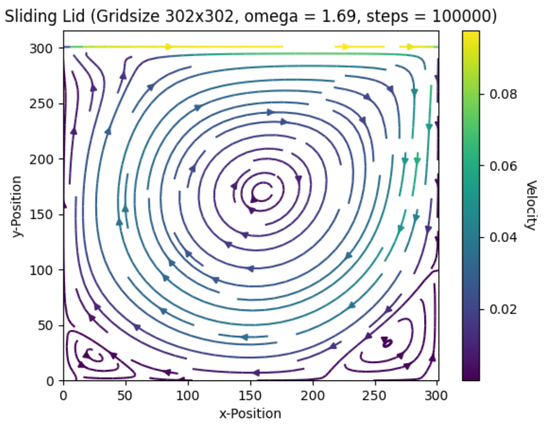
(a)



(b)



(c)



(d)

Table 4.4.3(a)-(e) depicts the progression of the simulation with grid size (302 X 302) at different time steps (2000,8000,20000,100000)

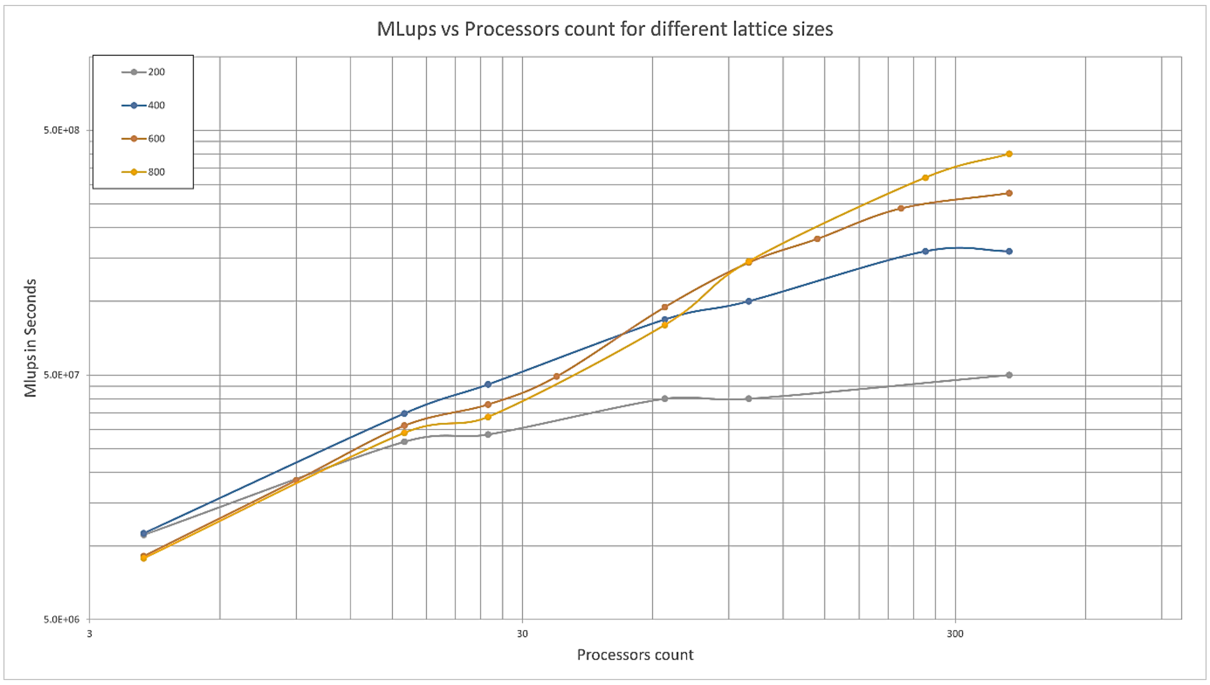


Table 4.4.4 denotes the MLUps vs number of processes for various grid sizes (200,400,600,800). This is repeated for different counts of processors. Amdahl's formula is followed for speedup.

5. Conclusion:

The comprehensive study of LBM has showed the efficiency of this methodology for performing simulations of Fluid Dynamics. Through theoretical analysis and output validations, the advantages and capabilities of LBM have been determined.

The introduction section presented fundamental principles of fluid flow simulation and the importance of LBM which is famous for its numerical stability and flexibility.

In subsequent discussion of theoretical concepts Boltzmann Transport Equation was discussed along with its division into streaming and collision processes with 2D lattice (D2Q9) structure.

The core section of experimentation discussed the strategies to implement various simulations that include Couette flow, Poiseuille flow, shear wave decay and Sliding Lid problem along with brief introduction to parallel computing using Message Passing Interface (MPI). This highlighted the robustness of LBM and efficiency increase due to parallel computing.

The Presentation of numerical results along with validation from theoretical modelling, showed coherence in LBM and highlighted its potential for scaling to more complex simulations.

6. References:

[1] Saleh S. Baakeem, Saleh A. Bawazeer, and Abdulmajeed A. Mohamad, "Comparison and evaluation of Shan–Chen model and most commonly used equations of state in multiphase lattice Boltzmann method," *International Journal of Multiphase Flow*, vol. 128, p. 103290, 2020.

[2] Zhen Chen, Chang Shu, and Danielle S. Tan, "Immersed boundary-simplified lattice Boltzmann method for incompressible viscous flows," *Physics of Fluids*, vol. 30, p. 53601, 2018.

[3] Dan Pu, Lin-fang Shen, Zhiliang Wang, Miao Li, Zhenquan Li, and Pengyu Wang, "Multiphase Seepage Flow Characteristics of Micro-Fractures in Subsurface Reservoirs Using the Lattice Boltzmann Method," *Volume 9: Offshore Geotechnics; Petroleum Technology*, 2023.

[4] Meratun Junnut Anee, Sadia Siddiqa, Md. Farhad Hasan, and Md. Mamun Molla, "Lattice Boltzmann simulation of natural convection of ethylene glycol-alumina nanofluid in a C-shaped enclosure with MFD viscosity through a parallel computing platform and quantitative parametric assessment," *Physica Scripta*, vol. 98, 2023.

[5] K. J. Petersen and Joshua R. Brinkerhoff, "On the lattice Boltzmann method and its application to turbulent, multiphase flows of various fluids including cryogens: A review," *Physics of Fluids*, vol. 33, p. 41302, 2021.

[6] Changsheng Zhu, Jieqiong Liu, Li Feng, and Xin Deng, "Research on the simulation of PF-LBM model based on MPI+CUDA mixed granularity parallel," *AIP Advances*, vol. 8, p. 65017, 2018.

[7] Pedro Valero-Lara and Johan Jansson, "LBM-HPC - An Open-Source Tool for Fluid Simulations. Case Study: Unified Parallel C (UPC-PGAS)," *2015 IEEE International Conference on Cluster Computing*, pp. 318–321, 2015.

[8] Liu Yong-chang, "Lattice Boltzmann model for two-dimensional shallow water flows realizing parallel computing on GPU," *Chinese Journal of Hydrodynamics*, 2011.

[9] S. Succi, *The lattice Boltzmann equation: for fluid dynamics and beyond*: Oxford university press, 2001.

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

[11] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems," *Physical review*, vol. 94, no. 3, p. 511, 1954.

[12] H. Hamedi and M. H. Rahimian, "Numerical Simulation of Non-Newtonian Pseudo-Plastic Fluid in a Micro-Channel Using the Lattice Boltzmann Method," *WJM*, vol. 01, no. 05, pp. 231–242, 2011, doi: 10.4236/wjm.2011.15029.

[13] L. P. a. A. Greiner, ., *"Hpc with python: An mpi-parallel implementation of," [Online].*

[14] Xiang Zhao, Zhen Chen, Liming Yang, Ningyu Liu, and Chang Shu, "Efficient boundary condition-enforced immersed boundary method for incompressible flows with moving boundaries," *Journal of Computational Physics*, vol. 441, p. 110425, 2021, doi: 10.1016/j.jcp.2021.110425.

[15] Tytti Saksa, "Navier-Stokes Equations," *Fundamentals of Ship Hydrodynamics*, 2019.

[16] S. Mizzi, D. Emerson, R. Barber, and J. Reese, "MODELING LOW KNUDSEN NUMBER TRANSITION FLOWS USING A COMPUTATIONALLY EFFICIENT CONTINUUM-BASED METHODOLOGY," in 2006.

[17] M. Lawrence-Brown, K. Liffman, J. Semmens, and I. Sutalo, "Vascular arterial haemodynamics," *Mechanisms of Vascular Disease: A Reference Book for Vascular Specialists*, pp. 153–176, 2011, doi: 10.1017/UPO9781922064004.009.

[18] T. Huang and H. Lim, "Simulation of Lid-Driven Cavity Flow with Internal Circular Obstacles," *Applied Sciences*, vol. 10, p. 4583, 2020, doi: 10.3390/app10134583.