High Performance Computing :
Computational Fluid Dynamics with Python

Dwaipayan Roy Chowdhury

5360085

dwaipayan0502@gmail.com

May 30, 2022

Table of Contents

1

Introduction

With engineering development, we have seen advancements in scientific methodologies in performing experiments and those experiments are often expensive to perform without prior prediction on the behaviour of such framework. In today's simulation world, we always try to optimize and maximize the extent of our computation with a linear Big O thus enhancing the accuracy and performance. Here, we talk about basic fundamentals of fluid flow and its simulation which helps us decipher the maths behind such algorithms. This popular LBM method was first proposed by Shan and Chen where mean field interactions for multiphase and multicomponent fluid flow was demonstrated. [1]

We have chosen to use the Lattice-Boltzmann simulation framework as the fundamental methodology as it provides outstanding numerical stability with very high constitutive flexibility. It is also accounted for higher accuracy with continuous numerical checks amongst Macro and mesoscaling of fluid simulations. [2]

The intricate basics of lattice Boltzmann deals with a pack of fluid particles and then considering that as a distribution function. LBM is a well-known mechanism to solve the Navier stokes equation in a mesoscopic scale. Here, we consider a lattice automata as a base to perform the fluid flow. This methodology works on particle model and mesoscopic kinematic theories. We compute several fluid properties (e.g. kinematic viscosity) and physical states/moments by iterating the lattice positions across forward direction. The main benefits of using LBM framework are:

1. Simple algorithm: With comparison to conventional CFD methodologies (e.g. Finite Difference Method, Finite Volume Method) LBM provides simple calculation procedure.

2. Efficient performance: Smart and constructive parallel computation and able to be incorporated in LBM for robust datasets. [3] [4] [2]

With a phenomenal track of success rate, LBM sought researchers from varied verticals. Thus we here talk about this method in follow subroutines:

1. Theory and mathematical aspects: LBM has a couple of underlying mathematical theorems associated. We will discuss about them one by one.

2. Implementation: We provide small pseudocodes to present a smooth maths versus implementation synchronization.

3. Graphical simulation results: Provide with the plots to visualize the results.

2

Lattice Boltzmann method

In this part, we will describe how the equations of LBM are being used in computation. The backbone of LBM is **Boltzmann Transport equation (BTE)**. The non-linear complexity of Boltzmann transport equation has created a resort of further simplification of linearized Boltzmann equation. [5]

2.1 The Boltzmann Transport equation (BTE)

The basic mathematical structure and principles of BTE comes from exploding a particle probability function $f(x, v, t)$ where x is the instantaneous position of the pack of particles is and v be the microscopic velocity. The transport phenomenon is formulated by the principle of relaxation of probability distribution to the Maxwell velocity distribution function. The formula for BTE is:

$$\frac{df(\boldsymbol{x},\boldsymbol{v},t)}{dt} = -\frac{f(\boldsymbol{x},\boldsymbol{v},t) - f^{\mathrm{eq}}(\boldsymbol{v};\rho(\boldsymbol{x},t),\boldsymbol{u}(\boldsymbol{x},t),T(\boldsymbol{x},t))}{\tau} \tag{1}$$

where, f^{eq} is the state of equilibrium, $T(x, t)$ being the temperature at position x of the time step t, τ is the relaxation time, $u(x, t)$ is the macroscopic velocity and $\rho(\boldsymbol{x}, t)$ is the macroscopic density. The above mentioned BTE equation consists of 2 main subroutines streaming and collision. [6] We here concentrate on 2-Dimensional lattice (D2Q9) where the density or the positions can be distributed along a mesh of 2D automata and the velocity is distributed along 9 lattice points.

- We create an equidistant 2D mesh for the particle position and discretize $f(x, t)$ across the mesh.
- For velocity, we create a 3 x 3 space lattice and create a direction coefficient c_i which are called velocity sets. This has a 2D weight and axis denotations are being done by

$$\mathbf{c} = \begin{pmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{pmatrix}^T.$$

where each column denotes the velocity occupancy of each particles starting from 1 to 9. That's how we perform special and velocity discretization and velocity is projected into lattice grid by spatial superimposition of 9 lattice points. Thus we conclude from this part that the Probability density function (PDF) will move to the corresponding neighbour lattice from position r to $r + c_i\Delta t$ position at the next time step $t + \Delta t$. [7]

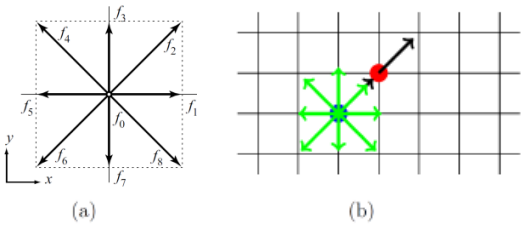


Figure 2.1: (a) The discretization of velocity space to spatial D2Q9 lattice. (b) The propagation stream to the next lattice. [8] [7]

Since, we have discretized the positions into 2D meshes using lattice points inside the automata and the velocity vectors into 2D 3x3 positions, we can now apply the discretization theorem (Lattice Boltzmann theorem) based on the mentioned mesh concept. The LBM is divided into 2 main subroutines:

2.1.1 Streaming

The left hand side of Lattice Boltzmann equation provides a fictive particles performing propagation over a discrete mesh of automata. The 2D orthogonal 9 velocity model is used to perform propagation over time steps. The initial position x is updated to the next adjacent

following lattice point $x + c_i\Delta t$ in $t + \Delta t$ time. Hence, we conclude by saying that the time derivative in Boltzmann transport equation ($\frac{\delta f(x,t)}{\delta t}$) in (1) is being discretized here and is modelled under streaming operator. [8]

Collision

The right hand side of lattice Boltzmann equation depicts the collision phase of the streaming thus completing a tenure of fluid propagation cycle. Physically, this represents particle interaction and then followed by the reverse order scattering phenomenon. The Boltzmann-Transport equation can be rewritten as

$$\frac{d}{dt} f(\mathbf{r}, \mathbf{v}, t) = -\frac{f(\mathbf{r}, \mathbf{v}, t) - f^{eq}(\mathbf{r}, \mathbf{v}, t)}{\tau} \tag{2}$$

The τ here is known as the relaxation time. In Boltzmann equation, we encounter with an advection equation (streaming) and a source term which is collision operator. The f^{eq} is mainly solved used Bhatnagar Gross Krook (BGK) method which holds an equilibrium distribution function. [7] [9]

$$f_i^{eq}(\rho(\mathbf{r}), \mathbf{u}(\mathbf{r})) = w_i \rho(\mathbf{r}) \left[1 + 3\mathbf{c}_i \cdot \mathbf{u}(\mathbf{r}) + \frac{9}{2}(\mathbf{c}_i \cdot \mathbf{u}(\mathbf{r}))^2 - \frac{3}{2}|\mathbf{u}(\mathbf{r})|^2\right] \tag{3}$$

We need to calculate the macroscopic velocity and density here to get the equilibrium term that's used for collision. For that, we perform the following two step method in solving he puzzle.

Step 1:

We sum up the PDFs for position along a kernel that fives the cumulative total of the density of molecules typically written like

$$\rho(r) = \sum_i f_i \tag{4}$$

After finding the density, we then calculate the macroscopic velocity be the formula [7]

$$\frac{\text{Sum of momentums of individual particles}}{\text{Total density (mass)}}$$

$$u(r) = \frac{1}{\rho(r)} \sum_i f_i c_i \tag{5}$$

and w_i are the corresponding weights. [7]

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

2.2 Time step update in BTE

Streaming accounts for the variation of time step which plays as a very vital role in LBM. Thus, in our paper, our time updation follows Courant-Friedrichs-Lewy inequality equation to suffice conservation of momentum. [10]-

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

3

Implementation

3.1 Streaming

In streaming, we take the probability distribution function of position and velocity and then use the roll function of python in rolling and shifting of the 9x9 velocity lattice. The velocity component is the last or the second index of the PDF. This means in a mesh grid of N_x by N_y density or position lattice for all the positions of each lattice occupied particle is rolled by once along axis 0 and once along axis 1. After the roll, we shift the individual lattice by one indices. The roll and shift is the basic pillar of streaming.

$$np.roll(f([x][y][i], shift = c_i, axis = (0,1)) = f[nx][ny][i]$$

where, nx = (x + c_i[0])%X, ny = (y + c_i[0])%Y, i being the direction index for D2Q9 with c_i denoting the velocity vector for D2Q9 lattice.

ALGORITHM 1: ALGORITHM TO STREAMING

Input parameters: Probability density function f_x and the 9 velocity axis c_i
Output: f_x streamed for time step Δt

```
function STREAMING(f)
fnext = np.zero(f*)
for i=0,1,.....,8
fnext[:,:,i] = np.roll(f*[:,:,i], shift = ci, axis = (0,1)
return fnext
```

Table 3.1 shows the algorithm for streaming (N_x = 100, N_y = 50)

3.2 Collision:

Now for collision operation we calculate the updated density and velocity of the particles and then calculate the equilibrium from the imperial formula. To find the density we use density = np.sum(f, axis =1) and to calculate the velocity we use nativity np.dot((f,c)/(density[:,:,none]). These are basically the macroscopic velocity and density of the fluid which we need in the calculation of the probability density function. In calculating the equilibrium PDF, we need the weights of the velocity that will decide the chance of a particle to move along a specific velocity axis or direction. Then we implement the imperial formula [3] to compute the equilibrium PDF.

ALGORITHM 2: ALGORITHM TO COLLISION

Input parameters: Probability density function f_x, density $f_{density}$ and velocity $f_{velocity}$
Output: f_x collision after time step Δt

```
function COLLISION(f)
feq = equilibrium(ρ,u,w)
fnext += 1/τ*(feq - fnext)
return fnext
function equilibrium(ρ = ρ(t), u = u(t), w = np.array([4/9,..]))
updates velocity u = np.dot(f, ci)
updates density ρ = np.sum(f, axis = -1)
feq = w * ρ* (1 + 3 * u + 4.5 * (u) ** 2)-1.5 * u
return feq
```

Table 3.2 shows the algorithm for collision (N_x = 100, N_y = 50)

3.3 Shear wave decay:

In shear wave decay, our main aim to calculate the kinematic viscosity decay with time as well as the decay in intensity of velocity and density across grid for a defined position. Also, we have presented the velocity across time graph to show the gradual sinusoidal decay of velocity with time. Algorithm[3] shows the mathematical implementation in the software. The inputs for streaming and collision are constant for any set of experiments we perform. Here, we have initialized the velocity and density to 0 in the beginning and then provided a sinusoidal variation to the velocity along y direction. Our goal is to see if the amplitude of this sinusoidal function of velocity decays with time.

ALGORITHM 3: ALGORITHM TO SHEAR WAVE DECAY

Input parameters: Probability density function f_x and the 9 velocity axis c_i , time step *step* for timescale *steps*
Output: f_x updated with density $f_x[Nx, N_y]$, velocity $f_x[:,:,[0,1]]$ after *steps*

```
function Shear_wave_decay(f)
if(experiment_type = "velocity"):
    density = np.ones(Ny, Nx)
    velocity[:,:,1] = eps * np.sin(2 * np.pi/Ny * y)
else:
    density = np.ones(Ny, Nx)
    velocity = np.zeros((Ny, Nx, 2))
get lbmclass.lbm(density, velocity) with updated streaming and collision
get lbs.gather_velocity() to update velocity.
relaxation time τ = 1/ω
plot kinematic viscosity = decay_perturbation(t, viscosity)
function decay_perturbation(t, viscosity)
    size = Ny if exp_type == velocity else Nx
    return eps * np.exp(−viscosity * (2 * np.pi/size) ** 2 * t)
```

Table 3.3 shows the algorithm for shear wave decay (N_x = 100, N_y = 50, ω = 1.0)

3.4 Couette Flow:

Couette flow can be defined as a laminar flow between two parallel plates and the fluid is essentially considered as viscous in nature. There will be a relative velocity between two plates. In this experiment, we have considered the top plate to be moving with a defined lid velocity keeping the bottom wall fixed. This leads to a overall change in fluid flow profile. We here consider a domain of our experiment and initialize the velocity and density using numpy arrays. The streaming and collision structure is same as to the shear wave experiment. Here, we additionally incorporate two features. The bounce back of fluid particles due to the presence of wall and the lid velocity that makes the top wall moving.

ALGORITHM 4: ALGORITHM TO BOUNCE BACK OF COUETTE FLOW

Input parameters: Probability density function f_x and the lid or wall velocity
Output: f_x updated with bounce back from the wall

```
function couette_bounce_back(f, lid velocity)
    f[1,:,2] = f[0,:,4] #lower wall
    f[1,:,5] = f[0,:,7] #lower wall
    f[1,:,6] = f[0,:,8] #lower wall
    f[−2,:,4] = f[−1,:,2] #top wall
    f[-2, :, 7] = f[-1, :, 5] - 1/6 *  lid_vel #top wall
    f[-2, :, 8] = f[-1, :, 6] + 1/6 *  lid_vel #top wall
```

Table 3.4 describes the algorithm that is being used in countering the bounce back from the wall

We now explain how the bounce back is carried out.

For top wall, we have the nodes 2, 5 and 6 colliding directly to the wall. Thus, a bounce back function is to be written which updates the density function by replacing the top three nodes by the bottom three nodes of the immediate next set of channels [-2]. To do so, we also have to take care about the lid velocity that affects the replacement of the nodes. Since, the replacement of node by node 7 acts opposite to the direction of lid velocity, thus we subtract the lid velocity from the flow bounce back of the 5th node. And since the replacement of 6th node by 8th node is in favour of the lid velocity direction, we add this up to the replacement. And the replacement of 2 by 4th node is undisturbed as seen in the Fig. 3.1

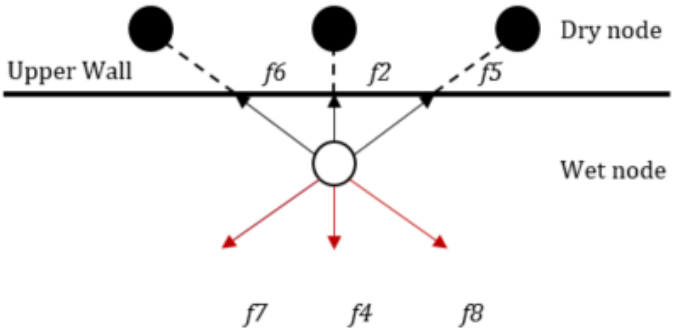


Figure 3.1 Bounceback in top wall

3.5 Poiseuille flow:

Poiseuille flow is a pressure induced fluid flow that is usually carried with two enclosed walls in upper and lower position. We assume the flow to be in a infinitely long tube where we can neglect the right and left wall bounce back. Also, the flow is considered to be laminar, hence turbulences are ignored. Thankfully we have an analytical solution for the Poiseuille flow known as Hagen-Poiseuille that's under the assumption of NSE equation for laminar flow.

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

Where, h is the diameter of the pipe, μ=ρν is the dynamic viscosity. [7]

ALGORITHM 5: ALGORITHM TO BOUNCE BACK OF POISEUILLE FLOW

Input parameters: Probability density function f_x
Output: f_x updated with bounce back from the wall
function **poiseuille_bounce_back**(f, *lid velocity*)
    $f[1,1:-1,2] = f[0,1:-1,4]$#lower wall
    $f[1,1:-1,5] = f[0,1:-1,7]$ #lower wall
    $f[1,1:-1,6] = f[0,1:-1,8]$#lower wall
    $f[-2,1:-1,4] = f[-1,1:-1,2]$ #top wall
    $f[-2,1:-1,7] = f[-1,1:-1,5]$ #top wall
    $f[-2,1:-1,8] = f[-1,1:-1,6]$ #top wall

Table 3.4 describes the algorithm that is being used in countering the bounce back from the wall

3.6 Sliding Lid:

Sliding lid, also known as Karman Vortex is a closed box experiment where we have the upper wall moving to the right with a lid velocity and the bottom wall is fixed like couette flow. The right and left wall is enclosed and thus will have periodic bounce backs from the right and left walls as well. Thus it's a closed box experiment and the algorithm for bounce back is mentioned below.

ALGORITHM 5: ALGORITHM TO BOUNCE BACK OF POISEUILLE FLOW

Input parameters: Probability density function f_x
Output: f_x updated with bounce back from the wall

```
function poiseuille_bounce_back(f, lid velocity)
    f[1,1:-1,2] = f[0,1:-1,4]#lower wall
    f[1,1:-1,5] = f[0,1:-1,7] #lower wall
    f[1,1:-1,6] = f[0,1:-1,8]#lower wall
    f[-2,1:-1,4] = f[-1,1:-1,2]  #top wall
    f[-2,1:-1,7] = f[-1,1:-1,5] - ½ * rhowall  * lid_velocity  #top wall
    f[-2,1:-1,8] = f[-1,1:-1,6] + ½ * rhowall  * lid_velocity #top wall
    f[1:-1,1,1] = f[1:-1,0,3]
    f[1:-1,1,5] = f[1:-1,0,7]
    f[1:-1,1,8] = f[1:-1,0,6]
    f[1:-1,-2,3] = f[1:-1,-1,1]
    f[1:-1,-2,6] = f[1:-1,-1,8]
    f[1:-1,-2,7] = f[1:-1,-1,5]
```

Table 3.4 describes the algorithm that is being used in countering the bounce back from the wall

3.7 Parallel Computing by Message Passing Interface

To obtain a successful MPI, we need to perform a domain decomposition. Here, we use mpi4py to achieve this in Python. It is a communication process amongst different cores of our system which helps us run bulk data-simulations at an efficient manner. To do so, we have to first divide the serialized code into fragments which require MPi and which doesn't. Here, in LBM treatment, we use a communicator function to perform communication amongst the ranks and then conditions the boundary to recognize the boundary ranks. Let's say we have P number of processes, then we distribute the processes in the form of a matrix such that $P = P_x \, x \, P_y$ where P_x, P_y ∈ Real number and P_x, P_y = argminP_x,P_y(|| P_y-P_x||). And then we divide the X length with the decomposed X core and same for Y axis. This is a very crucial aspect and has to be noticed that we don't end up having deciment values for the division. The MPI is completely unaware about the state of the decomposition. It only accepts grids those are argued by us. An interesting concept to discuss about is the ghost nodes. A ghost node is a dummy cell that providing communication between two rank and provides a outer kernel for the performing process.

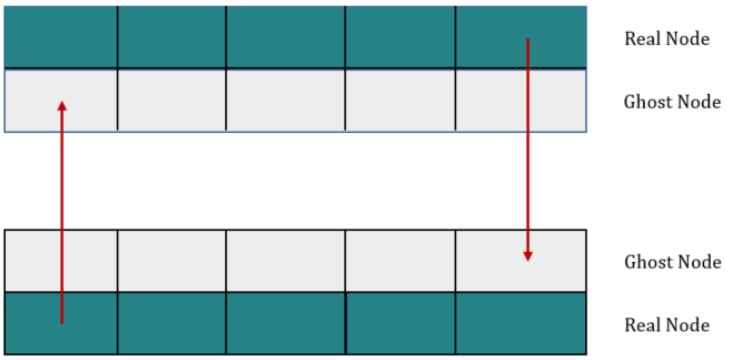


Fig 3.2: As shown in the figure, we decompose the complete grid and then add buffers (ghost nodes) to provide a kernel that for the rank boundaries. Typically required for streaming.

ALGORITHM 5: ALGORITHM FOR COMMUNICATION FUNCTION

Input parameters: Probability density function f_x
Output: f_x updated with communication done

Function mpi_communicator $(grid\ f, commCart)$
$top\_src, top\_dst = commCart.Shift(0, -1)$
$bot\_src, bot\_dst = commCart.Shift(0, +1)$
$lef\_src, lef\_dst = commCart.Shift(1, -1)$
$rig\_src, rig\_dst = commCart.Shift(1, +1)$
$fp1 = f[\ 1, :, :].copy()$- ½ * rhowall * lid_velocity #top wall
$p2 = f[-1, :, :].copy()$] + ½ * rhowall * lid_velocity #top wall
$commCart.Sendrecv(p1, top\_dst, recvbuf = p2, source = top\_src)$
Do this for all sides

Table 3.5 describes the algorithm that is used for communication between processes.

4

Numerical Results

In this chapter, we will discuss about the experimental setup for the shear wave decay project and how we have formulated till the simulated graphical results. A smooth syncing of analytical result vs numerical result has been presented.

4.1 Experiment validation

"A software without validation is like a conjecture, not a theorem"

~ Leslie Lamport

Also, in multiphase simulation, it is very crucial to make tests and validations to verify our results. These outcomes boost further development of codes upon that. We tried presenting every comparative study across the results shown.

4.1.1 Shear wave decay

With lattice Boltzmann, we see the dispersion and attenuation of waves because of the viscosity. We here focus on calculating the gradual decay of density (test case – I) and velocity (test-case 2) with time. The velocity and density decay has been observed along with kinematic viscosity change with factor ω. The velocity is given with a slight perturbation of sinusoidal push as an initial condition.

$$u(x, t = 0) = \begin{bmatrix} ux(y, t = 0) \\ 0 \end{bmatrix} = \begin{bmatrix} \epsilon sin\frac{2\pi y}{Y} \\ 0 \end{bmatrix} \tag{4.1}$$

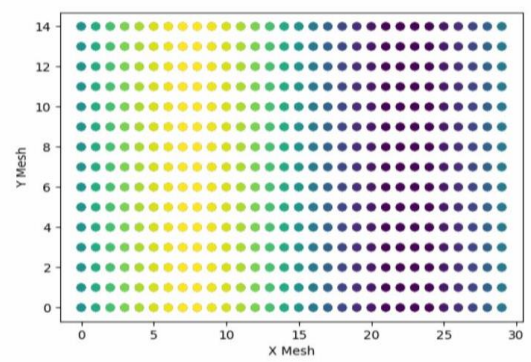
For analytical verification, we have use this time evolution formula: [11]

$$u_x(y, t) = \epsilon e^{(-\nu\left(\frac{2\pi}{Y}\right)^2 t)} sin\frac{2\pi y}{Y} \tag{4.2}$$

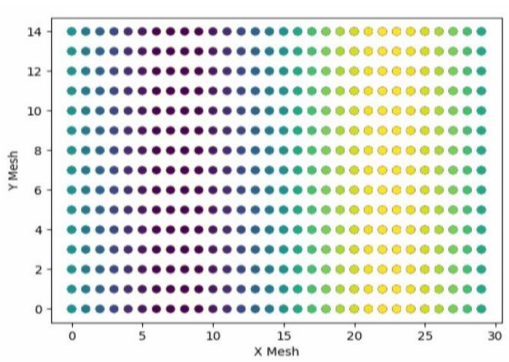
We have arrived to this formula using Navier stokes equation for an incompressible fluid. The navier stokes essential has a steadiness parameter which is time dependent along with the

pressure, volumetric and viscous parameters. We assume that the convective part $(u.\nabla)u$ and the pressure gradient (∇p) is negligible as compared to the viscous contribution $\nu\nabla^2 u$.

Density



(a) t = 0



(b) t = 100

Table 4.1 shows the density flow across grid (Nx = 30, Ny = 15, ω = 1.0). We observe that the density across a mesh of 30 x 15 has a density of particles shifting/streaming from left to right.

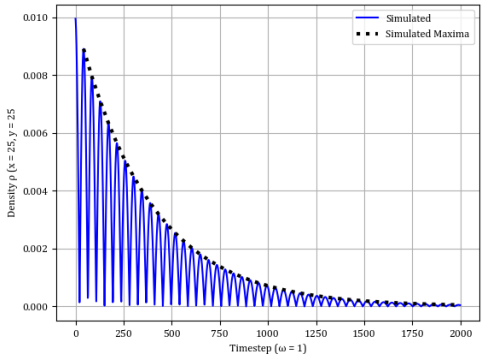
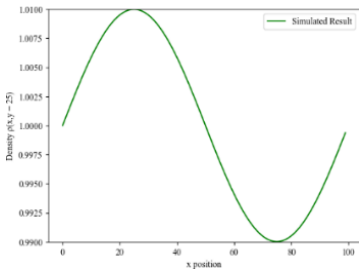
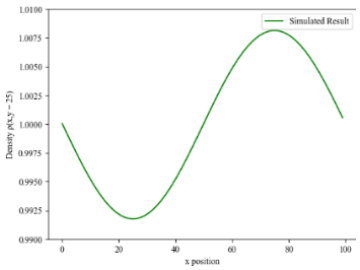


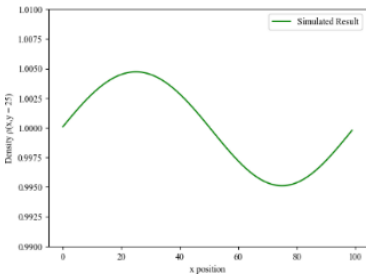
Table 4.2 shows the overall density decay measured at the centre of the density distribution for timescale 2000.



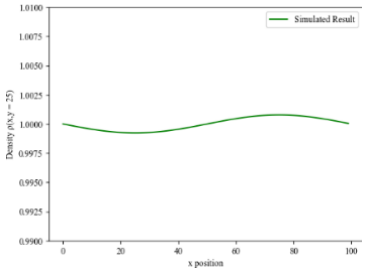
(a) t=0



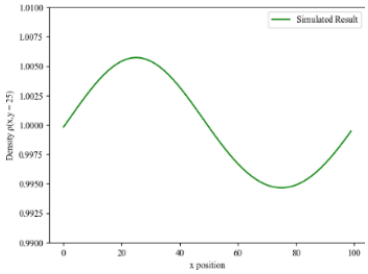
(b) t=100



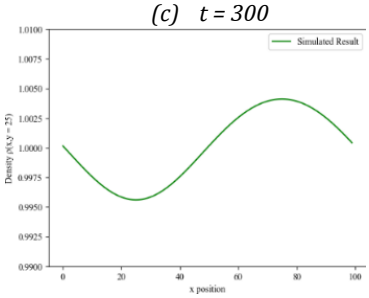
(c) t = 300



(d) t = 500



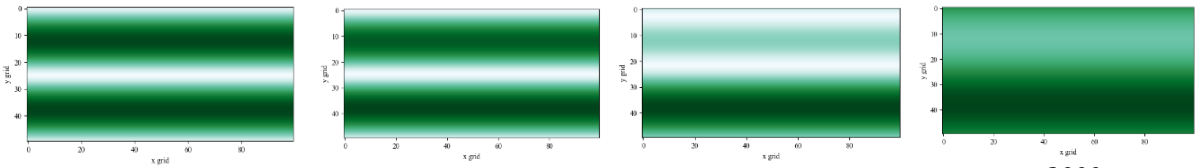
(e) t = 700



(f) t = 900

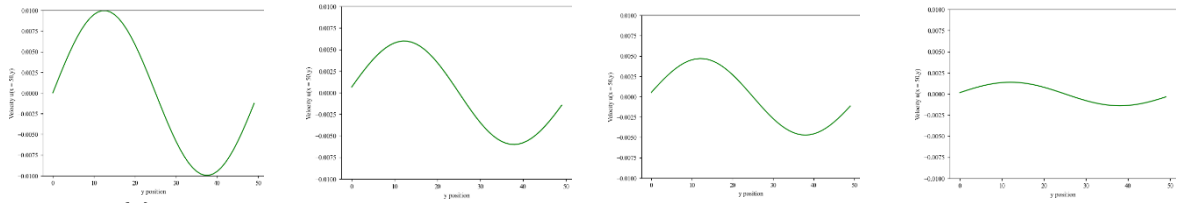
Table 4.3 shows the density decay for a specific position y=25 in space across the mesh $N_x = 100$

Velocity

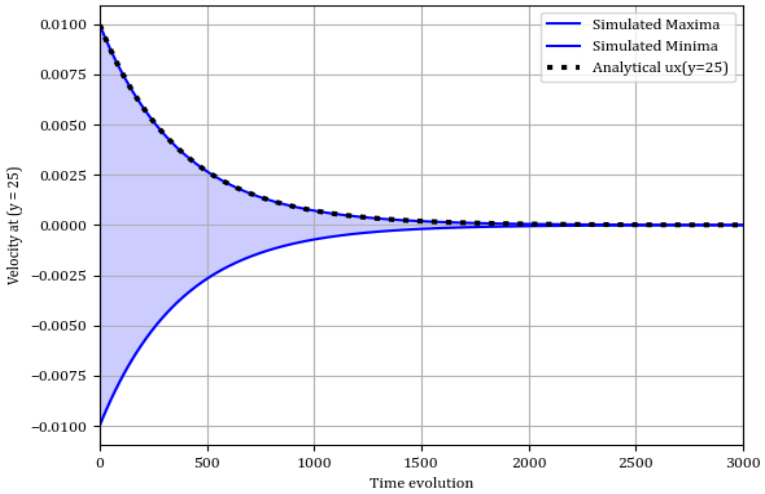


t = 0 *t = 1000* *t = 2000* *t = 3000*

Table 4.1 Velocity decay stream plot with time for 1000 time step. We observe that the streaming fades as the velocity losses its intensity with time (hence decay)



(a) t = 0 *(b) t = 100* *(c) t = 300* *(d) t = 800*



(e) Velocity decay with time for 3000 iterations

Table 4.2 Time evolution of sinusoidal velocity at x = 25 in the lattice size (100, 50). (a – d) shows the decay across y grid. X-axis here represents the y mesh grid whereas y here denotes the amplitude (limit set to +ϵ to −ϵ). Fig (e) shows the maximum velocity decay for a time step of 3000.

Kinematic Viscosity

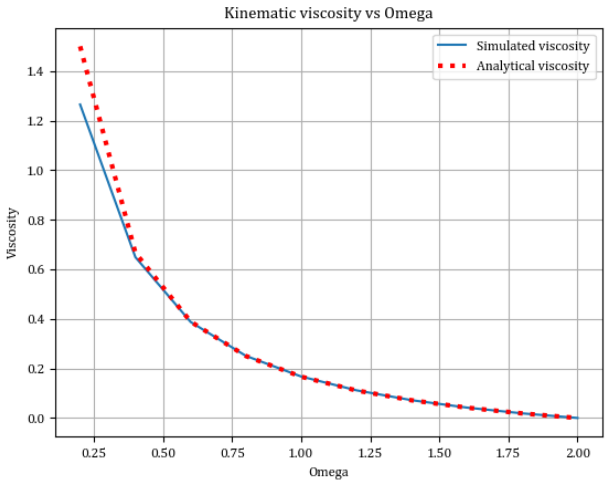
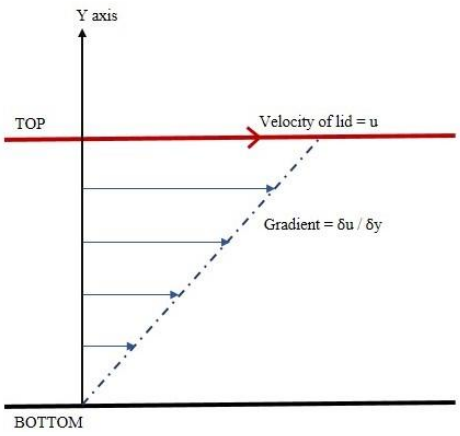


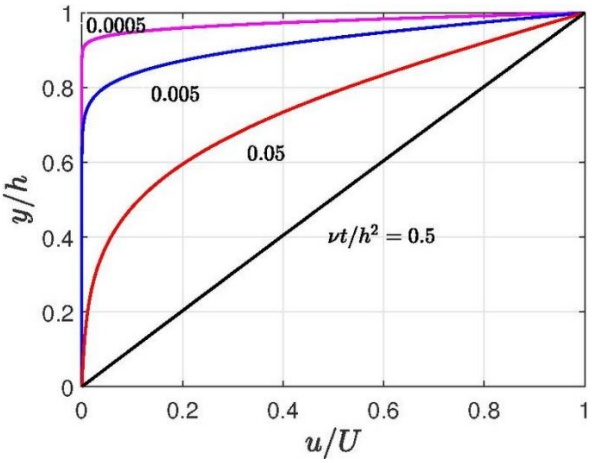
Table 4.3 shows the kinematic viscosity decay with time. We consider the analytical equation $\nu = \frac{1}{3}(\frac{1}{\omega} - \frac{1}{2})$ which varies with ω. And the simulated result follows the algorithm implementation presented in Algorithm [3]

4.1.2 Couette Flow

In couette flow, following the algorithm described in [4] we perform the experiment for around 4000 timestep with varied Nx and Ny. Typically the ω has been kept around 1.5 and the lid velocity around 0.1. The analytical formula is just a straight line equation $\frac{\delta u}{\delta x} * lid\ velocity$ from 0 to the lid velocity.



(a)



(b)

Table 4.3(a) is the couette analytical pictorial view where the top wall is moving and the bottom wall is fixed. Table 4.3(b) is a Wikipedia reference on how the couette flow profile must come. [12]

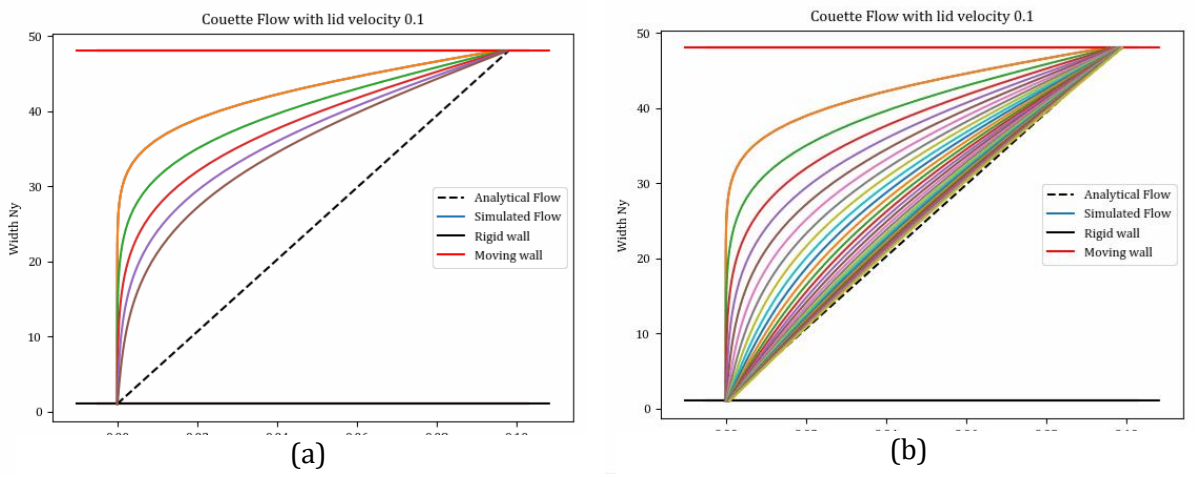
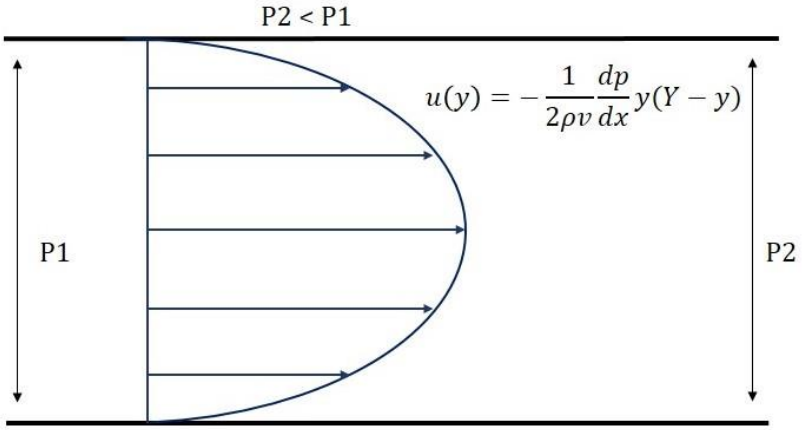
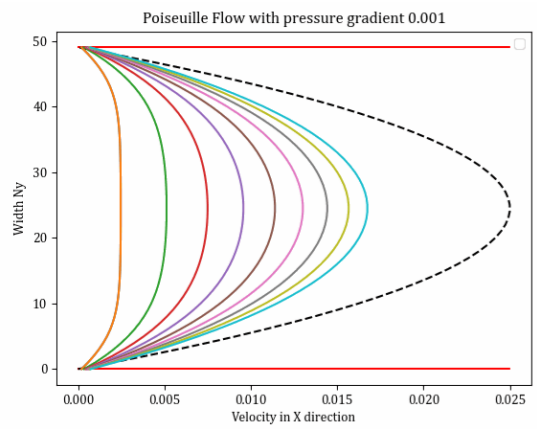


Table 4.4(a) shows the simulated couette flow profile after 2000 timestep and Table 4.4(b) shows the simulated profile after 4000 timestep.

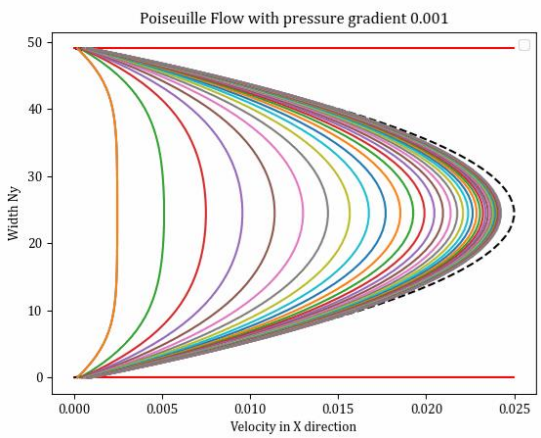
4.1.3 Poiseuille Flow:

In poiseuille flow, we follow the same streaming and collision that is being used in shear wave decay. Here, we additionally introduce pressure gradient as the driving force for the flow. This pressure gradient is incorporated in calculating the equilibrium of the flow thereby modifying the collision to some extent. In poiseuille, we take care of the periodic boundary for the pressure variation by calculating the equilibrium at the right and left boundary. We performed simulation for couple of pressure differences that meet with our expected analytical solution.

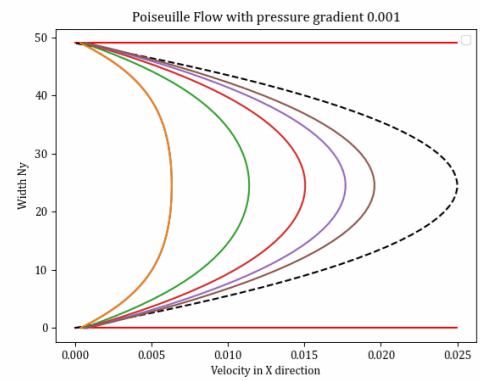




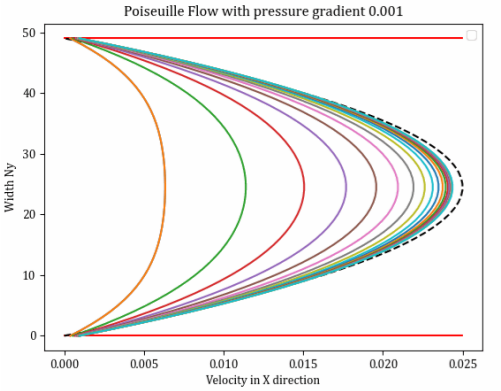
(b)



(c)



(d)



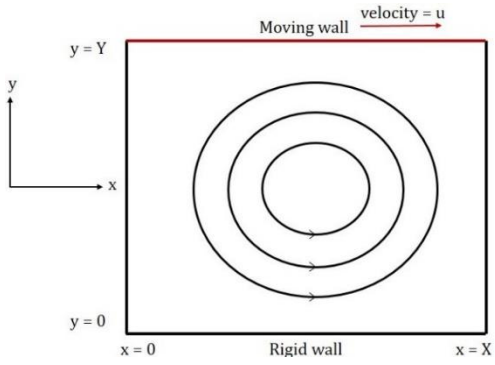
(e)

Table 4.4(a) is the analytical solution and an expected outcome of the Hagen-Poiseuille formula *[7] [13]*. Table 4.5(b) is the simulated poisuille flow after 4000 time step and Table 4.5(c) is the simulated plot after 8000 time step for X=50. For (b) and (c), the Y axis denotes the width of the pipe of the wall gap and X axis denotes the velocity in X direction. (d) and (e) refers to the same at X=1.

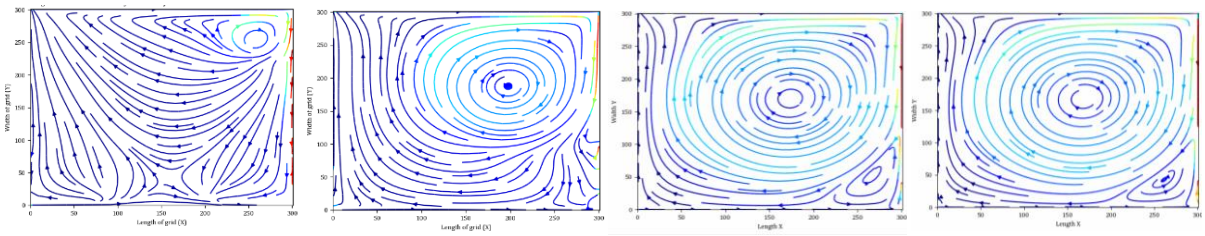
4.1.4 Sliding Lid

With the bounce back condition mentioned in [5], we experiment the sliding lid to observe the expected cavity. Following are the outcomes vs analytical expectation.

a) Serialized Sliding Lid:

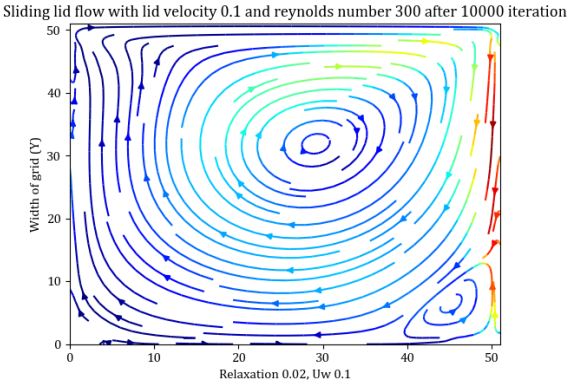


(a)

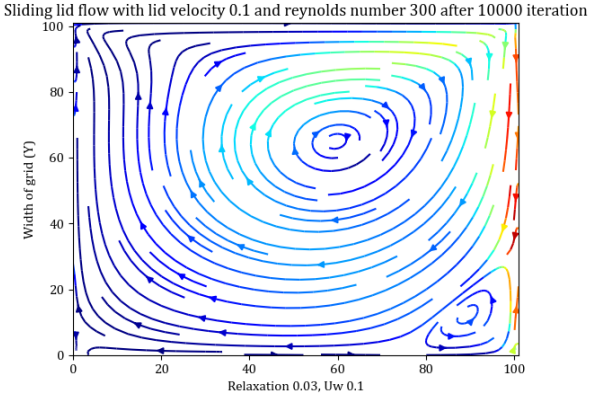


(a) (b) (c) (d)

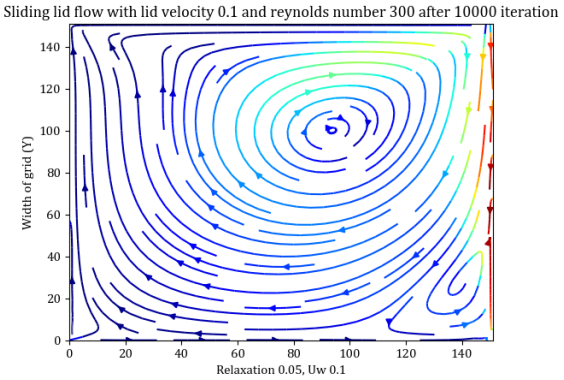
Table 4.5(a) is an analytical expectation of the experiment [14], Table 4.5 (b – e) represents the flow for various time step – 10000, 40000, 70000, 100000 respectively. Here Y denotes the width of the pipe or channel and x is the length of the box or the channel. And these are the serialized stream plots of the velocity.



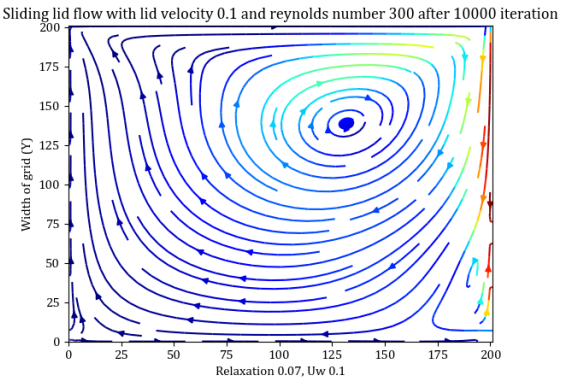
(a) Relax Time = 0.02, Uw = 0.1



(b) Relax Time = 0.03, Uw = 0.1



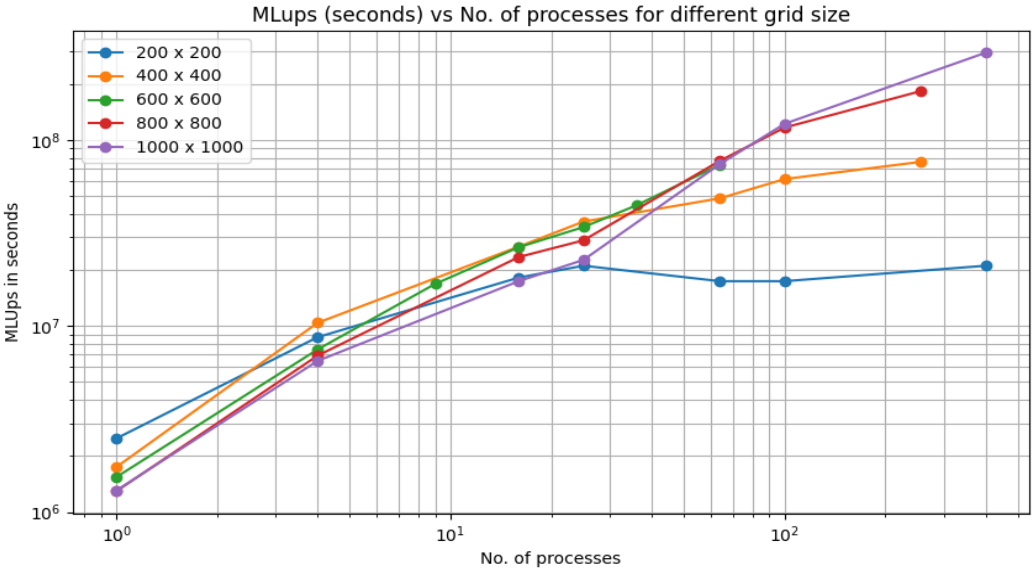
(c) Relax Time = 0.05, Uw = 0.1



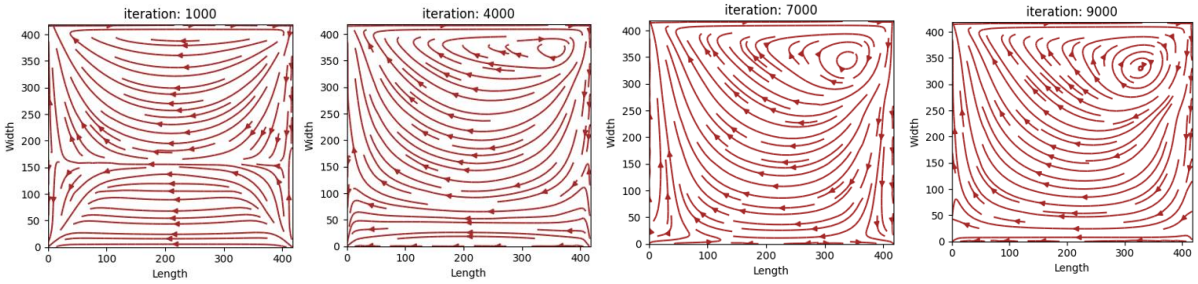
(d) Relax Time = 0.07, Uw = 0.1

Table 4.6(a-d) is a comparative study of sliding lid for different grid size (50, 100, 150, 200) and relaxation time. Relaxation time has been calculated by the formula $relax\ time\ = \frac{Length*Uw}{Re}$. We here notice that with increase grid size, for a same timescale of 10000 iterations, and same wall velocity 0.1, the lid is fully developed in smaller grid sizes and is partially developed in larger grids.

b) Parallelized Sliding Lid:



(a)



(b) (c) (d) (e)

Table 4.5(a) is the MLUps vs number of processes (grid size 400, 800, 1200, 1600, 2000 and for processes 1, 4, 16, 25, 64, 100, 400) and (b) – (e) are the parallelized sliding lid for 8 processes. It also follows the Amdalh's formula for speedup. [15]

5

Conclusion

In this paper, we presented the theoretical aspect of lattice Boltzmann as well as the practical implementation of the LBM code. The simulation strategies are carried out at most possible efficient manner. Chapter 1 talks about a brief introduction to Lattice Boltzmann technique.

Chapter 2 categorically describes the theory of Lattice Boltzmann and how we derive the streaming and collision operator. And we invoke them in a D2Q9 lattice. The BTE equation and

other important equations are described here. We have also presented the density, velocity and probability density lattice distribution and how we perform things in coding aspect.

Chapter 3 has the main experimental setups described and theories behind the simulations. Typical algorithms for streaming, collision, shear wave decay, couette, poiseuille, sliding lid and communication in MPI has been described here. Also, the MPI and domain decomposition is described in this chapter. The discretization, boundary handling for serialized and parallelized has also been described in this chapter.

Chapter 4 has all the numerical results jotted down. We present shear wave decay density and velocity changes with time. We also calculated the kinematic viscosity vs omega variations. For Couette and Poiseuille, we simulate the velocity vs X grid characteristics. We provide the Analytical and simulated plots for Couette and Poiseuille thereby ensuring the simulation generates accurate results. We also perform serialized and parallelized sliding lid simulation. For serialized, we ran the simulation for 100000 timestep and for parallel simulation, we provide with the same ran for 8 cores. To calculate the MLUps, we used BwUniCluster and performed simulations for grid size 400, 800, 1200, 1600, 2000 and for processes 1, 4, 16, 25, 64, 100, 400.

We have also checked the scalability of the LBM in Sliding lid simulations. We noticed that parallelization works much faster than that of serialized one. The smaller domains have shown less efficiency and also performed slower once we increase the number of processes. Thus, it follows the Amdahl's law for synchronization time.

Thus, this paper gives a comprehensive study on Lattice Boltzmann Method and an intricate details about the implementation strategy.

Bibliography

[1] J. M. A.R.Davies, "Simulation of a 3-D Lid-Driven Cavity Flow by a Parallelised Lattice Boltzmann Method," *Parallel Computational Fluid Dynamics.*

[2] A. K. D.Arumuga Perumal, "A Review on the development of lattice Boltzmann computation of macro fluid flows and heat transfer," *Alexandria Engineering Journal.*

[3] N. P. Moran Wang, "Lattice-Boltzmann Method," *Materials Science and Engineering Reports.*

[4] "dive-solutions," Dive Solutions, [Online]. Available: https://www.dive-solutions.de/articles/cfd-methods.

[5] J. S. Galsin, "Transport Phenomena," in *Solid State Physics*, Academic Press - Science Direct.

[6] K. Huang, "Statistical mechanics, john wily & sons," p. 10.

[7] L. P. a. A. Greiner., “Hpc with python: An mpi-parallel implementation of,” [Online].

[8] M. Technologies, “Streaming Computation for Lattice Boltzmann Method,” in *IEEE International Conference*.

[9] W. L. X. C. Y. P. Li-Shi Luo, “Numerics of the lattice Boltzmann method: Effects of collision models on the lattice Boltzmann simulations,” *National Library of Medicine.*

[10] R. P. a. T. D. Taylor, “Computational methods for fluid flow.”.

[11] K. H. L. a. Q. L. Linlin Fei, “Three-dimensional cascaded lattice boltzmann method: Improved implementation and consistent forcing scheme”.

[12] “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/File:StartupCouette.pdf.

[13] “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Hagen%E2%80%93Poiseuille_equation.

[14] M. M. R. B. A. C. H. Lamarti, “Numerical simulation of mixed convection heat transfer of fluid in a cavity driven by an oscillating lid using lattice Boltzmann method,” *Science Direct.*

[15] P. A. LaPlante, “Real Time Systems Design And Analysis”.

[16] “Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Couette_flow.