High-Performance Computing: Fluid Mechanics simulations with Python
Project Report

Jad Elkarchi, *Albert-Ludwigs University, Freiburg im Breisgau*
Matriculation number : 5653652
je303@students.uni-freiburg.de

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 have revolutionized fluid flow simulations, providing insights into complex fluid dynamics. This report focuses on the implementation of the lattice Boltzmann equation (LBE) and its principles in the context of high-performance computing (HPC) for fluid mechanics.

The LBE, derived from kinetic theory, enables the modeling of fluid flow by dividing the computational domain into a lattice grid, a 2D grid in our study. Within this framework, we study the streaming operator that governs the movement of particles, and also the collision operator that models their interactions. [1]

To validate and refine our 2D lattice Boltzmann model, we conduct many simulations that replicate well-known fluid mechanics experiments. These simulations serve as benchmarks to confirm the accuracy and reliability of our model while enhancing our understanding of fluid mechanics phenomena. Specifically, we investigate the behavior of shear waves, Couette flow, Poiseuille flow, and the sliding lid experiment. By closely examining these simulations, we can better understand the strengths and limitations of our model and refine its implementation for improved accuracy and performance.

While our first focus is on the serial implementation of the lattice Boltzmann equation, we later apply parallelization techniques for advancing our study. Parallelization enables us to exploit the computational power of clusters, facilitating simulations on a larger and more realistic scale. By leveraging clusters of processes, we can enhance the performance and efficiency of our simulations, ultimately enabling us to tackle more complex and computationally demanding fluid mechanics problems.

In this report, we present the methods used for implementing the lattice Boltzmann equation, including the principles of fluid mechanics, the LBE, and the collision operator. We then discuss details of our implementation, present the results obtained from simulations, and evaluate their accuracy and reliability. Furthermore, we analyze the potential for parallelization and scaling tests to enhance the performance of our model in the future.

By combining the principles of fluid mechanics with the computational efficiency of the lattice Boltzmann method, we aim to contribute to the field of computational fluid dynamics and pave the way for more efficient and accurate simulations.

2 Methods and Simulations

2.1 Principles of fluid mechanics

2.1.1 LBE and streaming operator

In our model, the discretization of the Boltzmann Transport Equation (BTE) is a crucial start. We employ a discretization scheme on both the velocity space and the physical space to represent fluid behavior accurately. The discretization on the velocity space is achieved by using the D2Q9 scheme, which considers nine prescribed directions, denoted as c_i. These directions form a velocity set that governs the movement of particles within the lattice grid.

In the physical space, we utilize a uniform 2D grid for discretization. This structured mesh consists of equidistant grid elements, enabling us to represent the spatial distribution of the probability density function f.

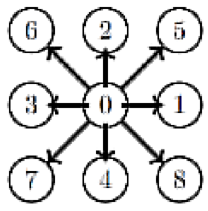


Figure 1: D2Q9 scheme

To initiate our model, we create a probability grid called f, representing the probability density function. In this discretization, we substitute the notation $f(r, v, t)$ with $f_i(r, t)$, where the index i refers to one of the nine possible velocity channels. Each $f_i(r, t)$ denotes the number of particles located at position r at time t, moving in direction c_i. This discretization enables us to propagate the values of f_i along the corresponding velocity directions in a step-by-step manner, effectively simulating the particle movement. This process is facilitated by the streaming operator.

$$f_i(r + \Delta t c_i, t + \Delta t) = f_i(r, t) \tag{1}$$

For our simulation, two crucial grids are employed: the density grid (denoted by ρ) and the velocity grid (denoted by u). The density grid represents the spatial distribution of density within the lattice, and the velocity grid captures the local velocity values. Both grids play essential roles in determining the initial conditions of the simulation and calculating the collision term later in the process. The formulas defining the discretized density and velocity grids are depicted in equation (2) and equation (3), respectively.

$$\rho(r) = \Sigma_i f_i(r) \tag{2}$$

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

2.1.2 Collision operator

The collision operator plays a crucial role in our model as it governs the relaxation of the probability density function $f_i(r,t)$ towards an equilibrium distribution $f_{eq}(r,v,t)$. The collision operator is represented by the equation (4).

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

In this equation, ω represents the relaxation parameter, determining the rate at which the distribution function relaxes towards equilibrium. The distribution function $f_{eq}(r,v,t)$ represents the equilibrium distribution that the probability density function locally relaxes to. This equilibrium distribution is calculated based on the local density ρ and velocity u at position r and time t.

$$f_i^{eq}(\rho(r), u(r)) = W_i \cdot \rho(r)[1 + 3 \cdot c_i \cdot u(r) + \frac{9}{2} \cdot (c_i \cdot u(r))^2 - \frac{3}{2}.|u(r)|^2] \tag{5}$$

By applying the collision term into our model, we examine how the relaxation parameter ω affects the flow of particles within the lattice. Adjusting the relaxation parameter allows us to control the level of relaxation towards equilibrium, and subsequently, it impacts the behavior of the fluid flow. A higher relaxation parameter leads to faster relaxation towards equilibrium, while a lower parameter slows down the relaxation process, thus acting similarly to viscosity parameter.

The collision term enhances the realism of our model by simulating the interactions and collisions between particles within the lattice. By considering the relaxation towards equilibrium, we capture the dynamic behavior of fluid flow more accurately. This collision term introduces local variations in the probability density function, enabling us to observe and analyze phenomena such as diffusion and viscosity.

2.2 Simulations

2.2.1 Study of shear wave

In our shear wave decay simulation, we employ a commonly used method in computational physics to measure the kinematic viscosity of a fluid. The shear wave method involves setting a sinusoidal velocity or a sinusoidal density profile within our simulation domain and observing how quickly it decays over time. This approach allows us to study the behavior of the fluid and investigate the relationship between the relaxation parameter (ω) and the kinematic viscosity (v).

To begin the simulation, we choose initial values for the density (ρ) and velocity (u) at time t=0. For the first case, we set the initial density distribution as ρ(r,0) $= \rho_0 + \epsilon \sin(\frac{2\pi x}{L_x})$ and the initial velocity distribution as $u(r,0) = 0$.

Here, L_x represents the length of the domain in the x direction. By observing the 2D density distribution over time, we gain insights into the dynamic behavior of the fluid.

Next, we consider the second case where the initial density distribution is ρ(r,0) = 1, and the initial velocity distribution is ux(r,0) = ϵ sin($\frac{2\pi y}{L_y}$). In this scenario, we observe the dynamic behavior of the fluid as well as its long-time limit.

By conducting these experiments and studying the results, we can establish a relationship between the relaxation parameter ω and the viscosity of the fluid. We assume that the initial conditions satisfy the Stokes flow condition (6),

$$\frac{du}{dt} = v \cdot \nabla^2 u \tag{6}$$

This condition is solved as a first-order differential equation, and we obtain a function that satisfies this equation (with the sinusoidal velocity profile, and with the right parameters $\epsilon, L_y, \omega, ...$). This function depends on the viscosity (v), time (t), and the periodicity of the probability field ($k_1 = \frac{2\pi}{L_y}$).

$$a(t) = a_0 \cdot e^{-v.k_1^2.t} \tag{7}$$

a_0 being the first amplpitude. During our study we can save $a(t)$ and thus calculate v using the formula (7).

Through our analysis, explore how viscosity scales with the relaxation parameter ω. The theoretical prediction suggests that the viscosity follows a specific relationship with ω, given by

$$v = \frac{3}{2}(\frac{1}{\omega} - \frac{1}{2}) \tag{8}$$

By investigating the relationship between the relaxation parameter ω and the kinematic viscosity through these shear wave decay simulations, we can deepen our understanding of the behavior of the fluid and validate theoretical predictions. This exploration allows us to assess the impact of ω on fluid properties and further refine our model accordingly.

2.2.2 Study of Couette flow

The Couette flow experiment is a classic fluid dynamics experiment used to study the flow of viscous fluids between two parallel plates. In this setup, one plate is fixed while the other plate moves with a constant velocity, creating a shearing effect on the fluid. By observing the behavior of the fluid under different conditions, valuable insights can be gained about its viscosity and flow patterns.

To simulate the Couette flow experiment on a 2D grid, we utilize the following formulas. Firstly, the steady state velocity profile can be described by the linear equation:

$$u(y) = \frac{U_w}{h}.y \tag{9}$$

where $u(y)$ represents the velocity of the fluid at a given vertical position y, U_w is the velocity of the moving plate, and h is the distance between the plates. We will try to reach this steady profile with our approach.

We increase our 2D grid's dimension by adding 2 ghost lines, top and bottom, acting as reflectors of the top and bottom channels and eliminating the periodic shape (in our case, in top and bottom walls). The following figure 2 show how we theoretically create these reflectors. Reflecting upper and lower channels, 5 with 7, 6 with 8, and 2 with 4.

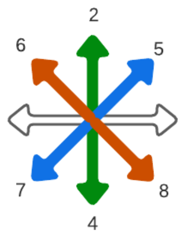


Figure 2: reflected channels

Then we describe the movement of the upper wall by redirecting the flow in top and bottom channels of the ghost points such that a wall movement is created, x_b is the index of the cell at the boundary.

$$f_i(x_b, t + \Delta t) = f_i(x_b, t) - 2.W_i.\rho_w.\frac{c_i.U_w}{c_s^2}. \tag{10}$$

We apply the boundary conditions showed above after the streaming process, and in our case we apply the moving boundary condiitions on the upper wall, which means that we apply this formula on every cell ghost in the top line.

By incorporating these formulas into our 2D grid simulation, we can observe and analyze the flow patterns, velocity distribution, and shear stress values within the fluid. Such experiment and computations allow us to deepen our understanding of fluid mechanics and facilitate the development of various engineering applications.

2.2.3 Study of Poiseuille flow

The Poiseuille flow experiment is a fundamental investigation in fluid dynamics that aims to understand the laminar flow of an incompressible fluid through a cylindrical pipe or channel, in our 2D model the pipe is a long and narrow grid. This setup involves applying a pressure gradient across the length of the pipe, resulting in a parabolic velocity profile within the fluid. We've used the fixed boundary conditions of the previous experiment both on the upper and the lower wall.

To simulate the Poiseuille flow experiment, we are implementing the following equations:

At the starting point of the grid, represented by x_0, we have:

$$f_i(x_0, y, t) = f_i^{eq}(\rho_{in}, u_n) + (f_i(x_n, y, t) - f^{eq}i(x_n, y, t)) \tag{11}$$

Here, $f_i^{eq}(\rho_{in}, u_n)$ represents the equilibrium distribution function for the fluid density ρ_{in} and the velocity component u_n. The term $(f_i(x_n, y, t) - f_i^{eq}(x_n, y, t))$ accounts for the difference between the non-equilibrium distribution function f_i at position (x_n, y, t) and the equilibrium distribution function f_i^{eq} at the same position.

At the endpoint of the grid, represented by x_{n+1}, we have:

$$f_i(x_{n+1}, y, t) = f_i^{eq}(\rho_{out}, u_1) + (f_i(x_1, y, t) - f_i^{eq}(x_1, y, t)) \tag{12}$$

Here, the components are similar the only change is that it's the cells at x_{n+1} that are changed. Also, the channels i considered here are the only channels horizontally pointing to the grid, in this case it's the channels 1, 5, and 8. These pressure inlets and outlets are applied before the streaming process.

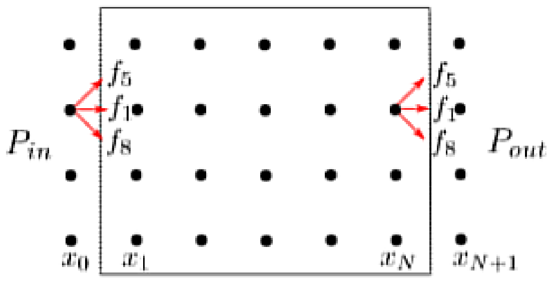


Figure 3: Summary of the poiseuille experiement

Also, The values of ρ_{out} and ρ_{in} grids, which represent the gradient of pressure outside the grid are calculated based on the given pressure values p_{out} and p_{in}.

$$\Delta p = p_{out} - p_{in} \tag{13}$$

$$\rho_{out} = \frac{p_{out}}{c_s^2} \tag{14}$$

$$\rho_{out} = \frac{p_{out} + \Delta p}{c_s^2} \tag{15}$$

We then introduce the analytical solution of the velocity field for a Hagen-Poiseuille flow in a pipe. Under the assumption of laminar flow, we can simplify the Navier-Stokes equations and consider only the streamwise component of the x, which is aligned with the pipe axis we then integrate it along y twice. Here is the final expression :

$$u(y) = -\frac{1}{2\mu}\frac{dp}{dx}y(h-y) \tag{16}$$

$$\mu = \rho.v \tag{17}$$

This formula 16 can be then be integrated and used to compare the analytical solution with the experimental one.

2.2.4 Study of sliding lid

This chapter discusses a new experiment involving a closed grid, where the top wall of the grid moves with a constant velocity to the east. This motion induces a flow within the grid, resulting in characteristic vortices (that we should obtain when running our model). To quantify this phenomenon, the Reynolds number (Re) is introduced, which represents the ratio between the inertial terms and the viscous ones in the fluid. It is calculated by dividing the product of a characteristic length (L), the flow velocity (u), and the kinematic viscosity (v).

$$Re = \frac{Lu}{v} \tag{18}$$

Building upon our previous Couette flow experiment, we will now apply the concept of a sliding lid to our simulation. Using the Lattice Boltzmann method. Our objective is to observe and understand the flow patterns that emerge as we vary the Reynolds number, specifically exploring values up to Re=1000.

2.3 Parallelization

A parallel sliding-lid lattice Boltzmann method (LBM) simulation uses the Message Passing Interface (MPI) for parallel computing. The sliding lid condition allows the fluid in the domain to move freely while the top boundary, representing the lid, slides horizontally (as seen earlier), introducing dynamic behavior to the fluid. To achieve this, the simulation domain is divided into multiple grids, a grid of grids, and each grid cell is simulated by an individual process in a parallel manner using MPI. [2]

In order to make the sliding-lid mechanism work correctly in parallel, it is first crucial to divide thoroughly the main grid into small and equal grids. Second, to communicate the edge information of each running processor to its neighboring processors. A cartesian world is created to arrange the processes in a logical grid structure. A direction set is used to define the neighboring processors for each process in the grid. During each iteration of the simulation, we exchange the boundary information of each processor with its neighbors. This boundary exchange allows the correct transfer of particles across the processor boundaries, ensuring that fluid dynamics are appropriately captured and maintained throughout the sliding-lid simulation.

By dividing the simulation domain into smaller grids and assigning each grid to a separate processor, the code achieves parallelism. Each processor runs an independent simulation on its assigned grid section. The processors then communicate with each other enabling them to interact and update their respective local simulations based on the changes in neighboring regions. This parallel approach allows the simulation to be efficiently distributed across multiple processors, significantly reducing the computational time and allowing the simulation to be scaled up to larger domains and higher resolutions. Which is the main goal we want to verify in this experiment.

The main data structure is the D2Q9 grid that represents the fluid density and velocity at each lattice point in the LBM simulation. Each processor operates on its local section of this grid, and through the MPI communication, the edge information is exchanged, enabling fluid particles to flow seamlessly and properly across the processor boundaries. By parallelizing the simulation in this manner, the sliding-lid LBM code efficiently simulates complex fluid dynamics while effectively utilizing the computational resources of modern parallel computing environments, in our case being BW_UniCluster.

3 Implementation

The implementation process begins by setting up the computational environment, ensuring the required libraries and packages are installed and the programming environment is configured. Next, the simulation parameters are defined, including the domain size (L_x, L_y), relaxation parameter ω, time step t, and any other relevant parameters based on the theoretical model. In the figure below all the necessary parameters for the base study are noted :

$$0 < \omega < 2$$

$$0 < \rho < 1$$

$$|u| < 0.1$$

$$\epsilon \in \{0.1, 0.01\}$$

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

$$W_i = \left[\tfrac{4}{9}, \quad \tfrac{1}{9}, \quad \tfrac{1}{9}, \quad \tfrac{1}{9}, \quad \tfrac{1}{9}, \quad \tfrac{1}{36}, \quad \tfrac{1}{36}, \quad \tfrac{1}{36}, \quad \tfrac{1}{36}\right]$$

A uniform 2D grid is then created to discretize the physical space, providing a structured mesh for the spatial discretization of the probability density function (f) and other variables involved in the simulation. The density (ρ) and velocity (u) fields are initialized according to the chosen initial conditions, such as a sinusoidal variation in density and zero initial velocity, or a sinusoidal variation in velocity and constant density.

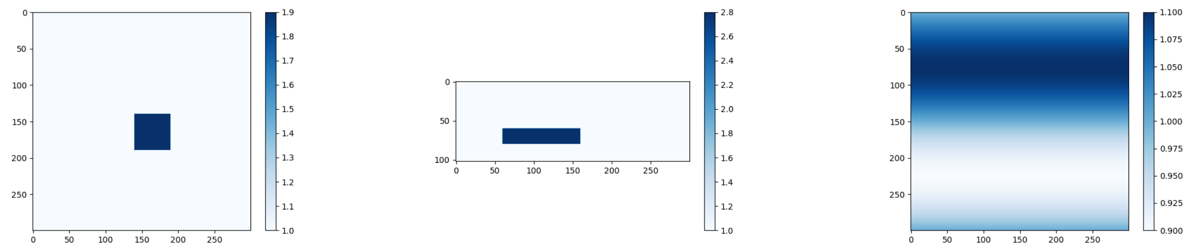
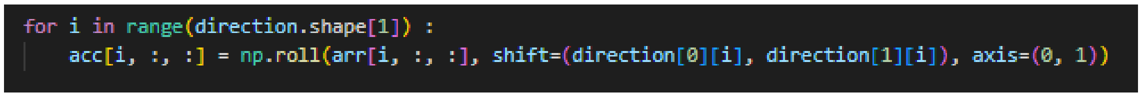


Figure 4: Three different types of initial probability density grid

The time integration scheme is then implemented to propagate the density and velocity fields in time. Through iterative steps, the distribution function (f) is streamed along the prescribed directions (c_i) for each time step (Δt) using a method in numpy package called `roll`, thus applying the streaming operator.



```
for i in range(direction.shape[1]) :
    acc[i, :, :] = np.roll(arr[i, :, :], shift=(direction[0][i], direction[1][i]), axis=(0, 1))
```

Figure 5: Rolling over the matrix for each direction in the 9 channels

The directions being implemented as shown in the figure below :

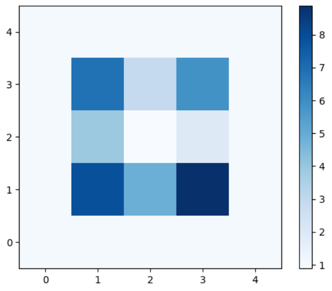
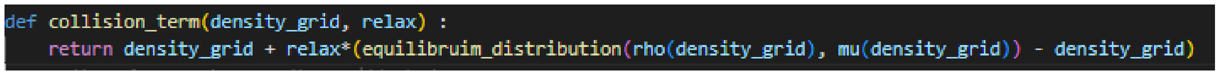


Figure 6: direction channels implemented, color gradient from light to dim represents from direction 0 to direction 8.

As for the collision operator it is applied as shown in (4), ensuring the relaxation of the distribution function towards the equilibrium distribution function (f_{eq}), which is calculated based on the density and velocity fields (ρ, u). This relaxation is controlled by the relaxation parameter (ω) as seen in earlier in the collision operator chapter. The following Figure shows how it is applied after each streaming step.



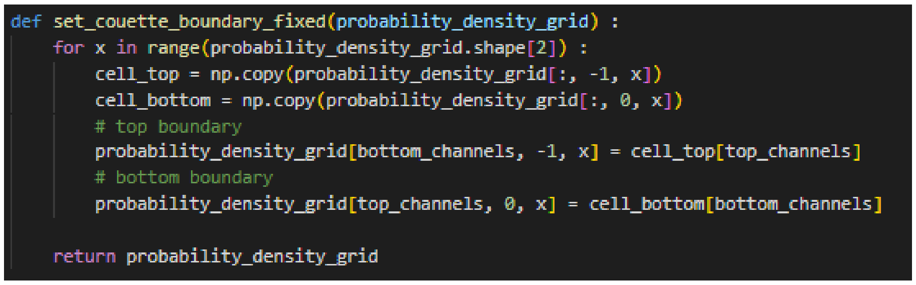
```
def collision_term(density_grid, relax) :
    return density_grid + relax*(equilibruim_distribution(rho(density_grid), mu(density_grid)) - density_grid)
```

Figure 7: Application of the collision term

3.1 Couette flow

Our approach to simulate the couette flow experiment, is to add ghost lines that will play the role of walls. This was possible by increasing the grid's size by 2. By implementing it this way, we were able to reverse the flow top channels became bottom channels and vice versa.

As said earlier the boundary conditions is called after every streaming step, in each call we go through the cells of the two walls and for each cell we reverse the channels. the following figure shows our code implementing the boundary condition.



```
def set_couette_boundary_fixed(probability_density_grid) :
    for x in range(probability_density_grid.shape[2]) :
        cell_top = np.copy(probability_density_grid[:, -1, x])
        cell_bottom = np.copy(probability_density_grid[:, 0, x])
        # top boundary
        probability_density_grid[bottom_channels, -1, x] = cell_top[top_channels]
        # bottom boundary
        probability_density_grid[top_channels, 0, x] = cell_bottom[bottom_channels]

    return probability_density_grid
```

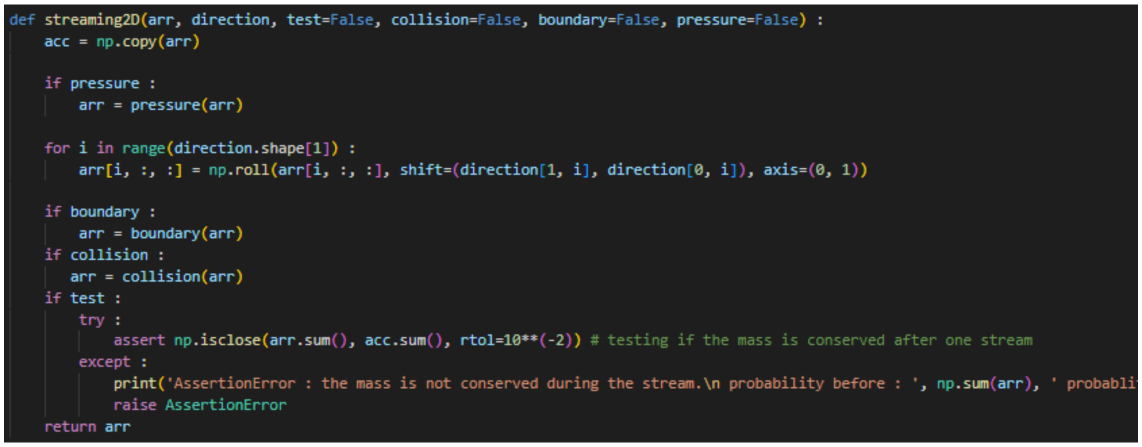
Figure 8: Caption of the wall reflection code

The only thing left is to make the top wall move east, and this was simple since we apply the formula 10 only to the top wall.

3.2 Poiseuille flow

In order to implement this experiment correctly, we've started by optimizing the dimensions of our grid, having a narrow but long grid imitating a pipe, the dimensions we chose were (300×100).

Of course we used the techniques we needed from the previous studies, and so we've added 4 ghost lines for the four sides, we used the couette's approach to create fixed reflecting wall on the top and bottom layers, as for the left and right layers we use them to transmit our pression using the formulas shown previously 11 and 12. A final streaming function is shown below (The pressure boundary is called as a function before the roll).



```
def streaming2D(arr, direction, test=False, collision=False, boundary=False, pressure=False) :
    acc = np.copy(arr)

    if pressure :
        arr = pressure(arr)

    for i in range(direction.shape[1]) :
        arr[i, :, :] = np.roll(arr[i, :, :], shift=(direction[1, i], direction[0, i]), axis=(0, 1))

    if boundary :
        arr = boundary(arr)
    if collision :
      arr = collision(arr)
    if test :
        try :
            assert np.isclose(arr.sum(), acc.sum(), rtol=10**(-2)) # testing if the mass is conserved after one stream
        except :
            print('AssertionError : the mass is not conserved during the stream.\n probability before : ', np.sum(arr), ' probabli
            raise AssertionError
    return arr
```

Figure 9: Caption of the final version of the streaming function

3.3 Sliding lid

Similarly to the two previous experiences, for each boundary cell in the grids we've fixed the left, right, and bottom walls, and set the top wall with a velocity u_w to the east. Nothing new when implementing this experiment, but the results of this specific simulation are quite interesting.

3.4 Parallelization

In our parallel implementation we followed a step by step process :

- **Initialization and Setup :** The code begins by initializing MPI communication, retrieving the rank and size of the current process, and calculating the dimensions of each local node. It then creates a Cartesian communicator `cartcomm` to arrange the processes in a logical grid structure. Then we define the `direction_dict` dictionary, which determines the neighboring processes (in terms of ranks) in the `north`, `south`, `east`, and `west` directions for each process in the Cartesian communicator.
- **Boundary Positioning :** The `proc_position` dictionary is created to identify the boundary positions of each processor. It is used to check whether each process has a neighboring process in each direction (`north`, `south`, `east`, `west`) based on the `direction_dict`. If a neighboring process is not present in a specific direction, it means that the current process is positioned at that boundary.
- **Collision Function :** The code also defines a `collision_function`. It applies the collision term to the `density_grid` using the relaxation parameter ω. The collision term is a key part of the lattice Boltzmann method, responsible for relaxing the distribution functions towards equilibrium and simulating collisions between particles.
- **Grid Initialization :** The `probability_density_grid` is initialized as a data structure representing the probability density function of the fluid at each lattice point. It is created using the `create_density_grid` function with the specified shape.
- **Main Simulation Loop :** The simulation is run for a specified number of frames (iterations). Within each frame, the following steps are performed :

  - **a. Boundary Communication :** The `send_cell_boundary` function is called to exchange the boundary information of the `probability_density_grid` with neighboring processes. This step ensures that particles can move freely across the processor boundaries, effectively simulating fluid dynamics across the entire domain.
  - **b. Velocity Calculation :** The velocity function calculates the velocity field u based on the updated grid. The velocity field is an essential component for determining the particle streaming direction in the streaming step.
  - **c. Streaming and Collision :** The `streaming2D` function is called to perform the streaming step of the lattice Boltzmann method. It moves particles according to their velocities, simulating the movement of fluid particles. The `collision_function` is applied to the updated grid to model collisions and relaxation towards equilibrium.

- **Results Saving :** After the simulation is complete, the velocity field v is extracted from u by removing the ghost points from each process's grid, and the results are saved to files using the `save_mpiio` function. The barrier calls `Barrier()` ensure that all processes synchronize at specific points before continuing.

4 Results

4.1 Streaming operator

To verify the accuracy of the streaming algorithm used in our simulation, we performed a validation test by rolling the probability density grid (f). Through this test, we ensured that the streaming operator correctly propagated the distribution function along the prescribed directions. Additionally, to visually assess the behavior of the particles within our model, we created a video showcasing a naive case where random particles were observed in motion.

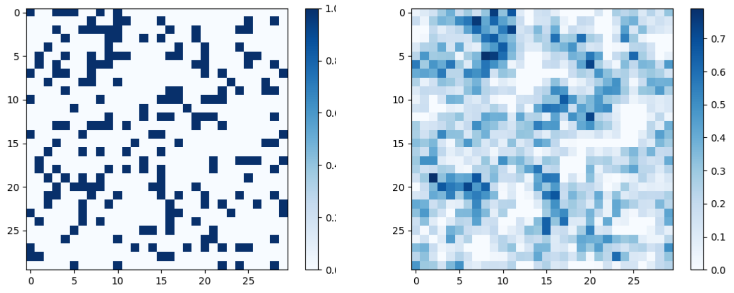


Figure 10: Streaming operator applied before (left) and after (right)

Furthermore, we added a dedicated test to verify the conservation of mass during the streaming process. Mass conservation is an important invariant in our experiments, and this test provided assurance that the streaming algorithm preserved this fundamental property throughout the simulations.

4.2 Collision application

After successfully implementing and verifying the streaming algorithm for the probability density grid (f), we proceeded to introduce the collision operator to our simulation. This crucial step involved applying the collision term into our animations, further enhancing the accuracy of our model. As we added the collision operator, we conducted tests to ensure the conservation of mass throughout the simulation. By validating mass conservation during the application of the collision term, we ensured that our model accurately captured the interactions and dynamics of the fluid particles, making it more faithful to real-world fluid behavior. Some of the integral tests of this part is in the simulations.

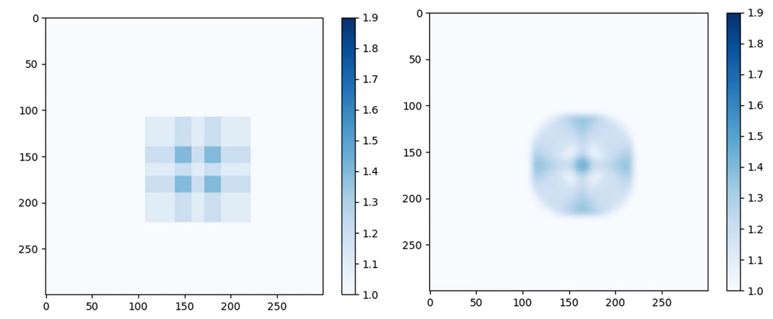


Figure 11: Streaming particles without collision (left) and with collision (right)

4.3 Simulations

4.3.1 Shear wave decay

In the shear wave decay simulation, we aimed to investigate the behavior of waves within our fluid model and establish a relationship between the relaxation parameter and viscosity. Initially, we set up a sinusoidal velocity profile in our simulation domain and observed how the wave decayed over time. Through visualizations of the density distribution, we witnessed the sinusoidal aniamtion of the wave, but as expected it was decaying.

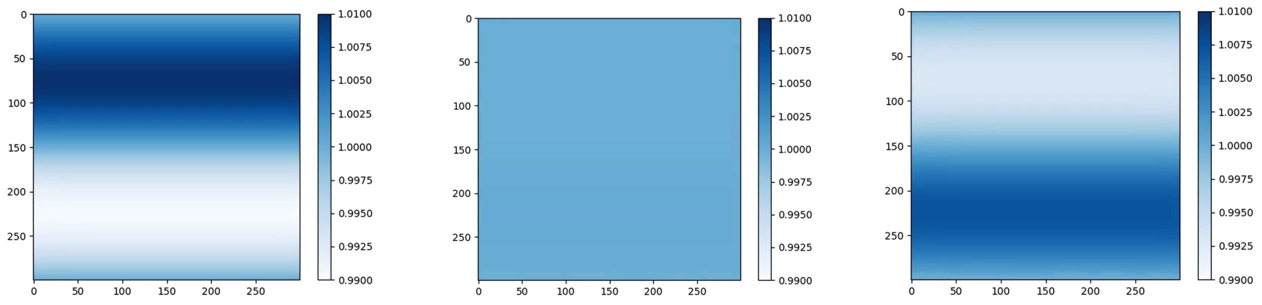


Figure 12: snapshots from the shear wave of a sinusoidal density profile

We studied the dynamic behavior of these waves and their interactions within the fluid having different k-parameter for each experiment ($k_1 = \frac{2.\pi}{L_y}$, $k_2 = \frac{4\pi}{L_y}$, ...). This allowed us to analyze the evolution of the density distribution over time, observing the complex patterns that emerged from the interplay of multiple waves. To enhance our understanding, we extended the simulation to include multiple shear waves.

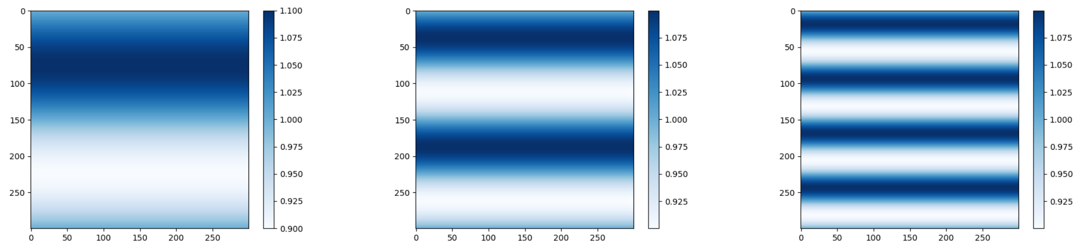


Figure 13: sinusoidal density profiles with different k_i parameter

The decay of shear waves over time dissipates. As the shear wave propagates through the fluid, it encounters resistance and experiences energy losses due to viscosity. This dissipation leads to a gradual attenuation of the wave's amplitude.

The process of shear wave decay can be better understood through a set of figures provided. Figure 14 illustrates the initial sinusoidal velocity profile imposed in the simulation, showcasing the wave's characteristic shape. As time progresses, the amplitude of our shear wave dimishes, resulting in a smoothening and broadening of the wavefront.

Additionally, Figure 15 depicts the relationship between the relaxation parameter and the decay rate of the shear wave. By varying the relaxation parameter, we observe different decay rates, indicating that the relaxation parameter influences the dissipation of energy within the fluid. This correlation supports the notion that the relaxation parameter is related to the fluid's viscosity.

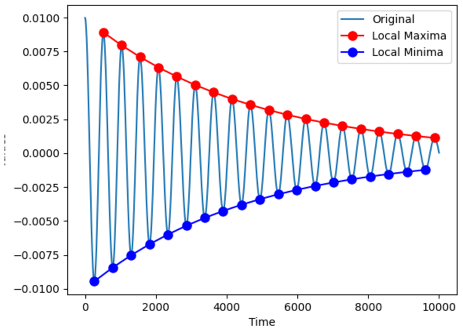


Figure 14: Shearwave amplitude over time

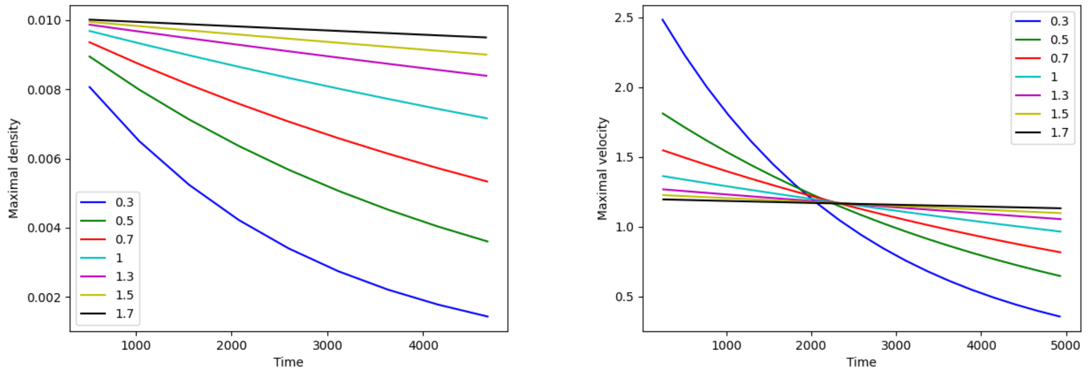


Figure 15: Density and velocity evolution over time (resp. left and right)

In Figure 15, each color represents a different value of ω parameter. For the maximal densities they are the local maximas of each exepriment with a given ω. As for viscosities they are normalized for better understanding, in such way they all meet in a median velocity at a fixed time, this way we can understand better how fast does the shear wave decays (same goes for the velocity plot, each color represents a different value of ω).

Another way to show how the density's amplitude is decaying overtime is projecting our density profile over the Y-axis instead of time. This shows us a how fast the decay goes and also some of the amplitudes both over time and over the position. Figure 16 shows some of the snapshots taken over time.

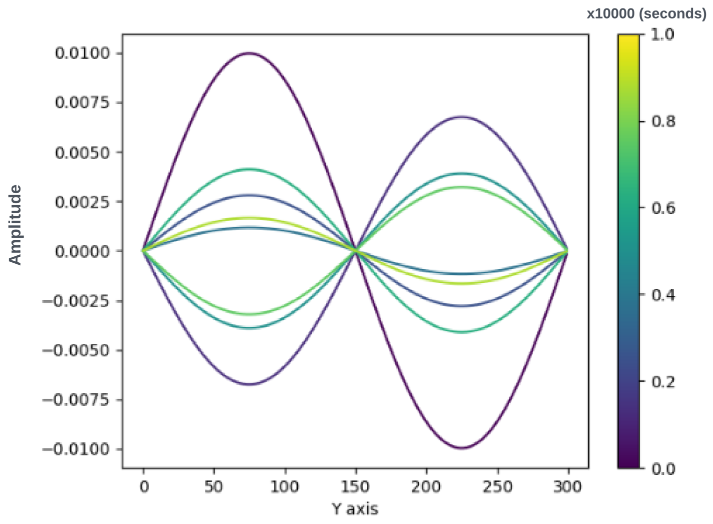


Figure 16: Density's amplitude over the Y-axis, $\omega = 0.7$

Furthermore, we explored the influence of the relaxation parameter on viscosity. By conducting the simulation with different relaxation parameter values, we observed how variations in relaxation affected the decay rate of the shear wave. This enabled us to establish a theoretical relationship between the relaxation parameter and viscosity as seen in the earlier theoretical chapter of shear wave decay,

By using equations (7) and (8), we created a plot showing how close our experiment is to the theoretical prediction. The equation (7) is used to find the velocity paramter v. We've calculated v for each local maxima and saved the mean as the velocity value for the used ω. Equation (19) and figure 17 show our analysis.

$$v = \frac{-ln(\frac{a(t)}{a_0})}{k_1^2.t} \tag{19}$$

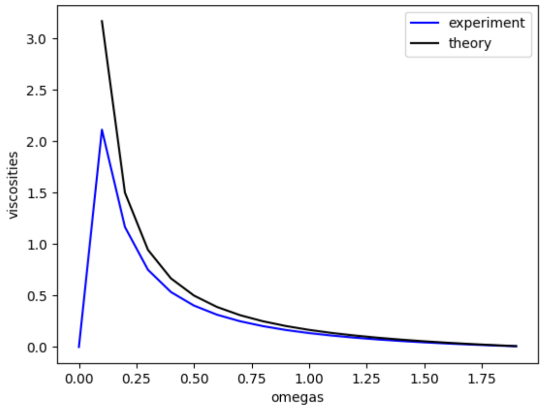


Figure 17: viscosity over omega

When considering the range of the relaxation parameter (ω) in the context of the shear wave decay simulation, it is important to understand the limitations of the Stokes flow assumption. Stokes flow is a simplified model that assumes the fluid's inertia is negligible compared to the viscous forces acting upon it. It provides accurate results when ω is relatively large.

However, as the relaxation parameter (ω) decreases, the simulation deviates further from the Stokes flow regime. The Stokes flow assumption becomes less valid, and the flow exhibits non-negligible inertial effects. In such cases, the dynamics of the fluid become more complex, and the simple Stokes flow equation is no longer sufficient to capture the flow behavior accurately.

4.3.2 couette flow

When it comes to showing the results of the couette flow simulation, we choose to plot the velocity profile as a heatmap. Below in figure 18 we see how velocity evolves through time and how the speed of particles propagates vertically. the two figures show the difference in the gradient. Of course the speed of this propagation depends on two factors : velocity of the wall u_w and the viscosity of the fluid v.

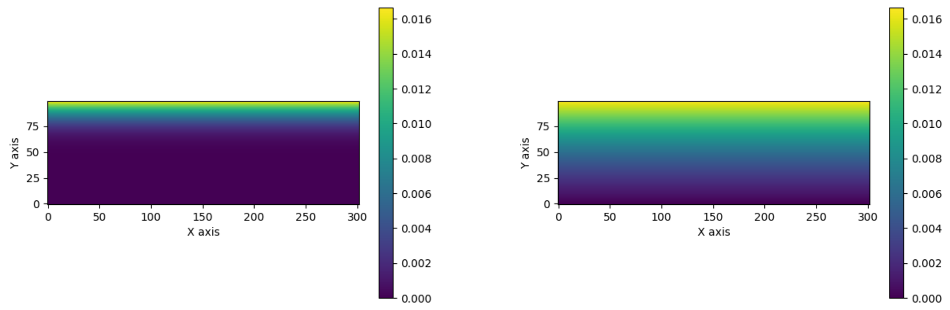


Figure 18: evolution of velocity with time

Another way to manifest this effect is by plotting the norm of the velocity according to the Y-axis, this will show a relaxation (convergence) of the velocity to reach a steady state where the velocity lineary decreases on the Y-axis. Below in figure 19, we save the velocity state on the Y-axis for a number of steps, the theoretical values are also plotted.

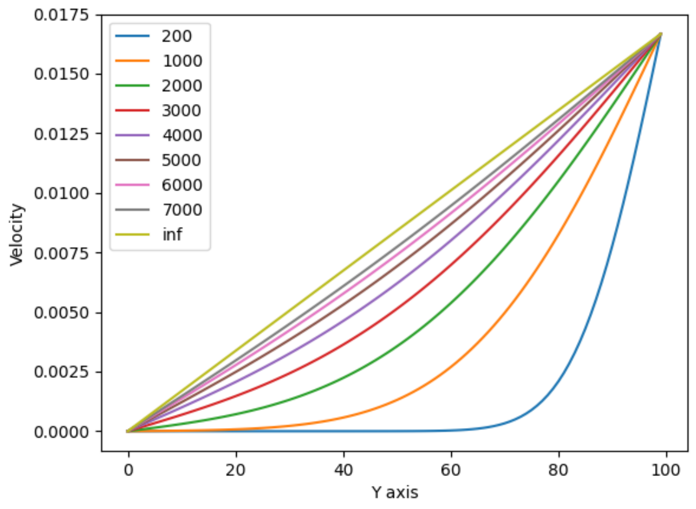


Figure 19: velocity plotted on the Y axis in couette simulation

4.3.3 Poiseuille flow

As seen previously in the theoretical study of the poiseuille flow experiment, two steady and narrow plates work as a 2D pipe for the particles inside. We create a difference in the pression inside and see how our fluid model will react to it.

Below in figure 20, we have three gradual states of our poiseuille flow, the 1^{st}frame is the initial state where the density is uniform in all the pipe. a snapshot of the 500^{th} frames show us how the hyperbolic flow is created in the middle of the pipe (i.e next to the walls the velocity decreases gradually to zero, but in the center of the Y-axis the particles are moving according to the pressure gradient), and at the 1000^{th} frames we see how the velocity keeps increasing and converging to reach it's equilibrium state.

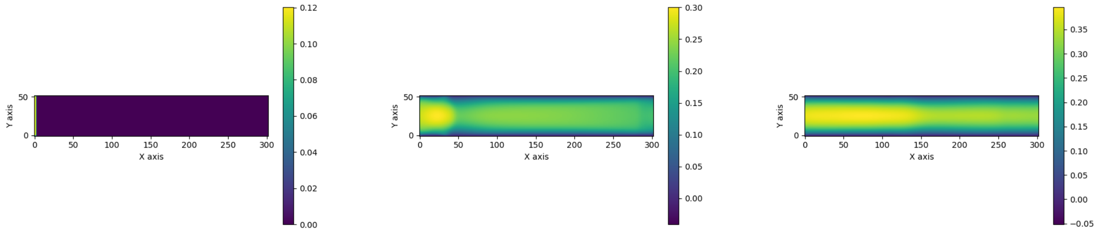


Figure 20: evolution of velocity with time, respectively 1 frame, 500 frames, 1000 frames

To better understand the hyperbolic shapes that are created in this experiment we plot the velocity profile over the Y-axis. In figure 21 we've plotted the velocity profiles for Three close steps showing the convergence of the velocity to an equilibrium. The theoretical plot shows how our experiment is valid and works properly.

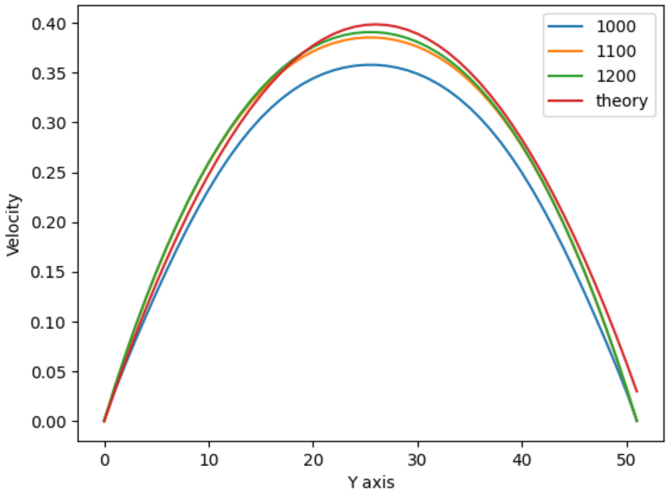


Figure 21: velocity plotted on the Y axis in poiseuille simulation

4.3.4 Sliding lid

The first plot displays the velocity profile as a heatmap. It provides a visual representation of the fluid flow within the simulation domain. The x-axis represents the horizontal position in the grid, while the y-axis represents the vertical position. The color intensity or heatmap depicts the magnitude of the fluid velocity at each point in the grid.

In this specific plot, the heatmap demonstrates the effect of the sliding wall on the fluid flow. As the lid of the box moves with a constant velocity to the right, it creates a flow pattern where the fluid is pushed towards the right wall. This can be observed by the high intensity or hotter colors near the right wall, indicating faster fluid velocity, and cooler colors on the left side, representing slower fluid velocity. This flow pattern highlights the impact of the sliding lid on the fluid motion, resulting in a flow smashing into the right wall.

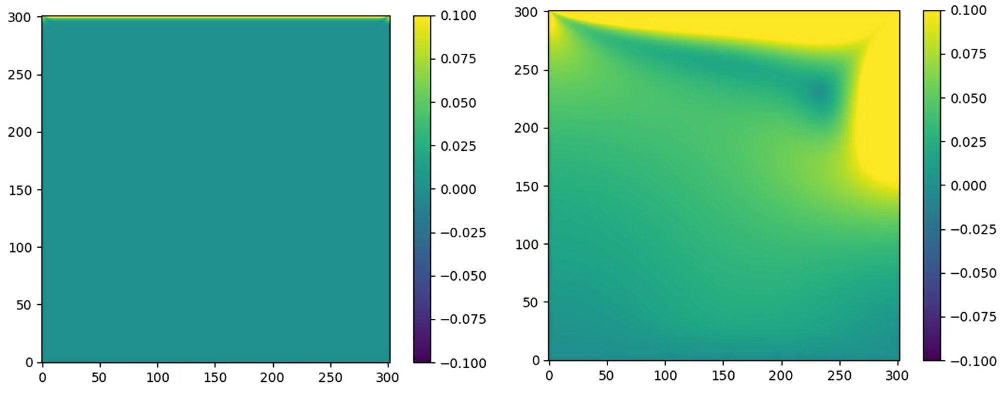


Figure 22: evolution of velocity with time

The second plot showcases a streamplot after 200'000 simulation steps in the streaming process. A streamplot is a visualization technique commonly used to depict the flow field in a fluid simulation. It uses vector lines to represent the direction and magnitude of the fluid velocity at different positions within the grid.

In this plot, the streamplot illustrates the formation of a vortex, which is a swirling flow pattern, due to the sliding lid. The vector lines curve and form circular paths, indicating the rotation of the fluid particles in a coherent manner. The presence of the vortex is a direct consequence of the sliding lid's motion and its influence on the fluid flow.

By examining the streamplot, we gain a clear understanding of the real effect of the sliding lid experiment on the fluid behavior. It visually confirms the creation of a vortex, which is a characteristic flow pattern associated with the sliding lid phenomenon. This information is crucial, as it provides insights into the complex behavior of fluids under specific experimental conditions.

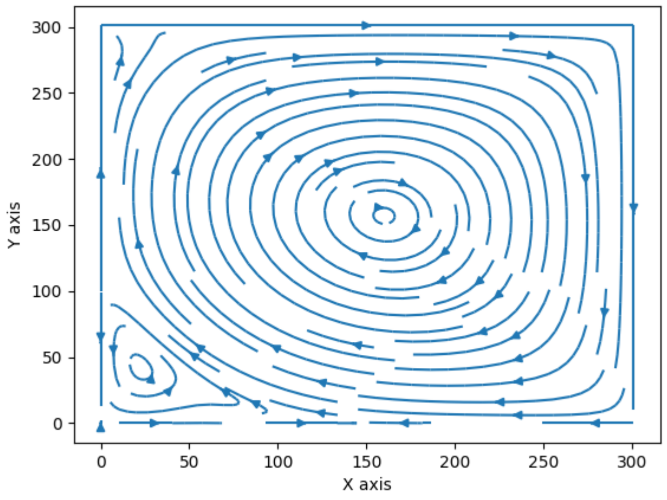


Figure 23: stream lines of the sliding lid experience after 200'000 steps

4.4 Parallelization

In order to put our parallelization process through a integral test, we ran our program with multiple processes and different grid sizes. The figure below 24 shows that thesliding grid parallel algorithm is working properly, with the except of having to save each of the processed cell grid in a reversed manner, a thing we unfortunately couldn't resolve, but the plot prooves the correctness of our approach.

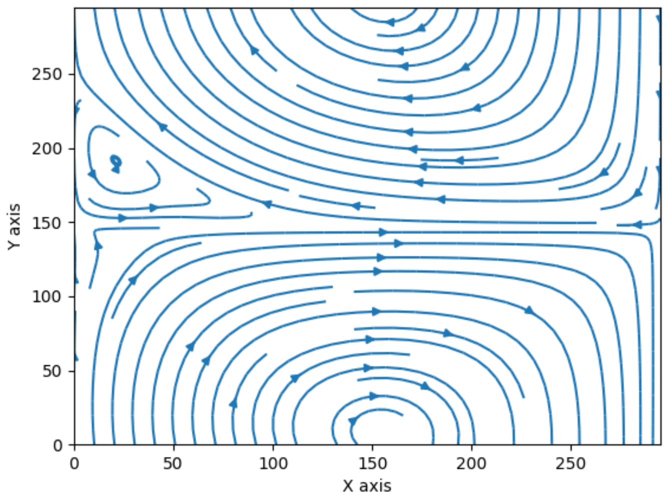


Figure 24: Siliding lid velocity profile run in parallel using 4 processors on a 300x300 grid

4.4.1 Scaling tests

Parallelizing the sliding-lid lattice Boltzmann method (LBM) offers significant advantages in terms of scalability, allowing efficient simulation of fluid flows across large domains and at higher resolutions. The size of the grid and the number of processors play a crucial role in determining the scalability and performance of the simulation. As the size of the grid increases, as seen inthe figure below going from 500 by 500 grid to 5000 by 5000, more lattice points need to be processed, potentially increasing the computational workload per processor. However, with a larger grid, the simulation can capture finer details of the fluid dynamics.

By distributing the grid among multiple processors, in our case up to 100 processor, the workload can be balanced, and the simulation can be performed in parallel, effectively reducing the computation time. As the number of processors increases, the simulation can be further divided into smaller chunks, leading to better load distribution and potentially faster simulations. The parallelization approach allows the sliding-lid LBM simulation to efficiently harness the computational power of multiple processors, resulting in increased Million Lattice Units Per Second (MLUPS) performance metric. As both the grid size and the number of processors scale up, the MLUPS metric can increase, enabling simulations with larger domains and higher resolutions to be executed in practical timeframes. We can clearly see in the figure below that our model scales well with the grid size seeing that the difference of operation per second values between a big grid and a small grid is very high.

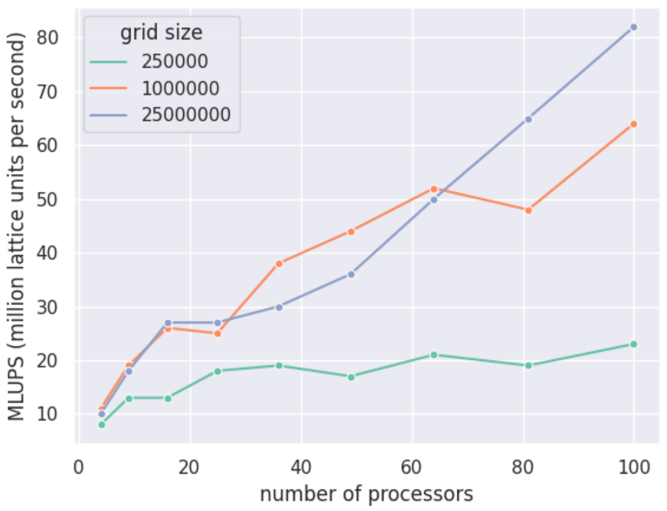


Figure 25: Scaling test on `BW_UniCluster` [3] on different grid sizes and different number of processors

However, the parallel efficiency and overall performance depend on factors such as communication overhead, load balancing, and the specific implementation of the parallelization strategy. Optimizing these factors is crucial to achieving near-linear scalability and fully exploiting the computational resources for sliding-lid LBM simulations.

5 Conclusion

In conclusion, our study of the lattice Boltzmann method using the $2DQ9$ model has provided valuable insights into fluid dynamics and the behavior of particles within a lattice grid. Through a series of experiments, including shear wave decay, Couette flow, Poiseuille flow, and the sliding lid, we have observed how the particles in the fluid adhere to fundamental rules such as the Navier-Stokes equations. This has allowed us to gain a deeper understanding of fluid flow patterns, velocity distributions, and the effects of pressure gradients.

Furthermore, we have conducted computational experiments by running our sliding lid simulation on a cluster of processors. This analysis has provided us with important information on the scalability of our model, demonstrating how it performs when applied to larger lattice units, increased frame counts, and distributed processings. By assessing the efficiency and performance of our model in this manner, we can optimize its implementation and make it more applicable to real-world scenarios.

By employing `mpi4py`, we were able to divide the lattice grid and associated computations among different processors, enabling simultaneous calculations on distinct portions of the grid. This parallel approach significantly reduced the overall computational time required to simulate complex fluid flow scenarios. It allowed us to handle larger lattice units, increase the number of simulation frames, and effectively utilize the available computational resources.

For engineers working in the field of fluid mechanics, understanding the impact of parallelization is of paramount importance. Fluid dynamics simulations often involve complex geometries and intricate flow patterns, which require extensive computational resources to accurately model and analyze. Parallelization techniques and inter-process communication, such as the one we employed with `mpi4py`, enable engineers to tackle these computationally demanding problems effectively.

Overall, the combination of theoretical analysis and experimental validation has enhanced our understanding of lattice Boltzmann simulations and their practical applications in fluid dynamics. Our findings contribute to the development of accurate and efficient computational tools for studying complex flow phenomena and aid in the advancement of various engineering and scientific fields.

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.

[3] the Steinbuch Centre for Computing (SCC) at Karlsruhe Institute of Technology. Bw unicluster parallel computing, https://wiki.bwhpc.de/e/category:bwunicluster_2.0.