Saad DAHLAB University - Blida 1

Institut d'Aéronautique et des Etudes Spatiales

Submitted by:

BOUAKIRA Iheb

Master Thesis

Field: Aeronautics

Option: Propulsion

Evaluation of the Lattice Boltzmann Method using Palabos on benchmark flows

Supervised by: Pr. REZOUG

Acknowledgements

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

I am over helmed in all humbleness and gratefulness to acknowledge my depth to all those who have helped me to put these ideas, well above the level of simplicity and into something concrete.

I would like to express my deep and sincere gratitude to my research supervisor, Pr. REZOUG teacher and Head of Laboratory of Aeronautical Sciences for giving me the opportunity to do this research and providing invaluable guidance throughout it. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance.

I would also like to express my special thanks and gratitude to Dr. HAMIDI, teacher and Head of Construction Department. She played a significant role to the writing of this thesis with her guidance and instructions, which translated her experience and willingness to work hard. In addition, her unwavering availability and empathy throughout the phases leading up to this result encouraged me in moving forward with confidence and clarity.

I am extremely grateful to my parents for their love, prayers, caring and sacrifices for educating and preparing me for my future. I am very much thankful to my brother for his support and guidance in all aspects of life and for committing to his role of older brother.

My Special thanks goes to the second family I made here in Blida, my friends and roommates that became brothers and all people who contributed to this work directly and indirectly.

Abstract:

This project provides a description of the Lattice Boltzmann Method (LBM), an alternate numerical method to conventional CFD methods. For its promising potential, LBM has grown in favor within the scientific community in recent years. It claims to reach the same degree of accuracy as classical CFD while also providing new benefits such as easy parallelization and the ability to perform complex and multiscale flows. Unlike conventional CFD, which works on the numerical solution of the Navier Stokes Equations, the Lattice Boltzmann Method focuses on microscopic particle interactions to reflect the macroscopic behavior of the fluid.

The objectives of this paper is to evaluate and analyze the Lattice Boltzmann Method's capacity to accurately simulate incompressible flows. This study describes the theoretical foundations of this novel method, as well as a CFD simulation of classical configurations. These configurations are developed using modified Palabos codes, with an emphasis on matching LBM solutions with analytical or existing solutions. The results and analyses suggest that LBM is a reliable method for modelling incompressible flows. The study also discusses how to use the Lattice Boltzmann Method and suggests strategies to continue the research.

Key-words: Lattice Boltzmann method, Boltzmann equation, Lattice gas automata, Palabos, Parallelization, Computational Fluid Dynamics, CFD, numerical method, simulation, benchmarking.

Résumé:

Ce projet fournit une description de la méthode Lattice Boltzmann (LBM), une méthode numérique alternative aux méthodes CFD conventionnelles. En raison de son potentiel prometteur, la LBM a gagné en popularité au sein de la communauté scientifique au cours des dernières années. Elle prétend atteindre le même degré de précision que la CFD classique tout en offrant de nouveaux avantages tels qu'une parallélisation aisée et la possibilité de réaliser des écoulements complexes et multi-échelles. Contrairement à la CFD classique, qui fonctionne sur la résolution numérique des équations de Navier Stokes, la méthode Lattice Boltzmann se concentre sur les interactions microscopiques des particules pour refléter le comportement macroscopique du fluide.

L'objectif de cet article est d'évaluer et d'analyser la capacité de la méthode Lattice Boltzmann à simuler avec précision les écoulements incompressibles. Cette étude décrit les fondements théoriques de cette nouvelle méthode, ainsi qu'une simulation CFD de configurations classiques. Ces configurations sont développées à l'aide de codes Palabos modifiés, en mettant l'accent sur la correspondance entre les solutions LBM et les solutions analytiques ou existantes. Les résultats et les analyses suggèrent que la LBM est une méthode fiable pour la modélisation des écoulements incompressibles. L'étude aborde également la manière d'utiliser la méthode Lattice Boltzmann et suggère des stratégies pour poursuivre la recherche.

Mots clés: Méthode de Lattice Boltzmann, équation de Boltzmann, Lattice gas automata, Palabos, parallélisation, dynamique des fluides numérique, CFD, méthode numérique, simulation, benchmarking.

الملخص:

يقدم هذا المشروع وصفًا لطريقة Lattice Boltzmann (LBM) ، وهي طريقة رقمية بديلة لطرق CFD التقليدية. نظرًا لإمكانياتها الواعدة ، نمت LBM لصالح المجتمع العلمي في السنوات الأخيرة. تدعي أنها تصل إلى نفس درجة الدقة مثل CFD الكلاسيكي مع توفير مزايا جديدة مثل الموازاة السهلة والقدرة على أداء التدفقات المعقدة والمتعددة النطاقات. على عكس CFD التقليدي ، الذي يعمل على الحل العددي لمعادلات Navier Stokes ، تركز طريقة Lattice Boltzmann على تفاعلات الجسيمات المجهرية لتعكس السلوك العياني للسائل.

تهدف هذه الورقة إلى تقييم وتحليل قدرة طريقة Lattice Boltzmann على محاكاة التدفقات غير القابلة للضغط بدقة. تصف هذه الدراسة الأسس النظرية لهذه الطريقة الجديدة ، بالإضافة إلى محاكاة CFD للتكوينات الكلاسيكية. تم تطوير هذه التكوينات باستخدام رموز Palabos المعدلة ، مع التركيز على مطابقة حلول LBM مع الحلول التحليلية أو الحالية. تشير النتائج والتحليلات إلى أن LBM طريقة موثوقة لنمذجة التدفقات غير القابلة للضغط. تناقش الدراسة أيضًا كيفية استخدام طريقة Lattice Boltzmann وتقترح استراتيجيات لمواصلة البحث.

الكلمات المفتاحية: طريقة Lattice Boltzmann ، معادلة Boltzmann ، أتمتة الغاز الشبكي ، Palabos، Parallelization ، ديناميكيات السوائل الحسابية ، CFD ، الطريقة العددية ، المحاكاة ، القياس المعياري.

List of figures

Contents

Nomenclature

Symbol	Significance	Unity
Re	Reynolds number	[-]
u	Velocity	[m/s]
ρ	Density	[kg/m^3]
V	Volume	[m^3]
C_L	Lift coefficient	[-]
C_D	Drag coefficient	[-]
F_L	Lift force	[N]
F_D	Drag force	[N]
V_i	Finite volume	[m^3]
x_i	Position on the *i*th direction	[m]
e_i	Local velocity of the particle	[m/s]
f	Distribution function	[kg.s^3.m^{-6}]
f^{eq}	Equilibrium distribution	[kg.s^3.m^{-6}]
f^{neq}	Non-equilibrium distribution	[kg.s^3.m^{-6}]
Ω	Collision operator	[kg.s^2.m^{-6}]
ξ	Particle velocity	[m/s]
t	Time	[s]
h	Spatial resolution	[m]
τ	Relaxation time	[s]
ε	Knudsen number (smallness number)	[-]
f_i	Discrete distribution function	[-]
f_i^*	Discrete distribution function after collision	[-]
c_i	Lattice velocity	[m/s]
w_i	Weighting function	[-]
$\Delta x, \Delta y, \Delta z$	Lattice spacing	[-]
c_s	Speed of sound	[m/s]

General introduction:

Computational Fluid Dynamics (CFD) is a fundamental fluid mechanics technique that uses numerical analysis and data structures to analyze and solve a wide range of fluid flow problems. This well-established solution is now broadening its scope, addressing anything from environmental issues to heart diseases and aeroacoustics.

Because conventional methods are incapable of simulating all possible situations, a variety of novel ways are emerging. A new approach to solving flow problems in the field of CFD has gained popularity in recent years; the Lattice Boltzmann Method (LBM). And the growing number of devoted articles is just one sign of this increasing popularity.

Instead of focusing on the macroscopic representation of the flow, LBM incorporates a microscopic representation of the flow as well as a spatial and temporal discretization. The approach simulates fluid physics by adopting simple physical processes including streaming in space and billiard-like collision interactions between microscopic particles. LBM is a model that operates on the premise that a fluid is made up of interacting molecules that can be described using classical mechanics.

The objective of this thesis is to evaluate the LBM-BGK code's ability to simulate incompressible fluid flows based on well-referenced conventional CFD configurations. To do this, the work is organized into four chapters.

The first chapter is devoted to putting into perspective the impact that computer performance development has played in the development of CFD as an alternative to often restrictive analytical and experimental methods. Then, in order to place the LBM within the broad spectrum of numerical methods, a classification of the primary CFD approaches is established.

The second chapter is devoted to the theoretical and mathematical understanding of LBM, as the key concepts of the method are developed and the governing equations demonstrated. The discretization of the Boltzmann equation will then result in the establishment of a mathematical model defining the physics governing the simulated processes. And therefore, enable the construction of a link between a molecular description and the fluid's macroscopic behavior, which remains the primary focus.

The third chapter discusses the transition from a set of equations to a code capable of simulating a wide range of complex phenomena. Given that the current LBM codes, available in open-source libraries, are written in C++, the most attractive features of the latter are highlighted. Before being able to establish the many techniques used by Palabos to develop a computational code that is simultaneously optimized in terms of memory consumption, effective in its layout of simulation elements, and robust enough to model cases with complex physics that conventional approaches struggle to simulate.

Finally, the fourth chapter focuses on the application and validation of Palabos. Following that, a full report on the actions required to take the first steps in Palabos is established. Then, codes for simulating two flows, namely the Poiseuille flow and the flow around a cylinder, are run to collect data that will be compared with the results available in the literature and used as a criterion for evaluating the reliability of the results obtained by the LBM-BGK model using the Palabos solver.

Chapter I: Computational Fluid Dynamics, methods, and solvers

1. Introduction

The evolution of the high-speed digital computer mostly during the past century had a significant impact on how fluid mechanics and heat transfer principles are applied to design problems of contemporary engineering practice. Problems that would have taken years to solve using the computational methods and processors available 30 years ago can today be solved in a few seconds of computer time at a very low cost. Many advances have been driven by the ready availability of previously inconceivable computational power. These first became apparent in industry, research institutions and laboratories, where it was pressing to solve different difficult problems. Nowadays, computer-induced changes have recently become apparent and obvious in almost every aspect of our daily life. [1]

Fluid mechanics equations that have been known for over a century can only be solved for a small number of flows. Analytical flow solutions are only attainable under specific limits and for a limited number of geometries because the behavior of these analytical solutions is so complex. The nonlinearity of the equations, as well as the presence of complex-shaped boundary conditions, make finding analytical solutions extremely difficult, if not impossible in some cases. The known solutions are tremendously helpful in understanding fluid movement, but they are rarely applicable to engineering analysis or design. Engineers have been compelled to adopt different methods in the past.

2. The theoretical, experimental and numerical approaches

The most typical approach is to use simplified versions of the equations. The majority of these are based on a combination of approximations and dimensional analysis, with

empirical input nearly always required. Dimensional analysis, for example, reveals that the drag force on an object can be represented by: $F_D = \frac{1}{2}.\rho S C_D V^2$

A similar method has emerged after noting that for many flows, non-dimensionalization of the Navier-Stokes equations leaves the Reynolds number as the only independent parameter. If the body shape is kept constant, an experiment on a scale model of that shape can yield the desired findings. The required Reynolds number is obtained either by careful selection of the fluid and flow parameters or through Reynolds number extrapolation. These methods are extremely helpful, and they continue to be the primary methods of practical engineering design today. The issue is that many flows require multiple dimensionless parameters to be specified, and it may be impossible to build up an experiment that scales the actual flow correctly. Flows around aircraft or ships are two examples.

Another solid approach was the experimental method, as experiments are a useful tool for determining global parameters such as lift, drag, pressure drop, and heat transfer coefficients. Details are essential in many circumstances; it may be necessary to know whether flow separation happens or whether the wall temperature surpasses a certain level. Experimental development may be excessively costly and/or time demanding as technological advancements and competitiveness involve more rigorous design optimization. Alternatively, when new high-tech applications necessitate flow prediction for which the database is insufficient.

Experiments are difficult, if not impossible, in other situations. For example, the measuring equipment might interfere with the flow, or the flow could be inaccessible. Some quantities are just not measurable or can only be measured with insufficient precision using current techniques. [2]

Traditionally, experimental and theoretical methods have been used to produce designs for fluid flow and heat transfer equipment and vehicles. But with the advent of electronic computers, an alternative - or at least a complementary method - became available: the

numerical approach (although many of the key ideas for numerical solution methods for partial differential equations were established more than a century ago, they were of little use before computers appeared). When complex flows are involved, experimentation is still vital, but there is a definite trend toward relying more on computer-based predictions in design. [1]

Since the 1950s, the performance-to-cost ratio of computers has exploded and shows no signs of slowing down. While the first computers developed in the 1950s could only execute a few hundred operations per second, today's machines can perform teraflops or 10^{12} floating-point operations per second. The ability to store data has also increased dramatically: a decade ago, ten-gigabyte hard discs were only available on supercomputers; now, we can find hard drives with incredible speeds and capacities measured in Terabytes on desktop PCs. A system that cost millions of dollars, took up a lot of space and required a full-time maintenance and operation team is now available on a PC.

The prevalence and use of numerical methods skyrocketed as the potential of computers were realized. The computational solution of fluid mechanics equations has become so essential that it currently attracts the attention of the vast majority of fluid mechanics researchers, and the proportion is growing. Computational fluid dynamics is the name for this field (CFD), which contain numerous subspecialties within it. [2]

3. What is CFD?

Computational fluid dynamics (CFD) is a science that uses digital computers to generate quantitative predictions of fluid-flow phenomena based on the conservation laws (mass, momentum, and energy conservation) that govern fluid motion. [3]

The governing equations for practical flows are frequently so complex that an exact solution is impossible to find, therefore a computer solution is required. Computational approaches substitute the governing partial differential equations with algebraic equations, allowing a computer to solve the problem. For local methods, like the finite difference, finite element and finite volume methods, a grid of discrete points are distributed throughout the computational spatial and temporal domain. Furthermore, the algebraic equations connect values of dependent variables at neighbouring grid points.

The number of grid points necessary to obtain an accurate solution is often determined by the dimensionality, geometric complexity, and severity of the dependent variables' gradients. A grid of ten million points may be necessary to model the flow around an entire aircraft. Each dependent variable, as well as a few auxiliary variables, must be stored at each grid point. when it comes to turbulent compressible three-dimensional flow, somewhere between five and thirty dependent variables per grid point could be involved.

Because most classes of fluid dynamics have nonlinear governing equations, the computational solution is typically performed iteratively. That is, the discretized equations are used to successively correct the solution for each dependent variable at each grid point. The iterative approach is often comparable to progressing the solution over a small time step. The number of iterations (or time steps) might range from a few hundred to thousands.

As long as the discrete equations are correct representations of the governing equations, the discretization method induces an error that can be minimized in principle by refining the grid. If the numerical algorithm that performs the iteration or advances in time is also

stable, then by refining the grid, the computational solution can be made arbitrarily close to the exact solution of the governing equations. [4]

4. Classification of numerical methods

A wide range of methods for obtaining fluid flow solutions has been developed to this point. Some of these approaches are general-purpose methods that may be used to any partial difference equation (PDE) with minimal modifications. Other approaches are more suited to finding fluid flow solutions. While this thesis focuses on the lattice Boltzmann method, it is just one of several methods available nowadays. Each of these methods has its own set of advantages and disadvantages, and the LB method is no different. Taking that into consideration, it becomes relevant to briefly present such methods in order to give a perspective on where the Lattice Boltzmann method is situated in the methods' landscape.

Despite the existence of a large number of simulation methods, they can be classified into two categories according to the approach taken toward the given problem: *conventional methods* and *particle-based methods.*

4.1. Conventional Navier-Stokes solver

the general idea is to solve the equation (or coupled system of equations) of significance by using a particular method of approximation. The continuity equation and the Navier-Stokes equations (or their incompressible counterparts) are the two fundamental equations to solve in CFD. Depending on the physics to be simulated and the applied approximations, other equations, including an energy equation and an equation of state, may be used to supplement the fundamental equations.

Typical approaches for unsteady (i.e. time-dependent) CFD can use a variety of methods to discretize the derivatives, allowing the aforementioned equations to be approximated on a computer. As a result, the solution for the next time step is generated from the present time step's solution. However, the method used to discretize the solution, i.e. how a finite set of numbers is used to represent the solution in continuous physical space, is

what distinguishes these conventional CFD methods. The solution variables, such as fluid velocity *u* and pressure *p,* must be represented in such a way that their spatial derivatives can be obtained throughout the domain in all of these methods.

This process of discretisation leads to matrix equations $Ax = b$ for many, if not all, conventional approaches, where **A** is a matrix that connects the unknown discretised solution variables in the vector **x**, and **b** represents the impact of source terms and boundary conditions. Resolving such matrix equations by inverting A to determine x is a linear algebra problem, and finding efficient solution methods for such problems has been the subject of much research. In the following section, some of these methods' basics are covered:

4.1.1. Finite difference method

The basis of this method is to approximate derivatives of λ using linear combinations ("finite differences") of λ_j . ". After performing a Taylor series of λ_j about x_j, we can find three simple approximations of the first-order derivatives:

$$\frac{\partial \lambda}{\partial x}|_{x_j} \approx \frac{\lambda_{j+1} - \lambda_j}{\Delta x} \ , \ \frac{\partial \lambda}{\partial x}|_{x_j} \approx \frac{\lambda_{j+1} - \lambda_{j-1}}{2\Delta x} \ , \ \frac{\partial \lambda}{\partial x}|_{x_j} \approx \frac{\lambda_j - \lambda_{j-1}}{\Delta x} \quad (1.1)$$

In concept, the finite difference approach is straightforward; simply take a set of equations and replace the derivatives with finite difference approximations. Fluids, on the other hand, are regulated by a complex system of linked equations with several variables. As a result, in order to apply the FD method for CFD, a variety of unique techniques must be used [5], which increases the amount of expertise and effort required to develop an FD CFD solver. Nevertheless, when compared to other conventional methods, the FD method can be simple and efficient [2].

Certain mathematical errors are inherent with FD CFD. Unless extra precautions are taken, the scheme is not conservative, which means that numerical errors allow the conservation of quantities including mass, momentum, and energy to be compromised. The FD approach has difficulties with complex geometries that do not fit the grid because it is built on a regular grid. This is perhaps the most crucial reason why other CFD methods have grown in popularity [6].

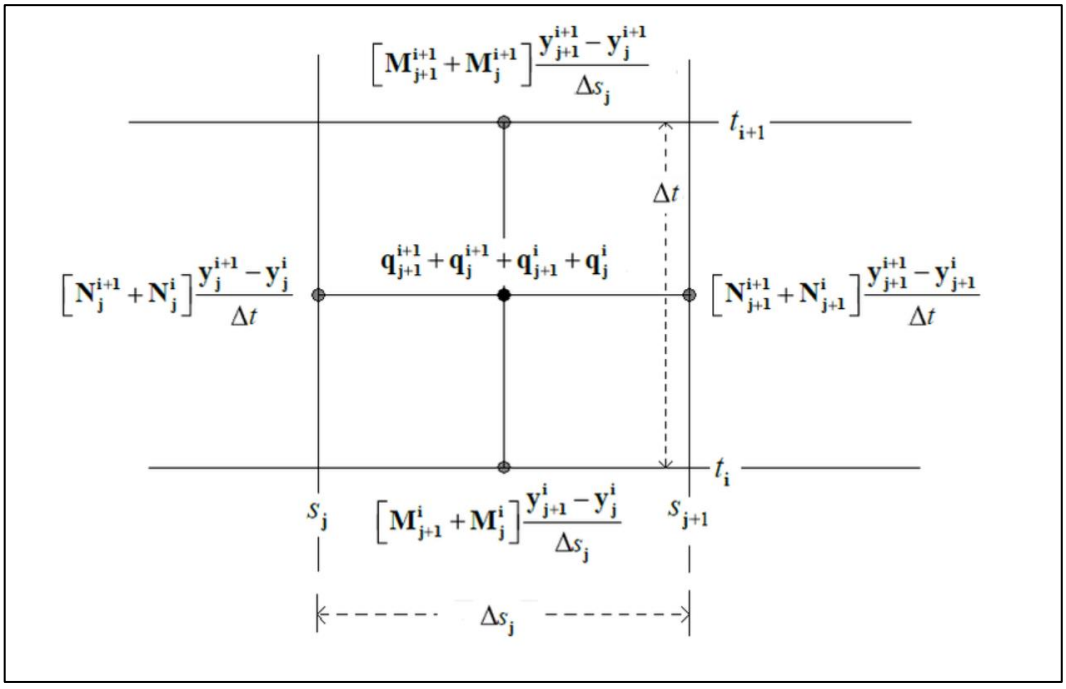

Figure 1 : Illustration of the finite difference method implementation [7]

4.1.2. Finite volume method

In the FV method, the spatial domain would not need to be partitioned into a regular grid. Instead, the simulated volume V is divided into multiple smaller volumes V_i each with a different shape and size. This provides for a more accurate representation of complex geometries than, for example, the finite difference method. Each finite volume V_i has a node in the centre where each solution variable $\lambda(x)$ is represented by its approximate average value λ_i inside that volume.

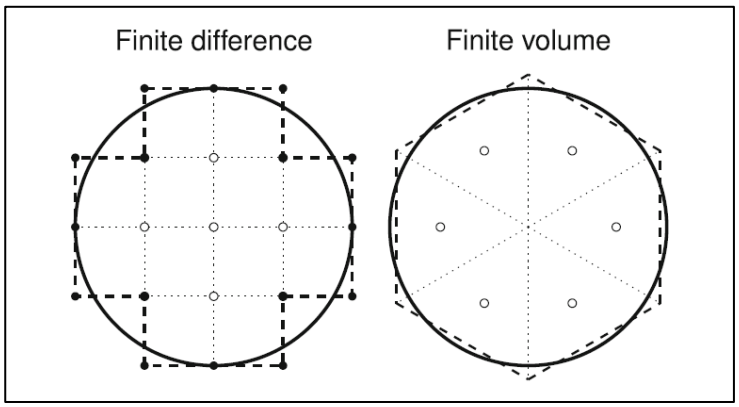

Figure 2: Simple finite difference and finite volume discretisations of the volume inside a circular surface. The effective surface in each case is shown as black dashed lines, and interior nodes as white circles. To the right, the dotted lines show the finite volumes' interior edges [6]

The FV method is not as general as the FD method, which may be used to any equation. Whereas it could be used to solve generic hyperbolic problems, it is primarily designed to solve conservation equations found in fluid mechanics [8]. This is primarily due to the fact that this approach is conservative by design, suggesting that, as opposed to the FD method, mass and momentum will always be conserved.

Furthermore, the FV method is well-suited to use with irregular grids, which implies that complex geometries may be represented well (the grid is adjusted to the geometry), and it is simple to "invest" more resolution on crucial locations in the simulation by refining the grid in these parts. The disadvantage of irregular grids is that creating adequate grids for complex geometries is a rather complex task in itself; indeed, it is an entire topic of study and research. While FV may not be as general as FD in the solvable equations, this is usually not a problem for the equations encountered in CFD.

4.1.3. Finite element method

PDEs are solved using finite element methods (FEM) by multiplying the PDE by a weight function *w(x)* and integrating over the particular domain. In general, FEM can be applied over an unstructured grid, with a discretised solution variable λ_i represented at every grid corner node x_i. The variable $\lambda(x)$ is interpolated between the grid corners using basis

functions $\phi_i(x)$ that meet specific conditions. The most basic 1D basis functions are linear functions of the form: $\phi_i(x_i)=1$, $\phi_i(x_{j\neq i})=0$ and are non-zero only in the interval (x_{i-1}, x_{i+1}). Moreover, a wide range of nonlinear basis functions (e.g., quadratic and cubic ones) are also possible, and the order of accuracy is often linked to the order of the basis functions [9].

Typically, the weighting functions are designated from the basis functions themselves, $w(x)=\phi_i(x)$. This results in a system of equations, one for each unknown value λ_i. Each value of λ_i is connected to λ_{i-1} and λ_{i+1} via integrals, assuming linear basis functions. The fundamental advantage of FEM is that it is mathematically well-suited for unstructured grids and for improving the order of accuracy using higher-order basis functions (yet these also require more unknowns). These grids can be modified dynamically to adjust for dynamic geometry, as when modelling a car crash. FEM, like FD methods, is not conservative by default, whereas FV methods are. Another drawback is the method's complexity in comparison to the FD and FV methods. In general, unstructured grids, for example, the integrals become difficult to solve. And like with the FD and FV methods, solving the complex Navier-Stokes equation system is not simple.

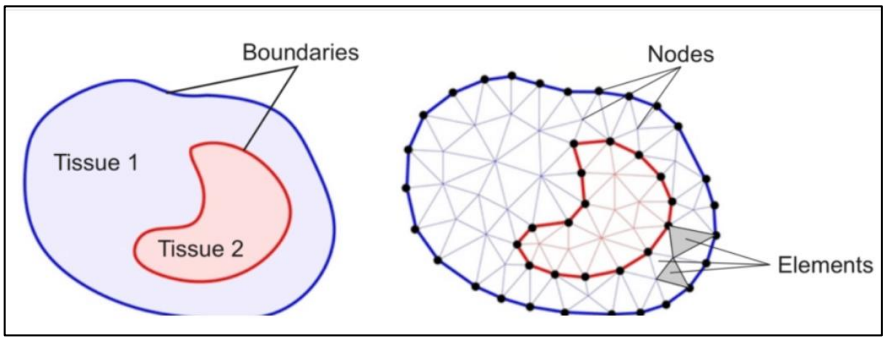

Figure 3: Schematic representation of a finite element method (FEM) model [10]

4.2. Particle-based solvers

Particle-based solvers are not built on directly discretizing fluid dynamic equations, and therefore use a differentiated perspective than the conventional solvers previously discussed. Instead, these methods use particles to represent the fluid. A particle can represent an atom, a molecule, a group of molecules, or a portion of the macroscopic fluid, depending on the approach. Consequently, whereas conventional Navier-Stokes solvers take a macroscopic perspective of a fluid, particle-based method typically adopt a microscopic or mesoscopic view.

This group of approaches includes the Lattice Boltzmann method. These will be elaborated on in the section that follows:

4.2.1. Molecular dynamics

At its core, molecular dynamics (MD) is an essentially simple microscopic method for tracking the position of particles that commonly represent atoms or molecules. These particles interact via intermolecular forces $f_{ij}(t)$, which are determined to be as consistent with the actual physical forces as possible. Knowing the total force $f_i(t)$ on the ith particle based on all other particles, its acceleration is determined using Newton's second law:

$$\frac{d^2 x_i}{dt^2} = \frac{f_i}{m_i} = \frac{1}{m_i} \sum_{j \neq i} f_{ij} \quad . \quad (1.2)$$

The particle position x_i is then numerically updated by integrating Newton's equation of motion. While there are numerous integrating methods, the Verlet algorithm is a particularly simple and effective one:

$$x_i(t + \Delta t) = 2x_i(t) - x_i(t - \Delta t) + \frac{f_i(t)}{m_i} \Delta t^2 \quad (1.3)$$

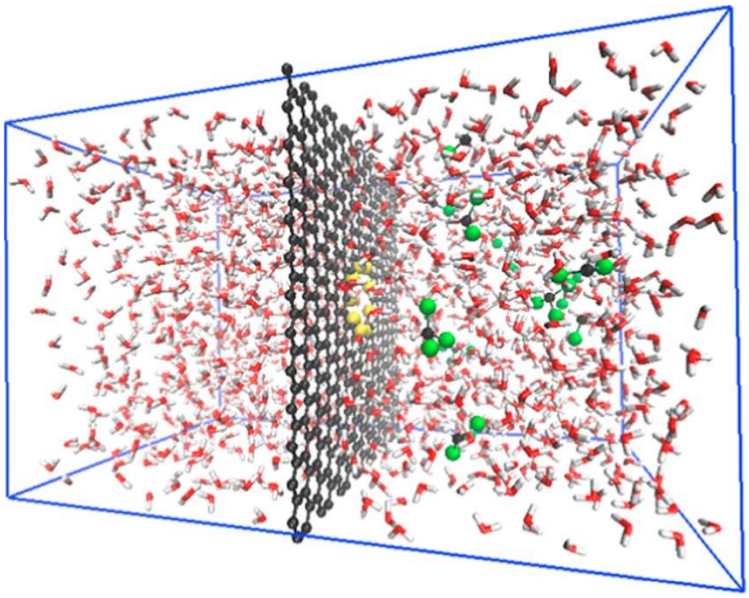

Figure 4: Molecular dynamics simulation. [11]

The present and previous positions of a particle are used in this scheme to determine its next position. The Verlet scheme can alternatively be expressed in terms of using the particle's previous velocity rather than its previous position [12]. While MD is an excellent method for simulating microscale phenomena including phase changes, chemical reactions and protein folding, a numerical method that tracks individual molecules is considerably too detailed for macroscopic phenomena—consider that a single gram of water involves over 10^{22} molecules. As a consequence, MD as a Navier-Stokes solver is completely inconvenient, and more appropriate methods should be selected for this application [13].

4.2.2. Lattice gas models

Hardy, Pomeau, and de Pazzis developed lattice gas models as an exceedingly simple model of 2D gas dynamics in 1973. Their particular model was later called the HPP model after its authors. In this approach, hypothetical particles exist on a square lattice where they stream forward and collide in a form that, quite like molecules in a real gas, preserves

mass and momentum conservation. Considering that the HPP lattice was square, each node had four neighbors, and each particle had one of the four possible velocities c_i that would transport a particle to a neighboring node in a one-time step.

However, Frisch, Hasslacher, and Pomeau did not publish a lattice gas model which can be used to simulate fluids until 1986. Their model was also called the FHP model after its authors. The difference between this model and the original HPP model is minor but significant: Instead of the HPP model's square lattice and four velocities, the FHP model had a triangular lattice with six velocities ci. This modification provided the model with enough lattice isotropy to perform fluid simulations [14].

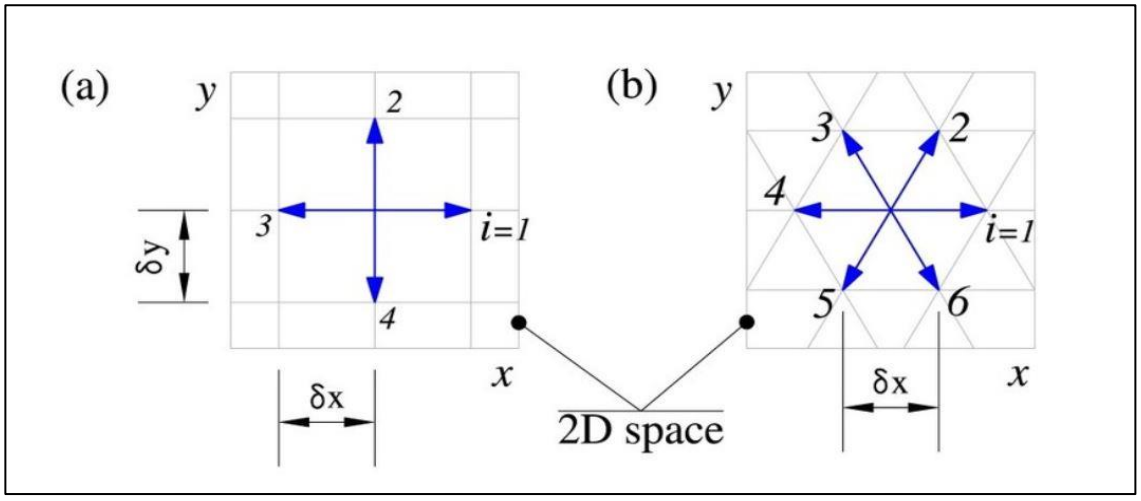

Figure 5:Two-dimensional lattices for the Lattice Gas Automata: (a) Square lattice for HPP model, (b) Equilateral triangular lattice for FHP model [15].

Lattice gas Automata is constructed as a fictitious and simplified molecular dynamics where time, space and velocity are all discrete. The domain consists of a regular network with particles residing in the nodes. A set of Boolean variables (in computer science this represents a variable that can take two possible states; "true" or "false" for example) to describe the occupancy of the particle is defined: $n_i(x,t)$ such that $(i = 1,\ 2,\ .....\ ,\ \mathrm{M})$ and M being the number of directions of the particle velocities at each node. The evolution equation for lattice gas automata is then:

$$n_i(x + e_i\ ,\ t+1) = n_i(x,t) + \Omega_i(n_i(x,t)) \qquad (1.4)$$

Where:

- e_i are the local velocities of the particle and
- Ω_i is a collision operator.

This equation implies and dictates that from an initial state with a given configuration of spaces (translated into nodes x_i) and time, the configuration of the particles will evolve at a time (t+1) in two sub-steps:

- *diffusion:* where each particle will move towards the nearest node in the direction of its velocity, hence the term $x+e_i$
- *collision:* which occurs when the particles arriving at a node interact and change the direction of their velocity according to the rules of diffusion. For simplicity, the exclusion principle is imposed and excludes that there is more than one particle allowed at a given time and at a node with a given velocity [16].

One of the promoted characteristics of lattice gas models is that the occupancy numbers n_i are Boolean variables (particles are either present or absent), implying that collisions are perfect: Roundoff error in floating-point operations performed in different CFD methods has no effect on lattice gas models. Furthermore, lattice gases can be extensively parallelized. However, the disadvantage of these collisions is that they become out of control as the number of velocities increases. In a node, for example, there are $2^{24}=16.8*10^6$ potential states for a three-dimensional lattice gas with 24 velocities. In this approach, the resolution of any collision was often determined by a search in a massive table created by a specialized programmer.

4.2.3. Lattice Boltzmann method

The main problem with lattice gases was statistical noise. At the microscopic level, lattice gases, like real gases, are bursting with activity. Even for gas in equilibrium, reducing the control volume leads the density (mass per volume) inside it to fluctuate even more severely over time: molecules are constantly moving in and out, thereby the law of large numbers holds less true for smaller volumes. The advent of the lattice Boltzmann method

in the late 1980s provided a more extensive solution to the problem of statistical noise. This method was first introduced by tracking the expected value $f_i = \langle n_i \rangle$ of the occupation number rather than the occupation number itself, therefore removing statistical noise by rather taking a mesoscopic approach. This was the initial way of obtaining the LBM; lattice gases track the behavior of concrete particles, whereas the LBM tracks their distribution.

In chapter 2, the technical aspects as well as the mathematical formalism of the Lattice Boltzmann method will be developed. For the time being, we will just discuss its advantages and disadvantages in comparison to the large range of available methods.

- Efficiency and simplicity:

Advantages:

- The LBM is comparable to pseudocompressible methods for solving the incompressible Navier-Stokes equation, which promotes simplicity and scalability by permitting artificial compressibility [17]
- The LBM, like pseudocompressible methods, does not incorporate the Poisson equation, which can be difficult to solve given its non-locality [17].
- The LBM's costliest calculations are local, i.e. constrained to within nodes, which further improves its parallelizability [18]

Disadvantages:

- LBM requires a lot of RAM. A huge number of memory access events are required for population propagation.
- Because the LBM is essentially time-dependent, it is impractical for modelling steady flows.

- Geometry

Advantages

  - The LBM excels in simulating mass-conserving flows in complex geometries such as porous media [18].
  - Moving boundaries that conserve mass are particularly successfully implemented in the LBM, making it an appealing tool for soft matter simulations [19].

- Multiphase and multicomponent flow:

  Advantages:

  - For the LBM, a variety of multiphase and multicomponent approaches are available.
  - Combined with the LBM's strengths in complex geometries, it implies that it is well adapted to reproduce multiphase and multicomponent flows in complex geometries.

  Disadvantages:

  - In multiphase and multicomponent simulations, the range of viscosities and densities is limited.

- Sound and Compressibility:

  ***Advantages*:**

  - Because the LBM is a (weakly) compressible Navier-Stokes solver, it may be well-suited for simulating phenomena involving sound and flow, such as aeroacoustic sound generating.

Disadvantages:

- The LBM is not adequate for directly simulating long-range sound propagation at realistic viscosities.
- The LBM is not appropriate for modelling very compressible (i.e. transonic and supersonic) flows.

5. Conclusion:

While the lattice Boltzmann method offers significant advantages, it is not well suited for all possible applications, as are all other numerical methods for fluids. However, the LBM is a relatively recent method that is still developing at a quick rate, which indicates that the spectrum of applications to which it may be used effectively is expanding.

Chapter II: Theoretical and numerical aspects of LBM

1. Introduction

This chapter focuses on the establishment of the mathematical formalism required to comprehend the Boltzmann equation. This equation should be discretized before it can be used as a mathematical model to describe the underlying physics adopted by the Lattice Boltzmann method in its numerical calculations.

Therefore, the first section will be dedicated to introducing the kinetic theory of gases, which is the cornerstone of the LBM. The Boltzmann equation is then presented in detail, with its various terms and forms.

The second section demonstrates mathematically the transition from the Boltzmann equation with its continuous variables to an equation discretized in velocity space, physical space and time. This introduces the patterns, which are then followed by the method for performing the iterative calculations required to simulate particular phenomena.

2. Kinetic theory and the Boltzmann equation

2.1. Kinetic theory of gases

In the context of fluids, we can consider three levels of description: microscopic, mesoscopic and macroscopic. Microscopic systems denote a molecular description based on Newton's dynamics. Macroscopic systems denote a fully continuum picture with tangible quantities such as fluid density and velocity that appear in the governing Navier-Stokes equations [20]. In between, there is the "mesoscopic" description, which does not track individual molecules, but rather distribution or representative collections of molecules.

The kinetic theory of gases attempts to explain the microscopic properties of a gas in terms of the motion of its molecules. The gas is assumed to consist of a large number of identical, discrete particles called molecules, a molecule being the smallest unit. Maxwell, Boltzmann and Clausius developed elements of kinetic theory between 1860-1880's. Kinetic theories are available for gas, solid as well as liquid.

This theory is the mesoscopic fluid description on which the LBM is based. The main variable in it theory is the ***distribution function.*** It can be regarded as a more generalized case of density ρ. We know that the density represents the density of mass in a physical space, noted $\rho(x,t)$, while the distribution function, noted, $f(x,\xi,t)$, represents the density of mass in both three-dimensional space and three-dimensional velocity space. More fundamentally, it can be defined as the probability of finding a particle with velocity between $\vec{\xi}$ and $\vec{\xi}+\overrightarrow{d\xi}$ within a volume $\overrightarrow{dx}$ around a position x.

In dimension D, the distribution function is expresses in $M.L^{-2D}.T^{D}$. For example, in a 3D space:

$$[f] = Kg \times \frac{1}{m^3} \times \frac{1}{(m/s)^3} = \frac{Kg.s^3}{m^6}$$

For: $D=1 \left\{ d\xi = d\xi_x \right\} \left\{ d\vec{x} = dx \right\}$

$D=2 \left\{ d\xi = d\xi_x.d\xi_y \right\} \left\{ d\vec{x} = dx.dy \right\}$

$D=3 \left\{ d\xi = d\xi_x.d\xi_y.d\xi_z \right\} \left\{ d\vec{x} = dx.dy.dz \right\}$

Because they represent the distribution function's moments, macroscopic variables such as density ρ and velocity *u* are linked to it. They can be obtained by integrating and weighing it over the velocity space. The macroscopic mass density is calculated as follows:

$$\rho(x,t) = \int f(x,\xi,t).d^3\xi \qquad (2.1)$$

The integration over the three-dimensional velocity space translates the contribution of all possible velocities to the density of particles at a position x and a time t.

Accordingly, by taking into account all possible velocities, the contribution of ξf can be used to calculate the macroscopic momentum density as the following:

$$\vec{p} = \rho(x,t).u(x,t) = \int \xi . f(x,\xi,t).d^3\xi \qquad (2.2)$$

Also, we can find the macroscopic total energy:

$$\rho(x,t).E(x,t) = \frac{1}{2}\int |\xi|^2 . f(x,\xi,t).d^3\xi \qquad (2.3)$$

This total energy contains two types of energy; the so-called kinetic energy due to the bulk motion of the fluid and the internal energy due to the random thermal motion of gas particles. So, the macroscopic internal energy density can be found as:

$$\rho(x,t).e(x,t) = \frac{1}{2}\int |v|^2 . f(x,\xi,t).d^3\xi \qquad (2.4)$$

Where v is the relative velocity, defined as: $v(x,t) = \xi(x,t) - u(x,t)$.

2.2. Boltzmann equation

The Boltzmann equation defines the fluid's statistical behavior. It describes the time evolution of the distribution function. As previously mentioned, *f* is a function of the following variables: position ***x***, particle velocity ξ, and time *t*, so we can express its total derivative in respect to time as the following:

$$\frac{df}{dt} = \left(\frac{\partial f}{\partial t}\right)\frac{dt}{dt} + \left(\frac{\partial f}{\partial x_\beta}\right)\frac{dx_\beta}{dt} + \left(\frac{\partial f}{\partial \xi_\beta}\right)\frac{d\xi_\beta}{dt} \qquad (2.5)$$

Some modifications can be made on the terms on the right-hand side; we have dt/dt =1, the particle velocity $\frac{dx_\beta}{dt} = \xi_\beta$, and according to Newton's second law $\frac{d\xi_\beta}{dt} = \frac{F_\beta}{\rho}$, which has a unit of: $\left[\frac{F_\beta}{\rho}\right] = N/Kg$. In addition, the total differential term can be expressed as: $\Omega(f) = \frac{df}{dt}$. We finally obtain the ***Boltzmann equation:***

$$\frac{\partial f}{\partial t} + \xi_\beta . \frac{\partial f}{\partial x_\beta} + \frac{F_\beta}{\rho} . \frac{\partial f}{\partial \xi_\beta} = \Omega(f) \qquad (2.6)$$

This can be seen as a kind of advection equation: the first two terms represent the distribution function being advected with the velocity ξ of its particles. The third term represents forces affecting this velocity. On the right hand side, we have a source term, which represents the local redistribution of *f* due to collisions. Therefore, the source term $\Omega(f)$ is called the *collision operator*.

2.2.1. The collision operator

The collision operator denotes the rate of change of the distribution function as a result of a collision. If no collisions occur, the Boltzmann equation is reduced to a pure convection equation (no diffusion). For any terrestrial fluid: $\Omega \neq 0$.

Many physical models describe collisions. We will only consider those who adhere to the following two criteria:

1. At equilibrium, Ω is such that the distribution is the Maxwell-Boltzmann distribution.
2. There is mass, momentum, and energy conservation during a collision:

$\int \Psi_k . \Omega . dx . d\xi$ for $k \in [0,1,2,3,4]$

Which can be extended into the following conservation properties:

Mass conservation: $\int \Omega(f) . dx . d\xi = 0$ (2.7)

Momentum conservation: $\int \xi . \Omega(f) . dx . d\xi = 0$ (2.8)

Total energy conservation: $\int \left|\xi^2\right| . \Omega(f) . dx . d\xi = 0$ (2.9)

Internal energy conservation: $\int \left|v^2\right| . \Omega(f) . dx . d\xi = 0$ (2.10)

The first collision operator expression introduced by Boltzmann accurately describes reality, but since it is integral, it gives the Boltzmann equation an integral-differential character that is difficult to solve both analytically and numerically. The BGK approximation, which respects the two aforementioned properties, is the best and simplest

approximation, to the extent that a very large majority of LBM codes incorporate this collision operator.

2.2.2. The BGK operator

This operator, named after its inventors Bhatnagar, Gross and Krook, introduces a very simple and practical approximation. It entails stating that each collision modifies the distribution function f by a factor proportional to the difference between the current distribution f and the equilibrium distribution f^{eq}; $\Omega_{BGK} \equiv C.\left[f - f^{eq} \right]$. [21]

The dimensional analysis of the Boltzmann equation shows that Ω has the dimension of f divided by time. As a result, C is homogeneous to the inverse of the relaxation time τ. The fact that the system is stable provides us with more detail.

Consider a positive perturbation applied to an equilibrium system $(f > f^{eq})$. Since the system is stable, f must decrease in time in order to return to its equilibrium value f^{eq}. We, therefore, have $\frac{\partial f}{\partial t} < 0$. As a result, the constant C is negative. We then write: $\Omega_{BGK} \equiv \frac{-1}{\tau}.\left[f - f^{eq} \right]$

We thus obtain: $\frac{\partial f}{\partial t} + \xi.\nabla f = \frac{-1}{\tau}.\left[f - f^{eq} \right]$ (2.11)

This equation is highly nonlinear since the equilibrium distribution is affected by many variables, including density ρ, velocity ξ, and temperature T (as indicated in the expression for the Maxwell-Boltzmann distribution). Nonetheless, this non-linearity is tolerable since the value of f^{eq} in x is determined solely by quantities evaluated in x. Meaning that it is local in the physical space.

Moreover, this argument specifically reflects one of the major benefits of the LBM, which is summarized in the following sentence: "the non-linearity is local, non-locality is linear." interactions between nodes are entirely linear, while the method's non-linearity enters in a local collision process within each node. This property makes the LBM very amenable to high-performance computing on parallel architectures, including GPUs. Coupled with

the method's simplicity, this means that parallelized LB simulations can be tailor-made for a particular case more quickly than simulations using a conventional method [18]

2.3. From a microscopic to a macroscopic scale

A question can now be raised: how can the Boltzmann equation explain the macroscopic behavior of a fluid using a molecular description and the distribution function?

The response is that by manipulating the Boltzmann equation, the macroscopic equations governing the fluid's behavior can be obtained directly. To be more specific, the mass conservation equation is obtained directly by integrating the Boltzmann equation over velocity space. The other macroscopic conservation equations are obtained by the moments of the equation i.e. multiplying the equation with functions of *x* and integrating over velocity space.

To deal with the force terms, we need to know the moments of the force term, which we can find directly using multidimensional integration by parts as:

$$\int \frac{\partial f}{\partial \xi_\beta} . d^3\xi = 0 \qquad (2.12)$$

$$\int \xi_\alpha . \frac{\partial f}{\partial \xi_\beta} d^3\xi = -\int \frac{\partial \xi_\alpha}{\partial \xi_\beta} . f . d^3\xi = -\rho \delta_{\alpha\beta} \qquad (2.13)$$

$$\int \xi_\alpha . \xi_\alpha . \frac{\partial f}{\partial \xi_\beta} d^3\xi = -\int \frac{\partial (\xi_\alpha . \xi_\alpha)}{\partial \xi_\beta} . f . d^3\xi = -2\rho u_\beta \qquad (2.14)$$

2.3.1. Mass conservation equation

As previously said, by integrating the Boltzmann equation over the velocity space we can find the continuity equation as follows:

$$\frac{\partial}{\partial t} \int f . d^3\xi + \frac{\partial}{\partial x_\beta} \int \xi_\beta . f . d^3\xi + \frac{F_\beta}{\rho} \int \frac{\partial f}{\partial \xi_\beta} . d^3\xi = \int \Omega(f) . d^3\xi \qquad (2.15)$$

The first two terms represent the moments calculated in equation (2.1) and (2.2); the force term is equal to 0 as shown in equation (2.12). In addition, the integral of the right-hand side is solved in the collision operator's properties in (2.7). We thus obtain the following equation, known as the ***mass conservation equation:***

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_\alpha)}{\partial x_\beta} = 0 \qquad (2.16)$$

In the above equation, it is clear that it is independent of the form of the distribution function, a point that will prove to be useful for future material.

2.3.2. Momentum conservation equation

By similarly taking the first moment of the Boltzmann equation, we find:

$$\frac{\partial (\rho u_\alpha)}{\partial t} + \frac{\partial \Pi_{\alpha\beta}}{\partial x_\beta} = F_\alpha \qquad (2.17)$$

Where

- $\Pi_{\alpha\beta}$ is the momentum flux tensor defined in the Euler equation as: $\Pi_{\alpha\beta} = \rho u_\alpha u_\beta - \sigma_{\alpha\beta}$
- $\sigma_{\alpha\beta}$ is the stress tensor.

Since the Euler equation describes only reversible momentum transfer, the stress tensor represents pressure forces, which are reversible. So for ideal fluids, we write: $\sigma_{\alpha\beta} = -p\delta_{\alpha\beta}$ [22].

Therefore, we can finally obtain the following equation:

$$\frac{\partial (\rho u_\alpha)}{\partial t} + \frac{\partial (\rho u_\alpha u_\beta)}{\partial x_\beta} = -\frac{\partial p}{\partial x_\alpha} + F_\alpha \qquad (2.18)$$

2.3.3. Energy conservation equation

When multiplying the Boltzmann equation by $\xi_\alpha . \xi_\alpha$ before integrating over the velocity space we obtain (here we directly split the moment similarly to the momentum equation):

$$\frac{\partial(\rho E)}{\partial t}+\frac{\partial(\rho u_\beta E)}{\partial x_\beta}=\frac{\partial(u_\alpha \sigma_{\alpha\beta})}{\partial x_\beta}+F_\beta u_\beta-\frac{\partial q}{\partial x_\beta} \qquad (2.19)$$

Where q is the heat flux given as: $q_\beta=\frac{1}{2}\int v_\alpha v_\alpha v_\beta . f . d^3\xi$

We can eliminate the bulk motion energy component $\frac{1}{2}\rho\left|u^2\right|$ by subtracting the mass conservation equation multiplied with u_α. We therefore obtain:

$$\frac{\partial(\rho . e)}{\partial t}+\frac{\partial(\rho u_\beta . e)}{\partial x_\beta}=\sigma_{\alpha\beta}\frac{\partial u_\alpha}{\partial x_\beta}-\frac{\partial q}{\partial x_\beta} \qquad (2.20)$$

It is known that the Euler equations are a simplification of the more general Navier-Stokes equations of fluid dynamics. They neglect the effects of viscosity by considering the fluid as "ideal". When both viscosity and the heat flux are added, The Navier-Stokes-Fourier system is obtained.

Assuming that $f \approx f^{eq}$, the equations previously demonstrated result in the Eulerian momentum and energy conservation equations. In that perspective and similarly to the macroscopic scope, it becomes clear that the phenomena of viscosity and heat diffusivity are related to the non-equilibrium of the distribution function.

The Chapman-Enskog analysis is an established method of connecting the kinetic and continuum pictures by finding the non-equilibrium contribution to *f*. Its main idea is expressing *f* as a perturbation expansion about f^{eq} [6]:

$$f = f^{eq}+\varepsilon f^{(1)}+\varepsilon^2 f^{(2)}+... \qquad (2.21)$$

Where ε is the smallness parameter, it labels each term's order in the Knudsen number.

In the absence of an external force field, the Maxwell-Boltzmann equilibrium distribution becomes the Maxwell distribution. It is the exponential of the ratio of kinetic and thermal energy. We then write:

$$f^{eq}=\frac{\rho}{(2\pi \tilde{R}T)^{D/2}}.e^{\left(\frac{-(v-u)^2}{RT}\right)} \qquad (2.22)$$

3. The Lattice Boltzmann implementation

Unlike conventional numerical methods based on the discretization of the equations of fluid mechanics, which can be hard to implement and parallelize, the LBM is based on microscopic models and mesoscopic kinetic equations.

This method follows the same approach taken by Ludwig Boltzmann concerning thermodynamic laws. The basic premise states that the collective behavior of a set of microscopic particles is at the origin of the observed macroscopic fluid flow and that its dynamics are not sensitive to the underlying details of microscopic physics [23].
Simple kinetic models are then constructed, which include the physics of microscopic or mesoscopic processes so that the averaged properties behave according to the equations of fluid dynamics on the macroscale. Despite the fact that the LBM is known as a particle-based method, it does not monitor the behavior of each concrete particle but rather their distribution. This is consistent with its primary focus: averaged microscopic behavior.

Analytically, the Boltzmann equation is much more difficult to solve than the NSE. However, it is paradoxically simpler to implement. The first explanation is that most LBM codes use a simplified version of kinetic theory in which mass and momentum are conserved during collisions. Moreover, considering that molecules have one inner degree of liberty, collisions are elastic and do not transport rotational or vibrational energy [24].
The second reason is that the force-free Boltzmann equation is a simple hyperbolic equation that describes the advection of the distribution function *f*.
This approach prevents us from discretizing the term $(u.\nabla u)$ present in conventional methods that is a major difficulty since it introduces complicated iterative numerical schemes along with approximation errors.

3.1. The discretized Lattice Boltzmann Equation

To numerically implement the Boltzmann equation the continuous variables t, x and c need to be discretized. The first parameter introduced here is the discrete-velocity distribution function f_i along the ith direction. Similar to the definition previously written, it represents the density of particles with velocity c_i at a time t and a position x. However, unlike the continuous distribution, the variables here are discrete. For example, the velocity c_i belongs to a small set of discrete velocities $\{c_i\}$. The points x where f_i is defined are arranged in space as a square lattice with Δx lattice spacing. Furthermore, f_i is only defined at specific times t, separated by a time step Δt.

One is free to choose the values of Δx, Δy, Δz, Δt, and c_i independently. However, it will be wise to correlate them. Indeed, if we can ensure that a particle moves by jumping from one mesh node to another mesh node during the time step t, we avoid the case where a particle is between two nodes, which would require interpolations. Therefore, with the space steps Δx, Δy and Δz chosen, each velocity c_i will be made to respect the following synchronization condition [25]:

$$\left\{ \begin{array}{l} c_{ix} \text{ is an integer multiple of } \dfrac{\Delta x}{\Delta t} \\ c_{iy} \text{ is an integer multiple of } \dfrac{\Delta y}{\Delta t} \\ c_{iz} \text{ is an integer multiple of } \dfrac{\Delta z}{\Delta t} \end{array} \right\}$$

It has previously been shown that there is a link between macroscopic quantities, such as density, momentum and total energy, and the various moments of the distribution function $M^{(n)} = \int f.c^n.dc$

Hence, new expressions must be found to replace them when the velocities only take a few values in finite dimensional set. The Gauss method of quadratures is one method for

obtaining an approximate value for a continuous integral. It states that there is a set of Q weighting coefficients w_i such that the following equality holds for all $n \in [0,1,2,3,4]$:

$$\int f.c^n.dc = \sum_{i=1}^{Q} w_i.f(x,c_i,t).c_i^n \qquad (2.23)$$

Which defines the distribution function discretized by the relation: $f_i(x,t) = w_i.f(x,c_i,t)$

Discrete velocities c_i along with weighting coefficients w_i form the so-called **velocity sets** $\{c_i, w_i\}$

These different velocity sets are useful to classify different lattices, each serving a specific purpose.

They are usually denoted as *DnQm*, *n* being the number of spatial dimension and *m* the number of velocities in the set. D1Q3, D2Q9, D3Q15, D3Q19, and D3Q27 are the most widely used velocity sets for solving the Navier-Stokes equation.

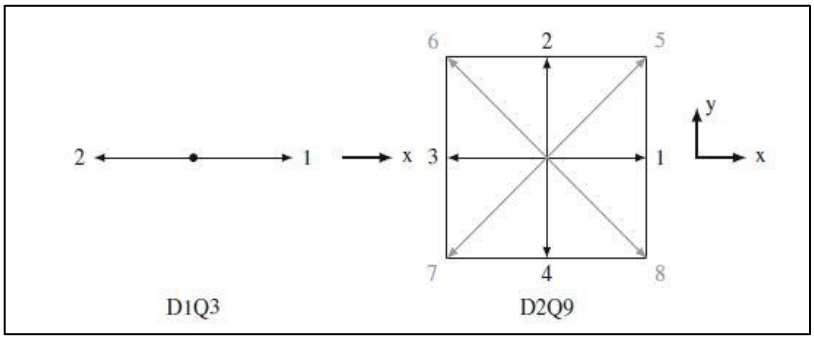

Figure 6: D1Q3 and D2Q9 lattice schemes [26].

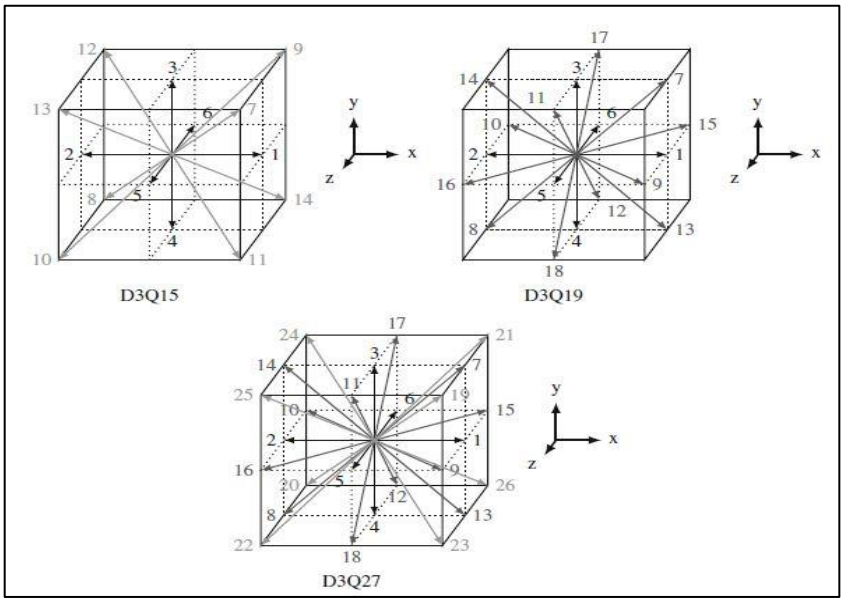

Figure 7: Three-dimensional lattice models: D3Q15, D3Q19, and D3Q27 [27].

To reduce memory and computational requirements, it is preferable to use as few velocities as possible. However, there is a tradeoff between higher precision and smaller velocity sets (e.g. D3Q15) (e.g. D3Q27). D3Q19 is the most widely used velocity set in 3D.

In the following section, the demonstration of how to obtain the lattice Boltzmann equation (LBE) by discretizing the Boltzmann equation in velocity space, physical space and time is explained:

$$\frac{f_i(x,y,z,t+\Delta t)-f_i(x,y,z,t)}{\Delta t}+c_{ix}.\frac{f_i(x+\Delta x,y,z,t+\Delta t)-f_i(x,y,z,t+\Delta t)}{\Delta x}$$
$$+c_{iy}.\frac{f_i(x,y+\Delta y,z,t+\Delta t)-f_i(x,y,z,t+\Delta t)}{\Delta y}$$
$$+c_{iz}.\frac{f_i(x,y,z+\Delta z,t+\Delta t)-f_i(x,y,z,t+\Delta t)}{\Delta z}$$
$$=-\frac{1}{\tau}\left[f_i(x,y,z,t)-f_i^{eq}(x,y,z,t)\right] \qquad (2.24)$$

Given that $c_{ix} = \frac{\Delta x}{\Delta t}$, $c_{iy} = \frac{\Delta y}{\Delta t}$, $c_{iz} = \frac{\Delta z}{\Delta t}$, we get:

$$\frac{f_i(x,y,z,t+\Delta t) - f_i(x,y,z,t)}{\Delta t} + \frac{f_i(x + c_{ix}.\Delta x, y, z, t+\Delta t) - f_i(x,y,z,t+\Delta t)}{\Delta t} + \frac{f_i(x, y + c_{iy}.\Delta y, z, t+\Delta t) - f_i(x,y,z,t+\Delta t)}{\Delta t} + \frac{f_i(x, y, z + c_{iz}.\Delta z, t+\Delta t) - f_i(x,y,z,t+\Delta t)}{\Delta t} = -\frac{1}{\tau}\left[f_i(x,y,z,t) - f_i^{eq}(x,y,z,t) \right] \quad (2.25)$$

Knowing the definition of a total differential:

$$df = \frac{\partial f}{\partial x}.dx + \frac{\partial f}{\partial y}.dy + \frac{\partial f}{\partial z}.dz \quad (2.26)$$

The last three terms on the left can be replaced by the following expression:

$$\frac{f_i(x + c_{ix}.\Delta x, y + c_{iy}.\Delta y, z + c_{iz}.\Delta z, t+\Delta t) - f_i(x,y,z,t+\Delta t)}{\Delta t} \quad (2.27)$$

We thus obtain:

$$\frac{\cancel{f_i(x,y,z,t+\Delta t)} - f_i(x,y,z,t)}{\Delta t} + \frac{f_i(x + c_{ix}.\Delta x, y + c_{iy}.\Delta y, z + c_{iz}.\Delta z, t+\Delta t) - \cancel{f_i(x,y,z,t+\Delta t)}}{\Delta t} = -\frac{1}{\tau}\left[f_i(x,y,z,t) - f_i^{eq}(x,y,z,t) \right] \quad (2.28)$$

We replace τ with a dimensionless relaxation time τ^*, such that: $\tau^* = \frac{\tau}{\Delta t}$. Finally, we obtain the lattice Boltzmann equation:

$$f_i(x + c_{ix}.\Delta x, y + c_{iy}.\Delta y, z + c_{iz}.\Delta z, t+\Delta t) - f_i(x,y,z,t) = -\frac{1}{\tau^*}\left[f_i(x,y,z,t) - f_i^{eq}(x,y,z,t) \right] \quad (2.29)$$

The value of the equilibrium distribution function is crucial since it determines the model's behavior. Then it must be discretized as well into the the specific set of velocities. For that sake, the equilibrium distribution is approximated as:

$$f^{(M)} = \sum_{i=1}^{Q} w_i . f_i^{eq} \;\; ; \;\; \sum_{i=1}^{Q} w_i = 1 \quad (2.30)$$

When performing a Taylor series expansion of $f^{(M)}$ up to second order and using the previous approximations we obtain [28]:

$$f_i^{eq} = \rho.w_i.\left(1+\frac{u.c_i}{c_s^2}+\frac{(u.c_i)^2}{2c_s^4}-\frac{u.u}{2c_s^2}\right) \qquad (2.31)$$

Where:

- w_i are the weights specific to the velocity set
- c_i the lattice velocity
- u the macroscopic velocity
- c_s the speed of sound (in the isothermal LBE model, it is defined by the relation: $p = c_s^2.\rho$ defined in all the velocity sets above as: $c_s^2 = \frac{1}{3}.\frac{\Delta x^2}{\Delta t^2}$. [29]

When the LBE is implemented, it is decomposed into two parts that are determined separately and successively [30]

- The collision process is first determined, in which particles in the same position redistribute their velocities due to interactions:

  $$f_i^*(x,t) = f_i(x,t) - \frac{1}{\tau^*}\left(f_i(x,t) - f_i^{eq}(x,t)\right) \qquad (2.32)$$

  Where: f_i^* is the distribution function after collision.

- The streaming (or propagation) is determined, in which particles change position due to their velocity:

  $$f_i(x + c.\Delta t, t + \Delta t) = f_i^*(x,t) \qquad (2.33)$$

The different steps in the LBE operations can be presented as the following: first during the collision step, the density ρ and the macroscopic velocity *u* are calculated to determine the equilibrium distribution function f_i^{eq} and the post-collision distribution f_i^*. We stream the resulting distribution to neighboring nodes after collision. One time step has elapsed since these two operations were completed, and the operations are then repeated. This process can be summarized in the following scheme:

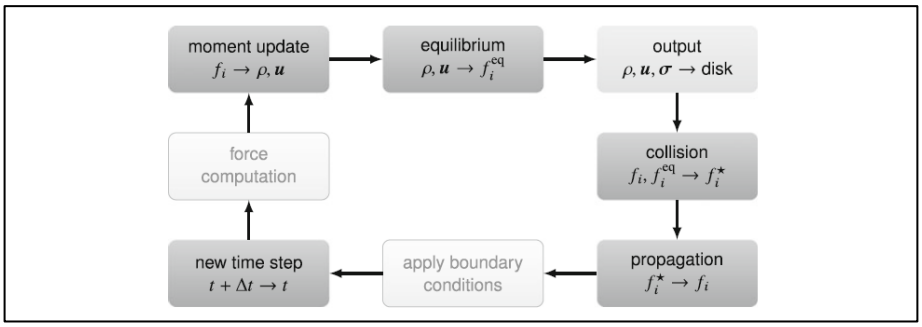

Figure 8: An overview of one cycle on the LB algorithm. The dark grey boxes show sub-steps that are necessary for the evolution of the solution. The light grey box indicates the optimal output step. The pale boxes show steps whose details are given later.

3.2. Boundary and initial conditions

The initialization of the Boltzmann lattice algorithm depends on the case to be simulated. Two cases can be distinguished:

1. If the goal is a steady-state solution, it is sufficient to bring initial populations to equilibrium. $f_i(x,t=0) = f_i^{eq}(\rho(x,t=0), u(x,t=0))$. This is a common choice for initial macroscopic fields : $\rho(x,t=0)=1$ and $u(x,t=0)=0$
2. If the goal is a time-dependent solution with non-homogeneous initial conditions, population initialization must take into account both equilibrium and non-equilibrium components: $f_i(x,t=0) = f_i^{eq}(x,t=0) + f_i^{neq}(x,t=0)$

The formulation of boundary conditions is required for the solution of any mechanical problem. The lattice Boltzmann method can be used to describe boundary conditions of various types by varying the values of the distribution function for the nodes

corresponding to the system boundary. This section is dedicated to the implementation of common boundary conditions in fluid mechanics, such as periodic conditions, no-slip conditions at solid boundaries, and imposed pressure conditions.

3.2.1. Periodic conditions

Periodic conditions are considered in the case of a closed loop flow, where the inlet and outlet are treated as lines of neighboring fluid nodes. Figure 9 depicts the implementation of this condition using the example of a node at the flow's outlet. It can be seen that the fluid particles carried by the distributions f_1, f_5, and f_8 will be advected to the three neighboring nodes of the opposite boundary (the inlet) according to the velocity directions c_1, c_5, and c_8 after one time step Δt.

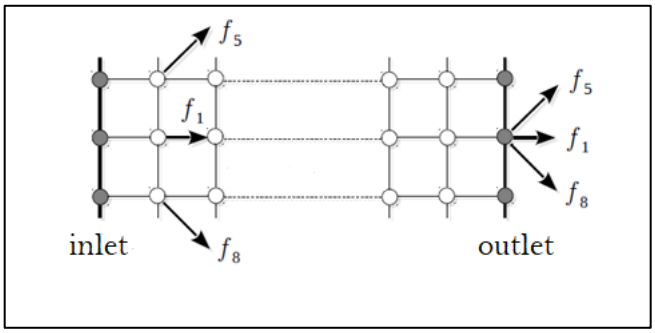

Figure 9: Implementation of periodic conditions

3.2.2. Bounce-back condition

The assumption of non-slip along a wall is a common application in fluid mechanics: the velocity of the flow in the direction tangent to the wall is zero. This assumption is usually made in conjunction with the assumption of the wall's impermeability (nullity of the velocity component in the direction normal to the wall). The LBM algorithm normally treats the joint implementation of these two conditions as a bounce-back of the fluid particles along the wall. Figure 10 depicts a schematic representation of the implementation proposed in [31].

Consider a boundary (grey line) half a length (h/2) away from the final line of fluid nodes. The fluid particles corresponding to the distribution function's (post-collision) values f_4,

f_7, and f_8 at time t_0 move away from the fluid node and bounce against the solid wall at time $t + \Delta t / 2$. They reverse their velocities (while remaining in the same direction) and return to the same fluid particle at time $t + \Delta t$, as distribution function values f_2, f_5, and f_6 (respectively).

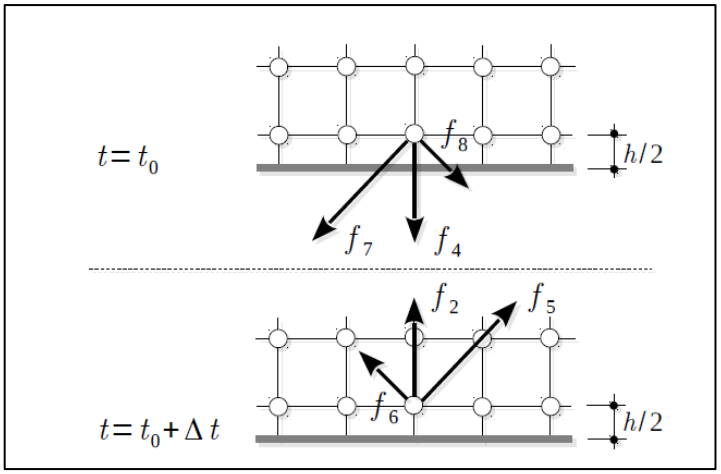

Figure 10: Implementation of bounce-back conditions

3.2.3. Pressure (velocity) condition imposed

During the LBM algorithm's propagation step, the distribution functions of a node are advected from neighboring nodes along the discretized velocity directions. Thus, after this step, the values of the distribution function for certain velocities (or directions) are known at the level of the nodes at the boundary, while others remain unknown due to a lack of information from outside the fluid domain. The unknown distribution function values must be determined using appropriate criteria. It is possible, for example, to calculate the unknown distribution function values in order to obtain the desired pressure or velocity. To begin, consider a node at the flow's inlet, as shown in Figure 11 that follows.

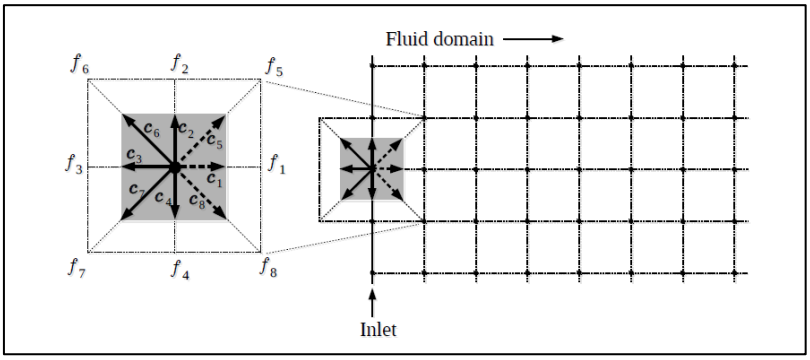

Figure 11: Implementation of velocity imposed conditions [32]

After a propagation step, the distributions f_0, f_2, f_3, f_4, f_6, and f_7 are known at this node because they are derived from neighboring nodes within the fluid domain, whereas the distributions f_1, f_5, and f_8 are unknown and can be adjusted to create a p_{in} pressure or velocity u_{in} (or mixed pressure/velocity conditions). The case of an imposed pressure (i.e., the Dirichlet condition) is studied first. Such a pressure will result in an initially unknown velocity at the flow's inlet. To complete the implementation, the two components of this velocity $\{u_1,u_2\}$ must also be calculated. We obtain the following relationships using the following equations:

$$\rho = m\sum_{i=0}^{8} f_i \ , \ u = \frac{m}{\rho}\sum_{i=0}^{8} f_i.c_i \qquad (2.34)$$

and assuming that the velocity component parallel to the input boundary is zero (i.e., u_2=0):

$$m.\sum_{i=0}^{8} f_i = \rho_{in} = \frac{p_{in}}{c_s^2} \qquad (2.35)$$

$$m.\sum_{i=0}^{8} f_i c_i.e_1 = \rho u_1 \qquad (2.36)$$

$$m.\sum_{i=0}^{8} f_i c_i.\ e_2 = 0 \qquad (2.37)$$

A fourth equation is required to solve such a system of equations with four unknowns, f_1, f_5, f_8, and u_1. The rebound condition, as proposed in [32] is applied to the non-equilibrium part of the distribution function along the normal direction to the boundary, i.e.:

$$f_1 - f_1^{eq} = f_3 - f_3^{eq} \qquad (2.38)$$

Finally, the result is:

$$u_1 = 1 - (f_0 + f_2 + f_4 + 2(f_3 + f_6 + f_7))\frac{c_s^2}{p_{in}} \qquad (2.39)$$

$$f_1 = f_3 + \frac{2}{3}\frac{p_{in}}{c_s^2} \qquad (2.40)$$

$$f_5 = f_7 - \frac{1}{2}(f_2 - f_4) + \frac{1}{6}\frac{p_{in}}{c_s^2}u_1 \qquad (2.41)$$

$$f_8 = f_6 + \frac{1}{2}(f_2 - f_4) + \frac{1}{6}\frac{p_{in}}{c_s^2}u_1 \qquad (2.42)$$

It is possible to impose the value of the normal component by making a simple change to the control variable and always assuming a zero velocity component parallel to the boundary. In this case, a different combination of (2.35) and (2.36) yields the following equations, allowing us to determine the unknown quantities: ρ_{in}, f_1, f_5, f_8.

$$\rho_{in} = \frac{1}{1-u_1}(f_0 + f_2 + f_4 + 2(f_3 + f_6 + f_7) \qquad (2.43)$$

$$f_1 = f_3 + \frac{2}{3}\rho_{in}.u_1 \qquad (2.44)$$

$$f_5 = f_7 - \frac{1}{2}(f_2 - f_4) + \frac{1}{6}\rho_{in}.u_1 \qquad (2.45)$$

$$f_8 = f_6 + \frac{1}{2}(f_2 - f_4) + \frac{1}{6}\rho_{in}.u_1 \qquad (2.46)$$

The complete process of a LB algorithm's cycle, including the boundary condition is shown in figure 12.

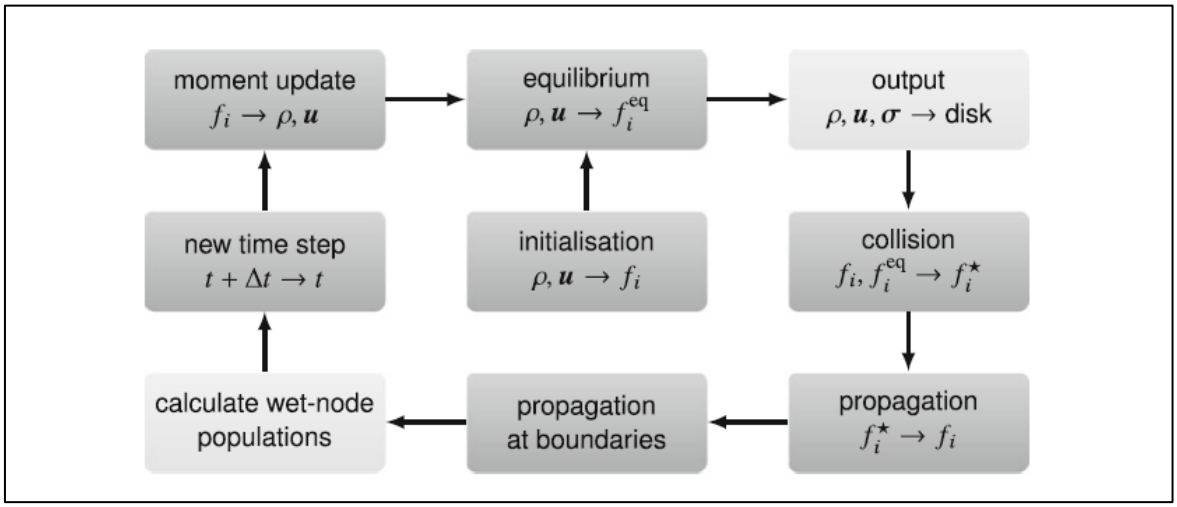

Figure 12: an overview of one cycle of the LB algorithm, considering boundary conditions and the one-off computation of initial conditions (center), but not considering forces. Optional sub-steps are shown in light grey boxes.

4. Conclusion

The understanding of the theoretical foundations of the LBM provides the essential elements necessary for the initiation to this method and its subsequent manipulation. While its implementation in a numerical environment proves to be particularly relevant as it is the discretization that will establish the close link between the described physical model and its implementation in software, which explains the high speed of execution, associated with LBM.

Chapter III: Framework structure

1. Introduction

Palabos is a solver that uses the C++ programming language in its interface; therefore, it is critical to first understand the motives for this choice by outlining the paradigms employed in this programming language as well as its primary features that benefit the solver.

Understanding these features will then allow us to discuss Palabos' techniques to appropriately laying out the main elements of a simulation as well as depicting the structure in which they are organized.

2. The programming language

It is the type of problem that determines which programming language to employ and which is the better option. This decision is based on the paradigm that each language is linked with. The programming paradigm is a fundamental style of computer programming that differs in how problem-solving processes are defined. Given the variety of paradigms and the interdependence of their concepts, we will limit ourselves to contrasting two paradigms that are frequently encountered in CFD works: procedural programming (found in the FORTRAN language, for example), and object-oriented programming (found in cases where a representation of real physical objects is required).

2.1. Procedural programming

This type of programming focuses on an algorithm that determines the logic behind a problem's solution. The problem is broken down into numerous tasks that must be completed in order to be solved, each of which is expressed as a function or subroutine, which is a set of instructions that, when run, provides the required output. Accessing and modifying data in the procedural paradigm is quite easy because there are only two types of variables; global and local. The global variables are declared at the beginning of the

program and are found throughout the resolution of the problem, while the local variables are those used to perform intermediate operations.

2.2. Object-oriented programming (OOP)

A programming paradigm focuses on thinking about problems in terms of objects and their behavior. The object is a data structure that closely mimics a physical object. Then there are the classes that are built around the objects to define their behavior and interactions with each other, and possibly to organize them in a hierarchy. The OOP puts its full force behind a concrete problem that may be destructured in a hierarchical manner and in which the data are at the heart of the problem (knowing that modifying them is more difficult than modifying them in procedural programming). Because of these factors, OOP is widely employed in the video game industry.

C++ is a general-purpose programming language that was created as an extension to the C language, making it multi-paradigm (including object-oriented paradigm, procedural, functional). It is a simple language in the sense that it allows programs to be broken down into logical units and pieces, and it comes with a large library and a wide range of data types. C++ is one of the fastest languages and is very near to low-level[1], providing for complete memory allocation and management control.

Inheritance and polymorphism are two of the many interesting features in C++ that will be very relevant to our work.

Inheritance is the process of creating a new class (derived class) that inherits the features of an existing class (base class). Inheritance enhances reusability while also reducing code length in OOP.

Polymorphism enables an object to select the form of a function to implement at both compile-time (overloading method) and run-time (overriding method).

[1] What is a low-level language? It's a programming language with little separation between the machine and the language itself. As a result, "close to the hardware" is a term used to describe low-level languages.

3. PALABOS (Parallel Lattice Boltzmann Solver)

3.1. Introduction

Palabos is a general-purpose computational fluid dynamics (CFD) framework with a kernel built on the lattice Boltzmann (LB) methodology. The native programming framework of the library is written in C++. The approach is based on a molecular representation of a fluid based on the Boltzmann equation, and it can explicitly integrate concepts derived from an understanding of molecule interactions. It keeps the cycle between the elaboration of a theory and the formulation of a corresponding numerical model short.

Palabos has proved its efficiency in the field of Computational Fluid Dynamics over the years [33]. Although LB solvers are explicitly time-dependent, Palabos' efficiency and accuracy are comparable to that of a commercial, explicitly stationary Navier-Stokes solver [34].

The close correlation between the physical model and the software implementation accounts for the high execution speed of LBM implementations. The transport in space of statistical groups of flow particles, for example, converts into copies inside a matrix-like data structure from a cell to one of its neighbors, as an artifact of LBM's heritage from Cellular automata. In a distributed-memory parallel computing platform, this gives direct control over the use of computer memory and its cache hierarchies, as well as data distribution and access.

More specifically, canonical LBM models are built on a distinction between collision, which includes all physics but does not include data communication between cells, and streaming, which is model-independent but communicates data between adjacent cells. As a result, the technical aspects of the algorithm (data access and parallel communication) are frequently incorporated into the streaming step. The Palabos kernel then takes care of the streaming while the user concentrates on the collision model's higher-level implementation.

3.2. Software architecture and application development model

Since it is designed around an efficient architecture and access to the key data chunks identified in a simulation, the Palabos software structure can be considered data-oriented. Two types of algorithms act on the data sets:

- Pure LB models, which respect the collision /streaming cycles: they are defined at a cell-base level (since the collision step encompasses the physics and nature of each collision that can change from a cell to another). To deal with the large number of collision terms used in the literature, we use an inheritance-based type of polymorphism. Object composition, another object-oriented idiom, is used to combine various elements of a collision model.
- Non-LB algorithms: they are defined at the component level (rectangular cell groups resulting from domain partitioning) and involve more explicit programming efforts. Even though, as previously said, the programmer is unaware of the implementation specifics of parallel communication and mesh homogeneity.

The software architecture is designed to be highly extensible as it allows the programmer to incorporate a variety of models [35]. The environment (which translates into lattice descriptors), the collision step with its physical models, and the data procession that can be implemented in the streaming step are all the steps of a simulation where these changes or additions can be made. As a result, we differentiate between the following mechanisms:

- Using C++'s "templating mechanisms," lattice descriptors can be written to define new data layouts.
- New local LB collision terms are described using collision models.
- Data processors are written to enforce coupling terms between different models or to define new non-local algorithms (type n°2 previously defined).

In the following section, the processes and technical aspects that were used to make the additions of a large number of models possible will be developed:

3.2.1. Data containers: Multiblock

Different types of multiblock describe the fundamental data containers of a Palabos simulation. It is possible to think of them as n-dimensional matrices. They hold a certain amount of data, such as each collision model associated with each LB cell, which is dynamically specified in terms of type and scale. There are four customizable types of multiblock. A new mesh is allocated to each instance (object) of a multiblock, regardless of its type. Meshes also have the same number of cells to ensure the parallel feature and allow interactions.

- *Multiblocklattice*: each part of this data structure corresponds to an LB cell. The variables of a cell, referred to as populations, constitute a discrete version of the statistical probability density function found in the kinetic theory. Along with other statistical details, an external force term may be introduced.

  The data layout is specified by a so-called lattice descriptor, which includes both individual cell data and global lattice information. The "ForcedD3Q19descriptor," for example, gives each cell 22 floating-point variables: 19 population, and a 3D force vector. In addition, each cell has a pointer to a "dynamics"[2] object that defines the collision model.

[2] *Values are assigned to cells in an ordinary table in a well-ordered and sequential manner. To get the value of a cell in memory, all you have to do is look up its index. Except that in a dynamic array, things are different. A dynamic array's size is variable. This has the benefit of adjusting the memory allocation to the program's requirements. However, without pointers, the array's values are dispersed in memory. Therefore, the values are ordered and findable thanks to these pointers. Knowing that each cell contains the value in question as well as the previous and following pointers, we can find the value of a specific cell by knowing the address of these.*

- *Multiscalarfield*: As a user-input, each element in this data structure contains a scalar variable (integer or floating-type point). Temperature in coupled, buoyancy-driven flows[3], or temporary storage of a scalar variable in a post-processing step are just a few examples.

- *Multitensorfield:* This data structure, like the multiscalarfield, contains a field of vectors or tensors with any number of components, such as velocity or force, during the simulation or post-processing stage. Depending on the memory layout's performance, it might be better to store directly in the multiblocklattice cell or in a separate multitensorfield.

- *Multiparticlefield:* This data container contains Langrangian particles (Lagrangian particles are used since the Lagrangian description emphasizes trajectories.) In the case of our simulation, this factor is just as fascinating).

  They can be ranked in ascending order based on their potential impact on the flow:
  - Fully passive: inert objects that have no impact on the flow and are only present in the environment.
  - One-way coupled: includes passive pointers to dynamic arrays of particles that are coupled to the flow and have a Lagrangian definition.
  - two-way coupled: contains particles that have a direct effect on the flow and are thus coupled in a more complex way.

It is important to keep in mind that:

- All multiblocks objects, including the multiparticlefield, allocate a new mesh to which particles are attached based on their space location.
- Due to the underlying similarity of mesh structures, particles are easily coupled with physical objects, which are described by multiblocks objects.

[3] *When heat is added to a fluid density that varies with the temperature, a flow can be induced due to the gravity acting on the density variations*

- Palabos provides a 2D or 3D structure that is separate, so all multiblocks are represented in 2D or 3D.

3.2.2. Collision model: Dynamics

These collision models are implemented as "dynamics" classes, which means they are attached ('interpreted') at runtime.

As previously mentioned, an LBM simulation follows a basic collision/stream cycle model. The first step, collision, contains all of the physics required for the model. Since collisions access all cells individually, this phase is entirely local in this regard. Furthermore, the computations performed in each cell are independent of the others (the data here is not communicated). On the other hand, the streaming step through its component-level definition is non-local because it allows the communication of data between adjacent cells by copying data from a cell to a local group of cells, defined by their velocities (which are defined in the cell's data layout contained in the lattice descriptor).

The collision model must not only explain the physics of the actual collision step, but also define the collision model's various properties through the implementation of macroscopic variable computation algorithms. Rescaling algorithms for populations and other statistical variables for a given set of cells with various time and/or space scales are also included.

The literacy is abundant with LB models and any code duplication must be avoided when implementing different models for different individual cells. For that matter, the dynamics classes (otherwise collision models) can be organized in a hierarchy. Since most of these models are based on the same type of equilibrium populations, the similarity in the computation of macroscopic variables for incompressible flows in LB models is

represented by an ancestor inheritance. Figure 13 shows how some models are organized hierarchically.

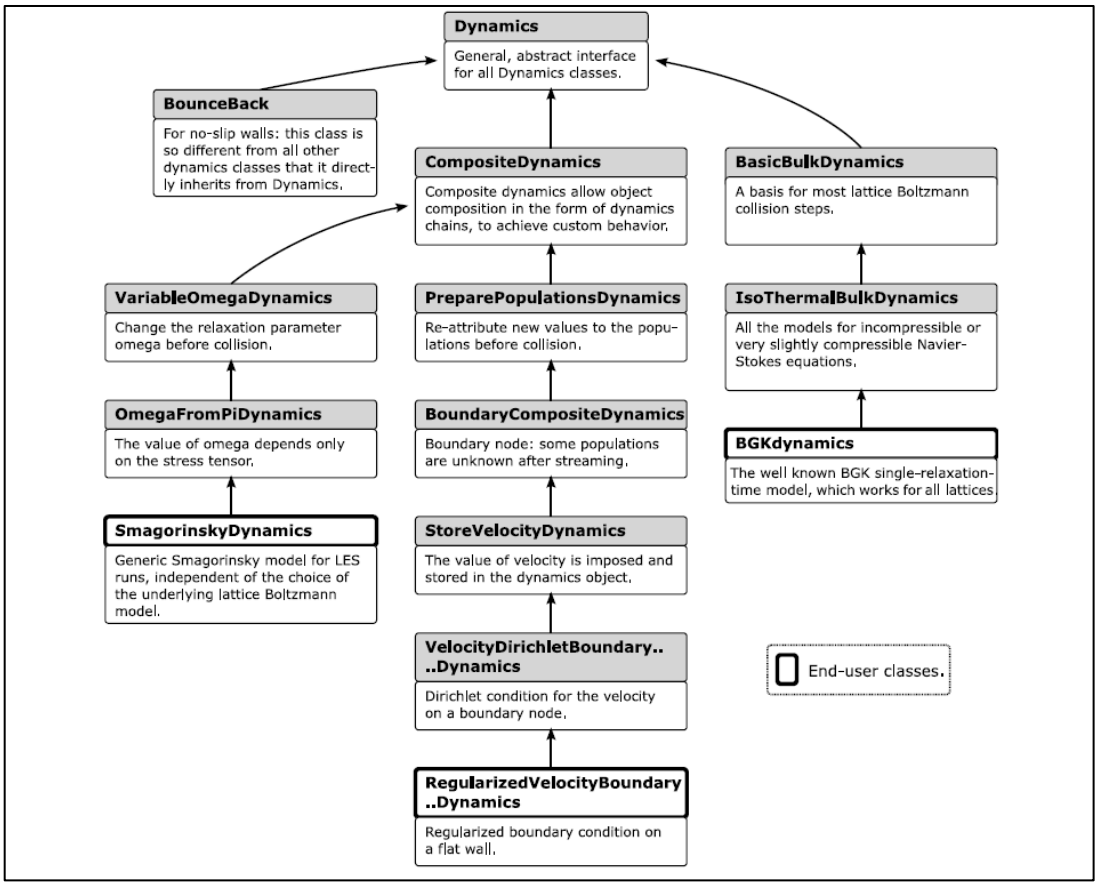

Figure 13: Inheritance graph of collision models.

Some proprieties cannot be inherited and must be constructed dynamically. For example, using an LES model can modify the collision term. However, adding these modifications eliminates the potential for interference or conflict. In each cell, the collision model is replaced by an LES model that obtains a copy of the original model to perform the collision phase. "Object composition," which is in Palabos equivalent to "Dynamics chains," enables this operation.

3.2.3. Non-local algorithms and couplings: Data processors

Data processors work on a broader level than collision models, acting on entire multiblock modules rather than single cells. In an LB simulation, they carry out operations that go beyond the collision/streaming paradigm. They can perform non-local operations that are not covered by the streaming step, or they can couple individual equations.
Some examples are as follows:

- Implementing a multi-phase or multi-component flow by coupling fluid components.
- Implementing a buoyancy-driven thermal flow by coupling a temperature and a fluid field.
- Implementation of accurate boundary conditions.
- Data post-processing, which involves extracting flow variables from the lattice and converting them to scalar/tensor fields before inserting them into data files.

Since data processors operate on a multi-block part basis, they have two distinct interesting features. The first is that they can share data across multiple multiblocks (of the same or different types) that they are working on simultaneously, which is important for practical different coupling implementations. The second allows for a reduction in the number of operations in the aforementioned subdomains; computation of average kinetic energy, computation of drag force acting on an obstacle (in our case, a wing section).

3.3. Memory organization and mesh refinement

During the streaming step, a cell must be able to communicate with its neighbors at any time. Depending on the complexity of the shapes on which the calculation is run, there are two key approaches to guaranteeing this access in an LB simulation. For exclusively normal shapes, since the data is organized in matrix-like data structures, simple index arithmetic allows access to the neighbors. In the case of irregular shapes, a neighbor-list approach is used, with pointers being used to reach all necessary neighbors. As a result,

more memory is needed. In the most complicated shapes, only six neighbors can be accessed directly via their pointers, while others can only be accessed indirectly. This is due to the high memory consumption and the fact that the number of neighbor pointers equals the number of populations (variables in a cell).

Palabos uses a more general matrix-based formalism than the LB code, in which the technique is dependent on the type of shapes. Of course, depending on the mesh, this formalism is revised, but the concept remains the same.

A powerful spatial hashing scheme is used for homogeneous simulations. A hash table is a data structure that stores data in an associative manner using a hash function. The data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data. During lookup, the key is hashed and the resulting hash indicated where the value is stored. Obviously, the low memory allocation is a benefit of such a system.

As for mesh-refined simulations, since Palabos decomposes multiblocks into components, it builds an octree to define[4] the relationships between those. The challenge then is to find the multiblock component's size that keeps a good compromise between domain shape and the parallel efficiency, because while the former gets optimal with small components, the latter does with larger components.

Palabos' approach to mesh refinement is vital to the memory organization among other factors. Palabos separates each multiblock with a different mesh level from the others and then implements them. Consequently, each level has a memory allocated accordingly. The fluid models shared by uniform and refined meshes are the same and remain usable at all levels of the simulation, but they must be rescaled using the Dynamics objects that handle communication between levels.

[4] An octree is a data structure in which each internal node is recursively subdivided into 8 octants

3.4. Parallelism

Palabos offers the convenience of parallel computing by breaking down the multiblock components into smaller subdivisions that can be computed simultaneously by multiple processors communicating via shared memory. Each component is therefore attributed through an MPI (Message Passing Interface) process. The strategy of domain attribution depends on the treated problem and relies on geometric criteria, on a principal of minimal inter-process communication, or is in some cases random to minimize the impact of highly imbalanced problems, such as multi-phase flows with rapidly moving inter-phase interfaces.

To manage MPI communication, Palabos encircles the components by a ghost layer that consists of envelope cells containing read-only only information (writing has no impact on other components) that are accessible from bulk cells. After each collision/streaming cycle or data processor modification of a multiblock, the information in ghost layers is updated. Evidently, these ghost layers augment the memory allocation, therefore affecting the computational performances and cost.

4. Conclusion:

The scope and concepts of the open-source Lattice Boltzmann library Palabos have been described. This, in turn, allows users to claim that the data structure of this software library is intended to combine great flexibility, spanning a wide variety of problems of interest to the LB community and the CFD framework in general, with high computational performances.

The understanding of the fundamental principles, schemes, and strategies utilized by Palabos in its memory allocation and data layout provides a better understanding of how, despite its newness, Palabos has proven its efficiency in the field of Computational Fluid Dynamics.

Chapter IV: Evaluation of the Parallel Lattice Boltzmann Solver

1. Introduction

Since the objective of this thesis is to evaluate the Lattice Boltzmann method on CFD benchmark cases using the PALABOS (PArallel LAttice BOltzmann Solver), this chapter will be divided into two sections. To begin, all of the steps required to install PALABOS on the computer used to execute the simulations will be described in details.

The test and evaluation method will be discussed in the second section by providing the two cases studied, namely the flow of a Poiseuille and that of a circular cylinder in 2D in both a steady and unsteady state. The numerical data generated from the simulation, their comparison with reference data, and their commentary are then developed.

2. First steps in Palabos

2.1. The configuration of the operating system

The University of Geneva website provides detailed documentation on how to start using Palabos. This solver can be used in several operating systems such as Windows, Linux or Linux-similar environments. All other prerequisites or recommended (but optional) packages are also listed. For more information, please refer to the "Get started" > "Palabos documentation" section.

After some technical problems encountered when using "Visual Studio 2019" under windows, the choice was made to use a Linux system, more precisely Ubuntu in its latest version 20.04.

2.1.1. Ubuntu operating system

Canonical created Ubuntu in October 2004 as open-source software. It is a highly robust operating system. This option was primarily motivated by the numerous advantages that Ubuntu provides over Windows. To name a few, it is an open-source operating system. Ubuntu offers a more user-friendly interface. It features a centralized software repository

from which we may download all necessary software. And, most importantly, since the Unix environment is the finest for programmers.

You have various options for using the Ubuntu operating system:

1. Use Ubuntu as the principal operating system or in conjunction with a Windows system. Because this option necessitates the creation of a new partitioning on the hard disks, and given the computer's limited storage space, it has been discarded.

2. Use a virtual machine. A virtual machine is the same as any other physical computer, as with a laptop. It has a CPU, memory, and drives for storing data. While the hardware components of a computer are solid and real, virtual machines (VMs) are generally thought of as virtual computers or software-defined computers within physical servers that exist only as code. However, given the cost that its use can inflict on RAM and the importance of memory resource integrity, this option is not the best choice for memory-intensive simulations.

3. As soon as the pc boots, use a USB to boot an identical image of the operating system from the BIOS. This option was chosen due to its balance of deployment convenience and physical resources requested for the simulations.

Creating a bootable Ubuntu USB stick from Microsoft Windows is a sensitive but feasible task, which we will go through in the next section. The ability to boot an operating system from a USB stick is referred to as live USB. Live USBs can be used to operate a computer, restore specific data, or simply test an operating system without having to install it on the computer's hard drive. The fact that system modifications, software, backups, and files are not permanently written to the USB is a significant downside of this technique. They will be deleted the next time you restart your computer.

However, there is a method around this limitation by editing the system file and utilizing software to establish a USB boot with persistent live drives. In 19.10 and later versions,

the size of a persistent partition is simply limited by the size of the drive (USB pen drive, SSD, HDD, memory card). To achieve this, the following steps were conducted:

2.1.2. Editing the ISO file

An ISO file (also known as an ISO image) is an archive file containing an identical copy (or image) of data found on an optical disc, such as a CD or DVD used for installation. First, download the ISO file for the most recent version of Ubuntu (currently Ubuntu 20.04 LTS). Edit the ISO file with HxD (a binary editor) to replace the two cosmetic boot options 'quiet splash' with 'persistent' (replace 12 characters with 12 characters). Figure 14 shows. The window displayed once the ISO file is opened and converted to a hexadecimal code

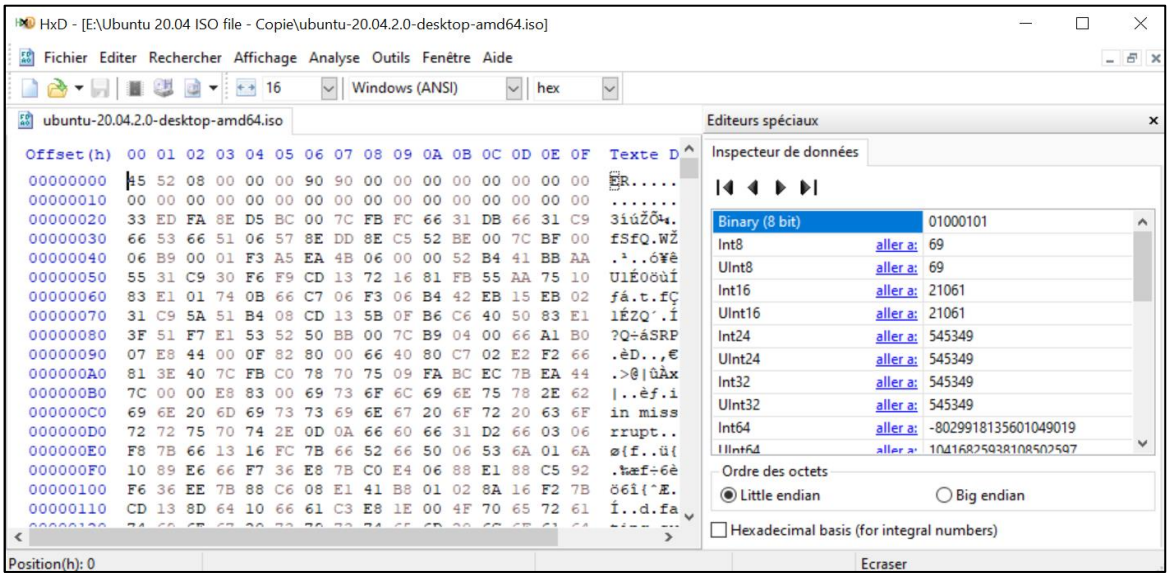

Figure 14: The window displayed once the ISO file is opened and converted to a hexadecimal code.

- Select 'Search' from the pull-down menu, followed by 'Replace...'
- Insure to substitute 'silent splash' with 'persistent ‘. It is critical that we replace 12 characters with the same number of characters. Otherwise, the changed file cannot be utilized to build a live system.
- Allow the binary editor to work by pressing the 'Replace all' button.

Figure 15 shows the “Replace” operation.

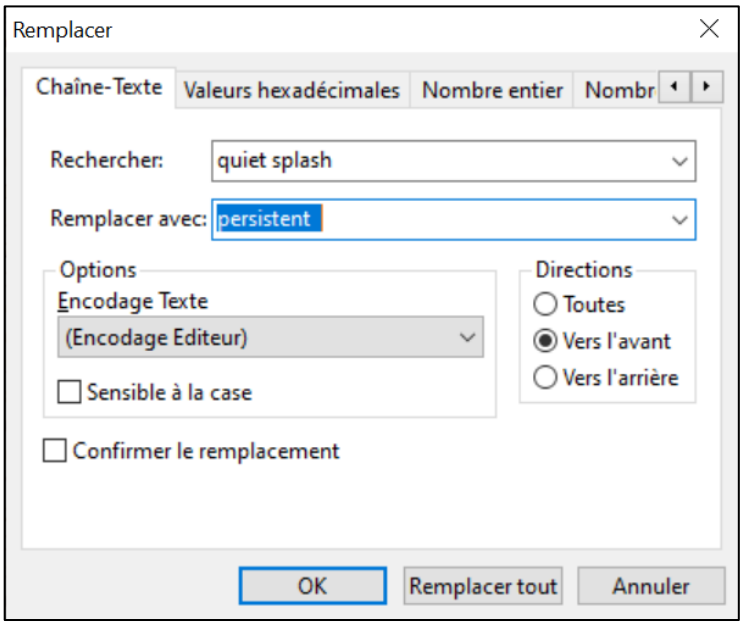

Figure 15: "Replace" window.

There should be 12 instances of 'silent splash' replaced with 'persistent' in the current 20.04 version. When the user presses the OK button, the dialog box shown in figure 16 displays. Figure 16 shows the dialog box indicating that 12 occurrences of 'silent splash' were replaced.

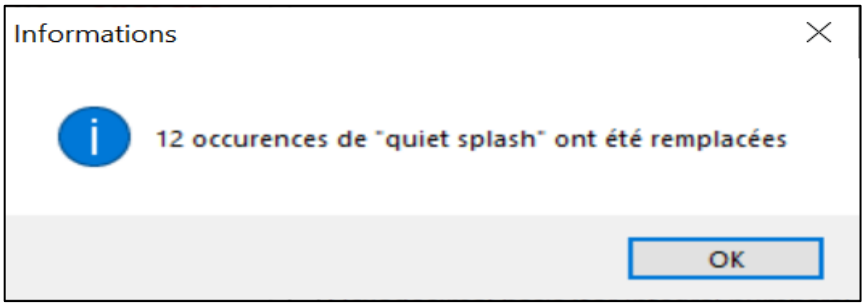

Figure 16: Dialog box indicating that 12 occurrences of 'silent splash' were replaced.

When the cloned drive is booted for the first time, the Ubuntu system will automatically create a casper-rw partition with an ext4 file system.

2.1.3. Configure the bootable USB

To begin, the designated USB to open Ubuntu is inserted. The Rufus 3.15 application[5] is then run, and the device is detected instantaneously. Then we select the ISO image that we wish to install on it. It is critical to choose the maximum available capacity for the size of persistent partitions. The operation can begin once the partition schemes and formatting options have been configured. All options are shown in figure 17.

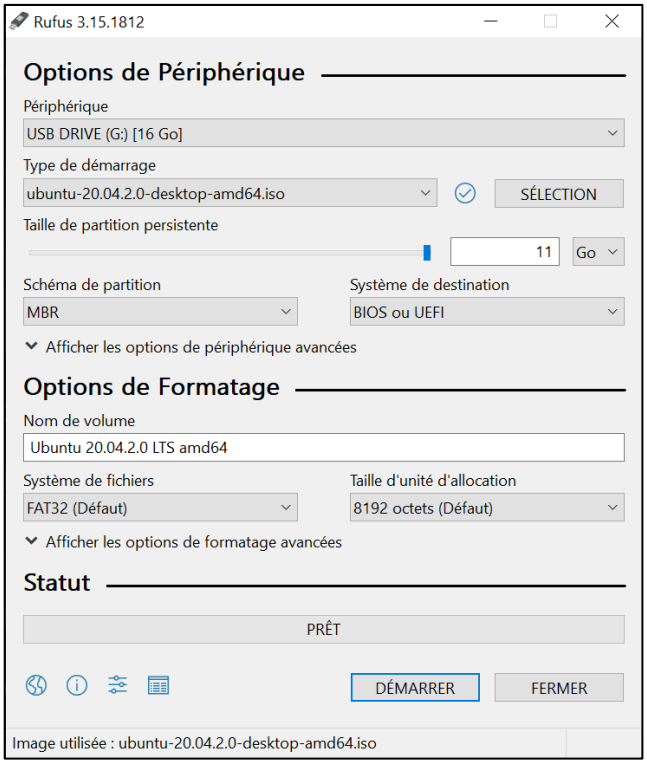

Figure 17: Rufus display of USB configuration.

[5] Rufus is a tool for formatting and creating bootable USB flash devices.

Wait for the "Ready" message to display, as shown in figure 18.

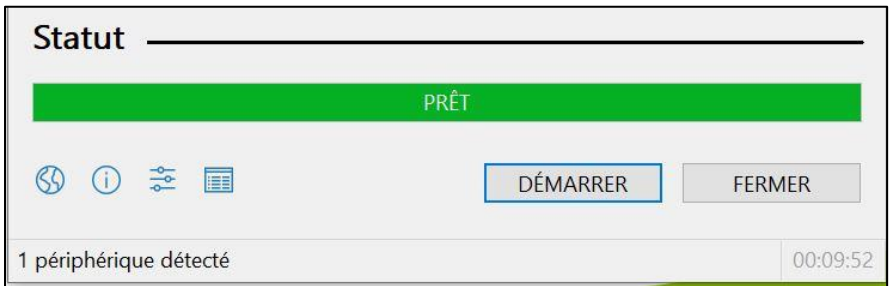

Figure 18: The dialog box where the "Ready" message is displayed.

The USB is now ready to be used just to run Ubuntu 20.04 and, therefore, is no longer capable of storing data as a standard USB. Once a bootable USB is created, it may be taken anywhere and used to run the operating system without installing it and access the persistent stored files.

2.1.4. Run Ubuntu Live

It is imperative to ensure that the computer's BIOS is set to boot from USB devices before inserting the USB flash drive and powering on the laptop. To boot from a USB drive, the boot sequence in the system's BIOS is modified to configure the boot priority and make

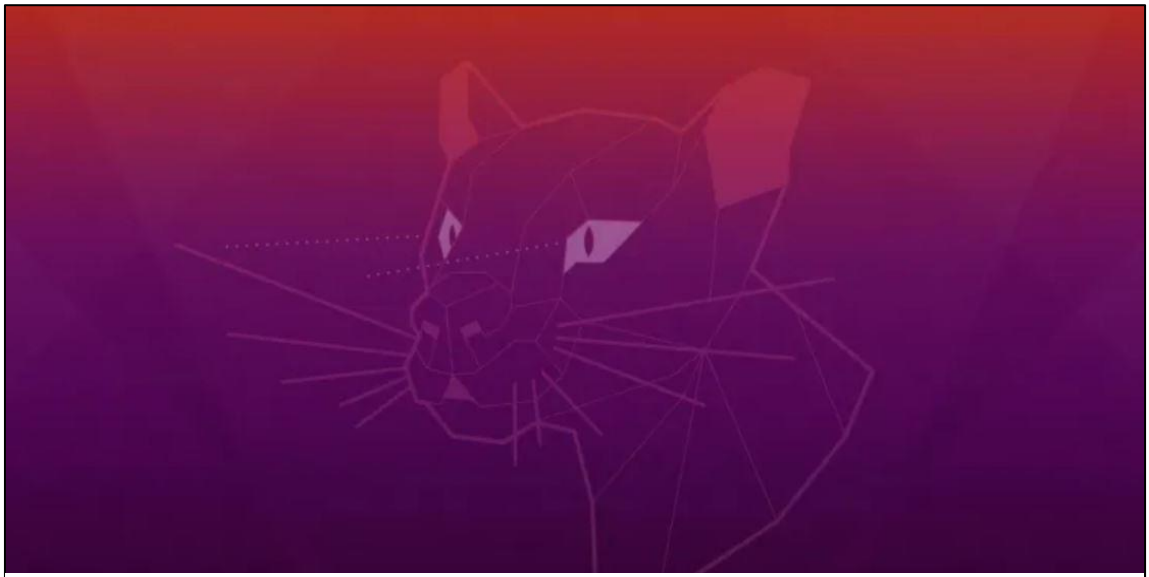

Figure 19: Ubuntu live desktop

the USB drive the top priority. Once the operating system is run, the desktop shown in figure 19 is displayed on the screen.

To make the Ubuntu framework more reliable and accessible, some packages need to be uploaded and installed from the Terminal as shown in figure 20. The following commands are executed:

§ sudo apt update && sudo apt upgrade -y

§sudo add-apt-repository universe

§sudo add-apt-repository multiverse

```
$ sudo add-apt-repository "deb http://download.webmin.com/download/repository sarge contrib"
$ sudo add-apt-repository "deb http://repo.ajenti.org/ng/debian main main ubuntu"

+-------------------------------------------------------------------------------+
| Following repositories might be helpful during packages testing:
+-------------------------------------------------------------------------------+
$ sudo add-apt-repository "deb http://ddebs.ubuntu.com precise main restricted universe multiverse"
$ sudo add-apt-repository "deb http://ddebs.ubuntu.com precise-updates main restricted universe mul
tiverse"
$ sudo add-apt-repository "deb http://ddebs.ubuntu.com precise-security main restricted universe mu
ltiverse"
$ sudo add-apt-repository "deb http://ddebs.ubuntu.com precise-proposed main restricted universe mu
ltiverse"

+-------------------------------------------------------------------------------+
| Missing GPG keys might be fetched by running:
+-------------------------------------------------------------------------------+
$ wget -q https://dl-ssl.google.com/linux/linux_signing_key.pub -O- | sudo apt-key add -
$ wget -q http://download.virtualbox.org/virtualbox/debian/oracle_vbox.asc -O- | sudo apt-key add -
$ wget -q http://www.webmin.com/jcameron-key.asc -O- | sudo apt-key add -
$ wget -q http://repo.ajenti.org/debian/key -O- | sudo apt-key add -
$ wget -q http://ddebs.ubuntu.com/dbgsym-release-key.asc -O- | sudo apt-key add -
```

Figure 20: commands executed in the terminal.

2.2. Obtain the Palabos library

The only step left to do before we begin running simulations is to download the open-source Palabos library from Gitlab. This library consists of a set of source codes that can be directly executed or modified to meet the demands of the programmer. It is divided into the folders detailed below and shown in figure 21:

1. Source codes for simulating configurations tailored for the application of LBM codes.

2. Coupled solvers, which are the contributions of researchers at the University of Geneva to the development of LBM by combining it with other numerical methods (such as FEM) to exploit the advantages of each in the numerical simulation of configurations that are too expensive numerically. More details can be found in [35].
3. External libraries, which are the work and contribution of developers and researchers from the scientific committee from all around the world who desire to the LBM advancement.

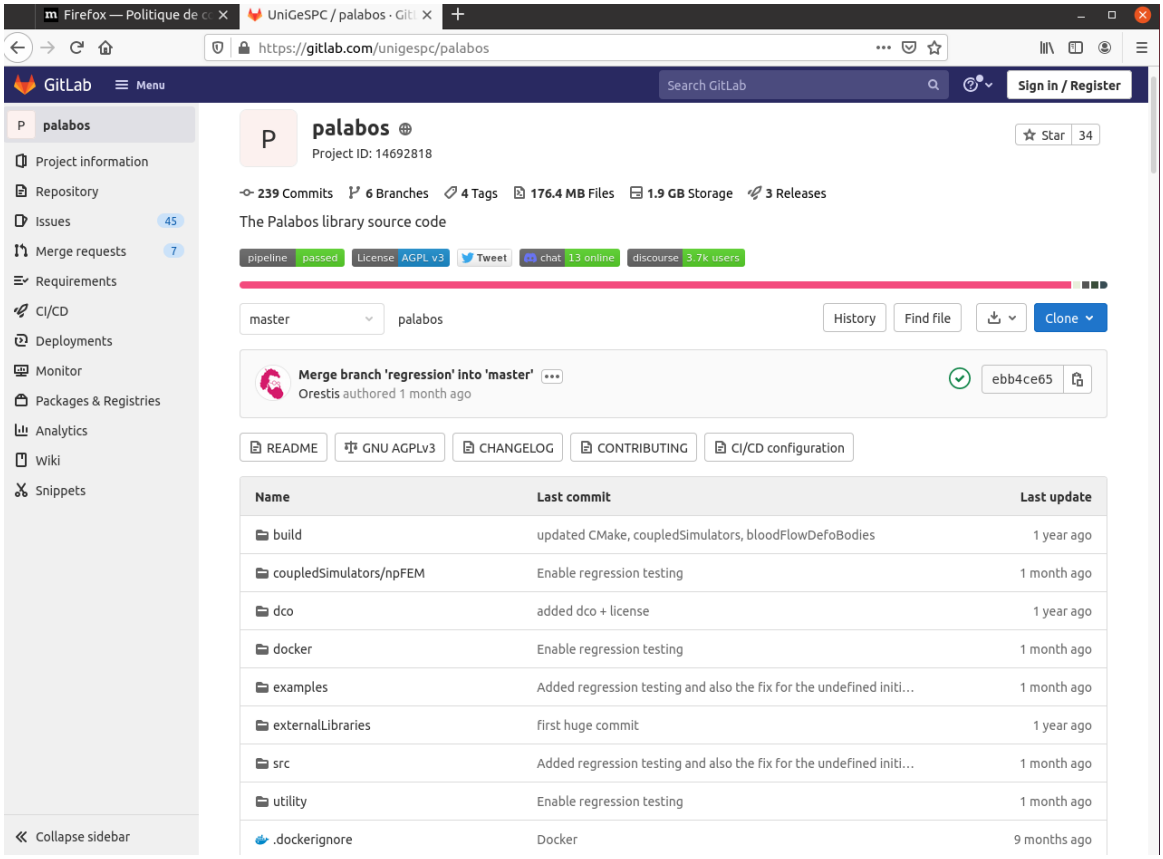

Figure 21:Palabos repository in Gitlab.

3. Validation of the LBM code

Verification and validation are the key methods for determining the accuracy and validity of computational methods of simulation. This section is devoted to the validation of the LBM code, using the LBM-BGK model based on the study of classical cases that are suitable for CFD code validation and benchmarking.

Based on the Boltzmann lattice method algorithm detailed in the previous chapters of this thesis, two codes were written based on the 2D codes available in the PALABOS library and were run for the simulation of classical flow configurations for incompressible fluids, quite used in the CFD code validation literature. Therefore, both the Poiseuille flow and the 2D circular cylinder flow are simulated using the LBM-BGK model.

3.1. Poiseuille flow

Plane Poiseuille flow is a flow created between two infinitely long parallel plates, separated by a distance h with a constant pressure gradient ΔP being applied in the direction of flow.

In this section, we compare the accuracy of the numerical solution generated by executing the LBM-BGK code in PALABOS to the analytical solution available in the CFD literature (refer to [36]).

Consider a fluid flow in a 2D duct formed by two parallel flat plates, separated by a distance $H=2H_0$ and length $L=10H_0$ (see figure 22).

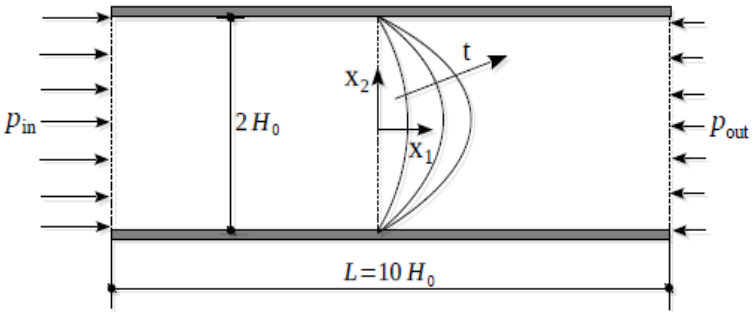

Figure 22: Poiseuille flow in a 2D duct

The flow is driven by a pressure difference $\Delta P = p_{in} - p_{out}$ between the inlet and outlet. The mechanical characteristics of the fluid are similar to those of water and pressure in normal conditions. At the walls of the pipe, the non-slip condition is applied. The geometrical, physical and numerical parameters used in the simulation are given in the following Table.

Parameter	**Value**	**Unit**
Duct width, *H*	$4.\ 10^{-4}$	[m]
Duct length, *L*	$2.\ 10^{-3}$	[m]
Density of the fluid, ρ_f	10^3	[$kg.m^{-2}$]
Dynamic viscosity, η	10^{-3}	[$N.s.m^{-1}$]
Imposed pressure difference ΔP	1	[$N.m^{-1}$]
Spatial resolution, *h*	10^{-5}	[m]
Lattice speed, *c*	1	[$m.s^{-1}$]
Relaxation time, τ	0.8	[/]

Table: geometrical, numerical and physical parameters.

The figures 23 to 27 show the code computed on Palabos, written in C++.

```
48 #include "palabos2D.h"
49 #include "palabos2D.hh"
50 #include <vector>
51 #include <cmath>
52 #include <iostream>
53 #include <fstream>
54 #include <iomanip>
55
56 using namespace plb;
57 using namespace plb::descriptors;
58 using namespace std;
59
60 typedef double T;
61 #define DESCRIPTOR D2Q9Descriptor
62
63 /// Velocity on the parabolic Poiseuille profile
64 T poiseuilleVelocity(plint iY, IncomprFlowParam<T> const& parameters) {
65     T y = (T)iY / parameters.getResolution();
66     return 4.*parameters.getLatticeU() * (y-y*y);
67 }
68
69 /// Linearly decreasing pressure profile
70 T poiseuillePressure(plint iX, IncomprFlowParam<T> const& parameters) {
71     T Lx = parameters.getNx()-1;
72     T Ly = parameters.getNy()-1;
73     return 8.*parameters.getLatticeNu()*parameters.getLatticeU() / (Ly*Ly) * (Lx/(T)2-(T)iX);
74 }
75
76 /// Convert pressure to density according to ideal gas law
77 T poiseuilleDensity(plint iX, IncomprFlowParam<T> const& parameters) {
78     return poiseuillePressure(iX,parameters)*DESCRIPTOR<T>::invCs2 + (T)1;
79 }
80
81 /// A functional, used to initialize the velocity for the boundary conditions
82 template<typename T>
83 class PoiseuilleVelocity {
84 public:
85     PoiseuilleVelocity(IncomprFlowParam<T> parameters_)
86         : parameters(parameters_)
87     { }
88     void operator()(plint iX, plint iY, Array<T,2>& u) const {
89         u[0] = poiseuilleVelocity(iY, parameters);
90         u[1] = T();
91     }
92 private:
93     IncomprFlowParam<T> parameters;
94 };
95
96 /// A functional, used to initialize the density for the boundary conditions
97 template<typename T>
98 class PoiseuilleDensity {
99 public:
100     PoiseuilleDensity(IncomprFlowParam<T> parameters_)
101         : parameters(parameters_)
```

Figure 23: Poiseuille source code 1

```
102     { }
103     T operator()(plint iX, plint iY) const {
104         return poiseuilleDensity(iX,parameters);
105     }
106 private:
107     IncomprFlowParam<T> parameters;
108 };
109
110 /// A functional, used to create an initial condition for with zero velocity,
111 ///   and linearly decreasing pressure.
112 template<typename T>
113 class PoiseuilleDensityAndZeroVelocity {
114 public:
115     PoiseuilleDensityAndZeroVelocity(IncomprFlowParam<T> parameters_)
116         : parameters(parameters_)
117     { }
118     void operator()(plint iX, plint iY, T& rho, Array<T,2>& u) const {
119         rho = poiseuilleDensity(iX,parameters);
120         u[0] = T();
121         u[1] = T();
122     }
123 private:
124     IncomprFlowParam<T> parameters;
125 };
126
127 enum InletOutletT {pressure, velocity};
128
129 void channelSetup( MultiBlockLattice2D<T,DESCRIPTOR>& lattice,
130                    IncomprFlowParam<T> const& parameters,
131                    OnLatticeBoundaryCondition2D<T,DESCRIPTOR>& boundaryCondition,
132                    InletOutletT inletOutlet )
133 {
134     const plint nx = parameters.getNx();
135     const plint ny = parameters.getNy();
136
137     // Note: The following approach illustrated here works only with boun-
138     //   daries which are located on the outmost cells of the lattice. For
139     //   boundaries inside the lattice, you need to use the version of
140     //   "setVelocityConditionOnBlockBoundaries" which takes two Box2D
141     //   arguments.
142
143     // Velocity boundary condition on bottom wall.
144     boundaryCondition.setVelocityConditionOnBlockBoundaries (
145             lattice, Box2D(0, nx-1, 0, 0) );
146     // Velocity boundary condition on top wall.
147     boundaryCondition.setVelocityConditionOnBlockBoundaries (
148             lattice, Box2D(0, nx-1, ny-1, ny-1) );
149
150     // Pressure resp. velocity boundary condition on the inlet and outlet.
151     if (inletOutlet == pressure) {
152         // Note: pressure boundary conditions are currently implemented
153         //   only for edges of the boundary, and not for corner nodes.
154         boundaryCondition.setPressureConditionOnBlockBoundaries (
155                 lattice, Box2D(0,0, 1,ny-2) );
```

Figure 24: Poiseuille source code 2

```
156         boundaryCondition.setPressureConditionOnBlockBoundaries (
157                 lattice, Box2D(nx-1,nx-1, 1,ny-2) );
158     }
159     else {
160         boundaryCondition.setVelocityConditionOnBlockBoundaries (
161                 lattice, Box2D(0,0, 1,ny-2) );
162         boundaryCondition.setVelocityConditionOnBlockBoundaries (
163                 lattice, Box2D(nx-1,nx-1, 1,ny-2) );
164     }
165
166     // Define the value of the imposed density on all nodes which have previously been
167     //   defined to be pressure boundary nodes.
168     setBoundaryDensity (
169             lattice, lattice.getBoundingBox(),
170             PoiseuilleDensity<T>(parameters) );
171     // Define the value of the imposed velocity on all nodes which have previously been
172     //   defined to be velocity boundary nodes.
173     setBoundaryVelocity (
174             lattice, lattice.getBoundingBox(),
175             PoiseuilleVelocity<T>(parameters) );
176     // Initialize all cells at an equilibrium distribution, with a velocity and density
177     //   value of the analytical Poiseuille solution.
178     initializeAtEquilibrium (
179            lattice, lattice.getBoundingBox(),
180            PoiseuilleDensityAndZeroVelocity<T>(parameters) );
181
182     // Call initialize to get the lattice ready for the simulation.
183     lattice.initialize();
184 }
185
186 /// Produce a GIF snapshot of the velocity-norm.
187 void writeGif(MultiBlockLattice2D<T,DESCRIPTOR>& lattice, plint iter)
188 {
189     const plint imSize = 600;
190
191     ImageWriter<T> imageWriter("leeloo");
192     imageWriter.writeScaledGif(createFileName("u", iter, 6),
193                                *computeVelocityNorm(lattice),
194                                imSize, imSize );
195 }
196
197 /// Write the full velocity and the velocity-norm into a VTK file.
198 void writeVTK(MultiBlockLattice2D<T,DESCRIPTOR>& lattice,
199               IncomprFlowParam<T> const& parameters, plint iter)
200 {
201     T dx = parameters.getDeltaX();
202     T dt = parameters.getDeltaT();
203     VtkImageOutput2D<T> vtkOut(createFileName("vtk", iter, 6), dx);
204     vtkOut.writeData<float>(*computeVelocityNorm(lattice), "velocityNorm", dx/dt);
205     vtkOut.writeData<2,float>(*computeVelocity(lattice), "velocity", dx/dt);
206 }
207
208 T computeRMSerror ( MultiBlockLattice2D<T,DESCRIPTOR>& lattice,
209                     IncomprFlowParam<T> const& parameters )
```

***Figure 25**:* Poiseuille source code 3

```
210 {
211     MultiTensorField2D<T,2> analyticalVelocity(lattice);
212     setToFunction( analyticalVelocity, analyticalVelocity.getBoundingBox(),
213                    PoiseuilleVelocity<T>(parameters) );
214     MultiTensorField2D<T,2> numericalVelocity(lattice);
215     computeVelocity(lattice, numericalVelocity, lattice.getBoundingBox());
216 
217            // Divide by lattice velocity to normalize the error
218     return 1./parameters.getLatticeU() *
219            // Compute RMS difference between analytical and numerical solution
220                std::sqrt( computeAverage( *computeNormSqr(
221                                *subtract(analyticalVelocity, numericalVelocity)
222                           ) ) );
223 }
224 
225 int main(int argc, char* argv[]) {
226     plbInit(&argc, &argv);
227 
228     global::directories().setOutputDir("./tmp/");
229 
230     IncomprFlowParam<T> parameters(
231             (T) 2e-2,  // uMax
232             (T) 5.,    // Re
233             60,        // N
234             3.,        // lx
235             1.         // ly
236     );
237     const T logT     = (T)0.1;
238     const T imSave   = (T)0.5;
239     const T vtkSave  = (T)2.;
240     const T maxT     = (T)15.1;
241     // Change this variable to "pressure" if you prefer a pressure boundary
242     //   condition with Poiseuille profile for the inlet and the outlet.
243     const InletOutletT inletOutlet = velocity;
244 
245     writeLogFile(parameters, "Poiseuille flow");
246 
247     MultiBlockLattice2D<T, DESCRIPTOR> lattice (
248               parameters.getNx(), parameters.getNy(),
249               new BGKdynamics<T,DESCRIPTOR>(parameters.getOmega()) );
250 
251     OnLatticeBoundaryCondition2D<T,DESCRIPTOR>*
252         boundaryCondition = createLocalBoundaryCondition2D<T,DESCRIPTOR>();
253 
254     channelSetup(lattice, parameters, *boundaryCondition, inletOutlet);
255 
256     // Main loop over time iterations.
257     for (plint iT=0; iT*parameters.getDeltaT()<maxT; ++iT) {
258        if (iT%parameters.nStep(imSave)==0) {
259            pcout << "Saving Gif ..." << endl;
260            writeGif(lattice, iT);
261        }
262 
263        if (iT%parameters.nStep(vtkSave)==0 && iT>0) {
264            pcout << "Saving VTK file ..." << endl;
```

Figure 26: Poiseuille source code 4

```
265             writeVTK(lattice, parameters, iT);
266         }
267
268         if (iT%parameters.nStep(logT)==0) {
269             pcout << "step " << iT
270                   << "; t=" << iT*parameters.getDeltaT()
271                   << "; RMS error=" << computeRMSerror(lattice, parameters);
272             Array<T,2> uCenter;
273             lattice.get(parameters.getNx()/2,parameters.getNy()/2).computeVelocity(uCenter);
274             pcout << "; center velocity=" << uCenter[0]/parameters.getLatticeU() << endl;
275         }
276
277         // Lattice Boltzmann iteration step.
278         lattice.collideAndStream();
279     }
280
281     delete boundaryCondition;
282 }
```

Figure 27: Poiseuille code 5

According to G. K. Batchelor in "An Introduction to Fluid Dynamics," [36] the analytical solution of the Poiseuille flow, taking into account the time evolution of the velocity profile of a duct cross-section, is written as follows:

$$u_1(x_2,t) = \frac{G.H_0^2}{2}\left[(1-\frac{x_2^2}{H_0^2}) - 4\sum_{n=1}^{\infty}\frac{\sin(\delta_n)}{\delta_n^3}.\cos(\frac{\delta_n x_2}{H_0}).\exp(\frac{-\delta_n^2\eta}{\rho_f H_0^2}.t)\right] \qquad (4.1)$$

Where:

- $\delta_n = \frac{(2n-1)\pi}{2}$ and
- $G = -\frac{1}{\eta}.\frac{\Delta p}{L}$

The analytical and numerical solutions are computed. The obtained results are exported to be plotted on a graph using Plotly, which is an open source graphing tool[6]. The velocity profile is displayed in the duct's center part.

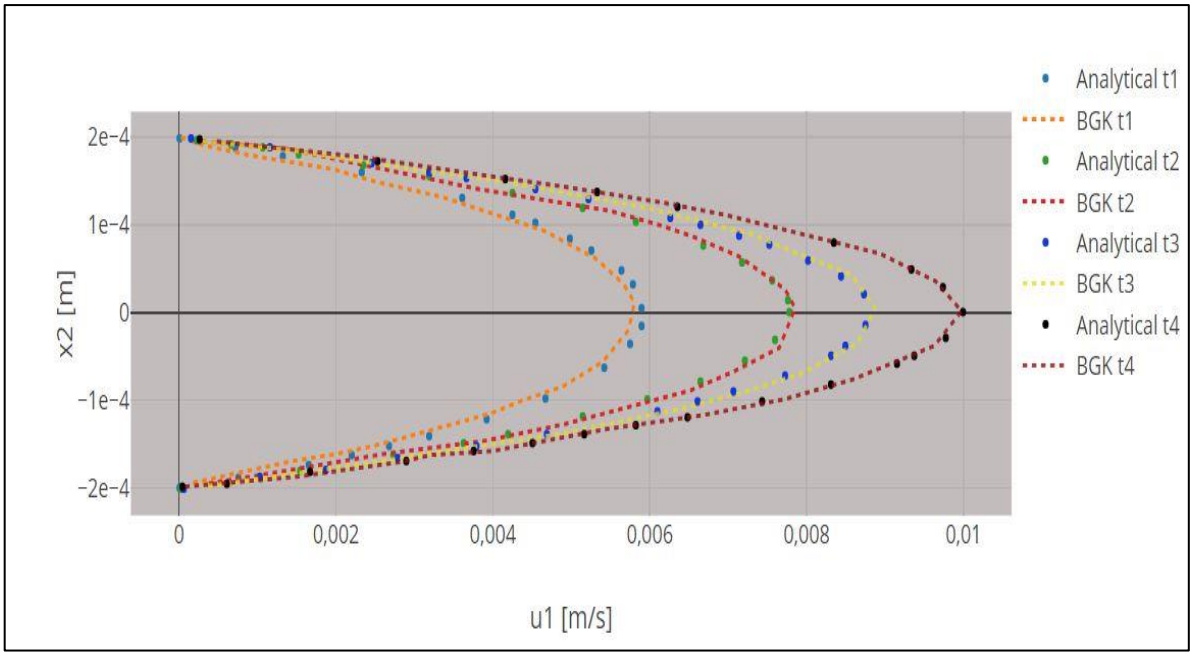

Figure 28: Comparaison between numerical results obtained by the LBM-BGK model and the analytical solution during the transient phase at t1 = 0,015s, t2 = 0,025s, t3 = 0,035s and permanent at t4 = 0,1s.

Figure 23 illustrates a very acceptable conformity between the numerical results, represented by the dashed lines, obtained by the LBM-BGK model and the analytical solution, represented by dots, during the transient phase at t1 = 0,015s, t2 = 0,025s, t3 = 0,035s and permanent at t4 = 0,1s.

3.2. Flow around a circular cylinder

The flow around a circular cylinder is a frequently used case of CFD to test various solid boundary conditions. In this study, we use the “bounce-back” condition. Curved cylinder walls are difficult to simulate accurately, especially in LB computations, which are often

[6] For more information, please refer to the tutorial section in https://plotly.com/chart-studio-help/tutorials/

performed on square lattices. The solver's performance in solving flow over complicated geometry may then be measured using the LBM, which can be compared to other well-known CFD methods. When the Reynolds number surpasses a critical value Re_{crit}, an unsteady flow occurs, resulting in a vortex street (von Kármán vortex street). As a result, the performance of unstable flows can be studied using the flow around a circular cylinder [37].

The flow is considered to be two-dimensional, isothermal, while the fluid is a Newtonian fluid, and of constant properties. Figure 24 shows the computational domain.

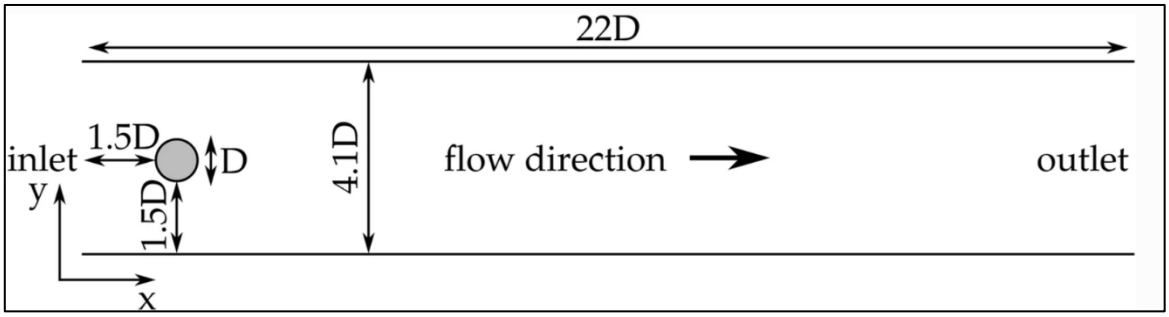

Figure 29: Computation domain

The code was tested in a simple setup—steady and unsteady 2D flow around a cylinder. Schäfer et al. [38] describe the benchmark in detail. A 2D domain with stationary no-slip walls on the upper and down sides is generated in Palabos.

The benchmarking is carried out at two different Reynolds numbers such that

$$\mathrm{Re}_{bm} = \frac{\overline{u_x}.D}{\nu} \qquad (4.2)$$

Where:

- $\overline{u_x}$ is the average flow velocity in the x-direction at the inlet,
- D is the cylinder diameter, and
- ν is the kinematic viscosity of the fluid.

The steady - state flow benchmark is established at Re_{bm} =20, while the unsteady case is examined at Re_{bm} =100. The values of the drag coefficient C_D and the lift coefficient C_L were tested:

$$C_D = \frac{2F_D}{\rho \overline{u_x}^2 D} \text{ , } C_L = \frac{2F_L}{\rho \overline{u_x}^2 D}$$

Where:

- F_D and F_L are respectively the drag and lift components of the forces applied on the cylinder
- ρ is the fluid density.

The resulting C_D and C_L were the stable values in the steady case, whereas in the unsteady case, these parameters were their maximum values, $C_D^{max}$and C_L^{max}, when a periodical steady state was achieved (the von Kármán vortex street). The momentum exchange approach is used in the current LBM simulation to calculate the fluid force on the circular cylinder. F_D and F_L were computed as the momentum change in a single direction during a time period Δt using the following equations:

$$F_D = \sum_{i=1}^{8} e_{x,i} \left[(f_i(B,t) + f_{\bar{i}}(N,t)) + (f_i(N,t) + f_{\bar{i}}(N+1,t)) \right] \Delta t \qquad (4.3)$$

$$F_L = \sum_{i=1}^{8} e_{y,i} \left[(f_i(B,t) + f_{\bar{i}}(N,t)) + (f_i(N,t) + f_{\bar{i}}(N+1,t)) \right] \Delta t \qquad (4.4)$$

Where:

- B denotes all solid cylinder boundary nodes,
- N denotes $B + e_i \Delta t$ if the point is fluid, and
- N+1 denotes $B + 2e_i \Delta t$.

The benchmarks were executed at various values of the diameter D: 8, 16, 32, and 64, which controlled the size of the domain and the grid resolution. Figures 25 and 26 depict the flow visualization at Re=20 and Re=100 by opening the generated GIF file in ImageMagick.

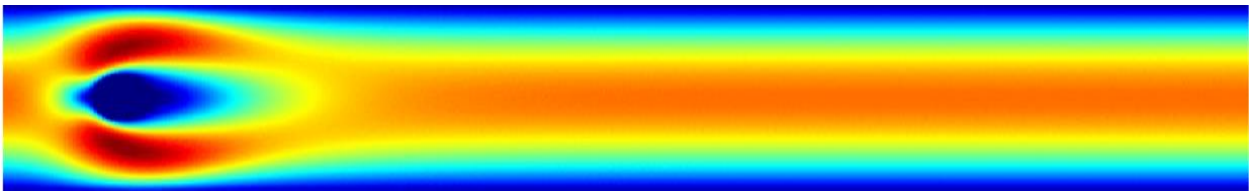

Figure 30: Simulation of the flow around a cylinder at Re=20.

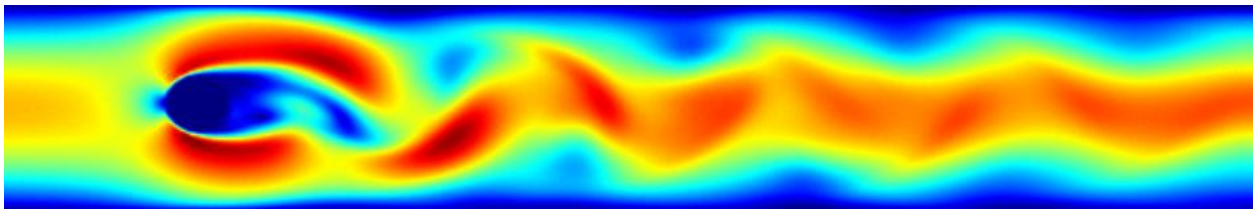

Figure 31: Simulation of the flow around a cylinder at Re=100.

It can be seen that in the case of flow at Re=20, there is an increase in velocity at the upper and lower levels of the cylinder wall, which is due to the deflection of the air streamlines. However, as no unsteady flow was created, no vortex is generated.
However, in the case of the flow at Re=100, a Von Karman vortex street is generated. This is explained by the increase in Reynolds number, which causes an unstable flow to appear. The oscillating motion of the conjunction zone, where both high velocity (red) and low velocity (blue) are found, explains why only values of C_D^{max} and C_L^{max} were collected.

The reference values were acquired using a traditional FE approach with an unstructured grid. The number of unknowns to be calculated is used to describe the grid resolution in this case. When Re=100, as the phenomenon to be simulated becomes time-dependent, the resolution is therefore expressed by both spatial and temporal unknowns. The benchmarking results are showed in the following table .

Re_{bm}	20		100			
D	C_D	C_L	C_D^{max}	C_L^{max}		
8	5.565043	- 0.000006	3.320698	0.596131		
16	5.494948	0.009826	3.235433	0.978701		
32	5.538439	0.011075	3.201211	1.004254		
64	5.586776	0.010525	3.227876	1.008553		
Reference values for a FE method with unstructured grid						
Unknowns					Spatial unknowns	Time unknowns
2298	5.4450	0.0200	2.8920	0.5540	2978	70
6297	5.5710	0.0130	3.2470	1.0740	29084	33
20487	5.5760	0.0110	3.2240	1.0060	29084	66

Table: Comparison of Palabos' CL and CD results with reference values.

At Re_{bm}=20, it can be seen that the performance is the lowest at the minimal diameter D =8, and it improves as the value of D rises. It can be also observed that the mistakes for C_L are larger than those for C_D.

The pattern at Re_{bm} =100 is similar to that at Re_{bm} =20; higher resolution computations performed better. Similarly, to when Re_{bm} =20 calculations were performed, the errors for $C_L^{max}$are larger than those for C_D^{max}.

General conclusion

Despite all of the existing CFD methods, tremendous progress is being made within the scientific community, leading to the creation of new and innovative approaches and methods. LBM has grown in recognition in recent years, and it is now the subject of countless theses. In light of the stakes and topicality of the method, the study conducted in this thesis takes on its full relevance.

We were able to analyze the evolution of the MBL by consulting many sources in order to better comprehend the method's theoretical roots. Once this knowledge was acquired, it was possible to describe the computations that enabled the formulation of the Boltzmann equation in discretized form, ready for implementation in a numerical environment.

The e-learning follow-up of the C++ programming language allowed the familiarisation with its paradigms and concepts. It also allowed to identify all the subtleties used in the Palabos library in order to develop a reliable and robust environment capable of generating a large amount of computing power without being costly for the memory resources, making it a tailor-made solution for complicated simulations.

After becoming acquainted with the Ubuntu interface and the Palabos library, the first computational codes were executed, producing a set of numerical data that was compared to the analytical and numerical data of other CFD approaches. The coherence and uniformity of the comparative results allowed us to demonstrate the LBM's efficiency and legitimacy as a method for modeling macroscopic flows from a mesoscopic description.

Perspectives

There are further aspects of LBM that can be examined. One of the key advantages of LBM is its ability to parallelize numerical calculations. Although parallelization was restricted to two processors in this study, an analysis of the efficiency and achievable encapsulation for a higher number of processors may be done.

Investigating the deployment of multi-phase flows or the simulation of three-dimensional flows may be of significance in evaluating LBM's capabilities.

The Lattice Boltzmann Method for incompressible flows in classical configurations was the focus of this project. Taking into account that lift and drag coefficients were calculated, a future interesting perspective would be to apply this method with minor changes to simulate the curved geometry of an airfoil and compare it the already acquired results in the aeronautical industry.

Because the LBM can only model incompressible flows, many efforts are being undertaken to overcome this limitation, as one of the key advantages of the LBM is its flexibility to implement different types of fluids simply by adjusting the collision operator. There are two potential models for implementing compressible flows in LBM: KT-LBM (Kataoka & Tsutahara, 2004) and QU-LBM (Qu, 2009).

Palabos also provides an additional library that combines the LBM with a modified version of FEM to model compressible fluids in supersonic speed.

References

[1] : Anderson, D.A, Tannehill J.C., Pletcher, R.H, Computational fluid mechanics and heat transfer.

[2] : Ferziger, M. Peric, A. Leonard, Computational Methods for Fluid Dynamics.

[3] : Howard H. Hu, in Fluid Mechanics (Fifth Edition).

[4] : C.A.J Fletcher: Computational techniques for fluid dynamics 1.

[5] : Patankar, Numerical Heat Transfer and Fluid Flow.

[6] : Krueger, T., Kusumaatmaja, H., Kuzmin, A., Shardt, O., Silva, G., & Viggen, E. M. (2016). The Lattice Boltzmann Method: Principles and Practice. (Graduate Texts in Physics). Springer.

[7] : Grindheim, J.V., Revhaug, I., Pedersen, E. and Solheim, P., 2018. ”A Simplified Solution to the Negative Tension Problem of a Three Dimensional Towed Underwater Cable Model”. In Proceedings of the 27º Congresso Internacional de Transporte Aquaviário, Construção Naval e Offshore (SOBENA 2018), Rio de Janeiro, Brazil.

[8] : LeVeque, Finite-Volume Methods for Hyperbolic Problems.

[9] : H. K. Versteeg, W. Malalasekera, An Introduction to Computational Fluid Dynamics: The Finite Volume Method.

[10] : Zanin, M.; Aitya, N.A.A.; Basilio, J.; Baumbach, J.; Benis, A.; Behera, C.K.; Bucholc, M.; Castiglione, F.; Chouvarda, I.;Comte, B.; et al.An Early Stage Researcher’s Primer on Systems Medicine Terminology. Netw. Syst. Med.2021.

[11] : Wang, Y., He, Z., Gupta, K.M., Shi, Q., & Lu, R. (2017). Molecular dynamics study on water desalination through functionalized nanoporous graphene. Carbon, 116, 120-127.

[12] : D. Frenkel, B. Smit, Understanding Molecular Simulation: From Algorithms to Applications.

[13] : M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids.

[14] : Wolf-Gladrow, Lattice-Gas Cellular Automata and Lattice Boltzmann Models.

[15] : Badarch, A. (2017). Application of macro and mesoscopic numerical models to hydraulic problems with solid substances.

[16] : S. Chen, G. Doolen, Lattice Boltzmann method for fluid flows, Annu. Rev. Fluid Mech. 30 (1998)

[17] : R.R. Nourgaliev, T.N. Dinh, T.G. Theofanous, D. Joseph, Int. J. Multiphas. Flow

[18] : S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond (Oxford UniversityPress, Oxford, 2001

[19] : B. Dünweg, A.J.C. Ladd, in Advances in Polymer Science

[20] : Gallagher, I. (2018). From Newton to Navier–Stokes, or how to connect fluid mechanics equations from microscopic to macroscopic scales. Bulletin of the American Mathematical Society, 56, 65-85.
[21] : Bhatnagar, P., Gross, E.P., & Krook, M. (1954). A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. Physical Review, 94, 511-525.
[22] : L.D. Landau, E.M. Lifshitz, Fluid Mechanics (Pergamon Press, Oxford, 1987).
[23] : Leo P. Kadanoff, Fractal Singularities in a Measure and How to Measure Singularities on a Fractal, Progress of Theoretical Physics Supplement, Volume 86, January 1986, Pages 383–386.
[24] : C.S. Wang Chang, G. Uhlenbeck, Transport phenomena in polyatomic gases. Tech. Rep.M604-6, University of Michigan (1951)
[25] : Sylvain Martin, Olivier Bonnefoy. Méthode de Boltzmann sur réseau - Application à la mécanique des fluides. Sciences fondamentales, Mathématiques, Techniques de l'ingénieur, pp.BM5220 V1, 2019, Sciences fondamentales | Mathématiques.
[26] : Chávez-Modena, Miguel. (2019). Analysis and optimization of multiple-relaxation lattice Boltzmann methods for under-resolved flow simulations.
[27] : Perumal D, Dr.Arumuga & Dass, Anoop. (2015). A Review on the development of lattice Boltzmann computation of macro fluid flows and heat transfer. Alexandria Engineering Journal.
[28] : Malaspinas, Orestis. (2010). Lattice Boltzmann method for the simulation of viscoelastic fluid flows.
[29] : LATT, Jonas. Hydrodynamic limit of lattice Boltzmann equations. Université de Genève. Thèse, 2007.
[30] : Mei, Renwei & Yu, Dazhi & Shyy, Wei & Luo, Li-Shi. (2002). Force Evaluation in the Lattice Boltzmann Method Involving Curved Geometry. Physical review. E, Statistical, nonlinear, and soft matter physics.
[31] : K. Han, Y. T. Feng, and D. R. J. Owen, Coupled lattice boltzmann and discrete element modelling of fluid-particle interaction problems.
[32] : Zou, Q., & He, X. (1995). On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. Physics of Fluids, 9, 1591-1598.
[33] : O. Malaspinas, Increasing stability and accuracy of the lattice Boltzmann scheme: recursivity and regularization, 2015.
[34] : F. Brogi, O. Malaspinas, B. Chopard, C. Bondadonna, Hermite regularization of the lattice Boltzmann method for open source computational aeroacoustics, J. Acoust. Soc. Amer.
[35] : Jonas Latt, Orestis Malaspinas, Dimitrios Kontaxakis, Andrea Parmigiani, Daniel Lagrava, Federico Brogi, Mohamed Ben Belgacem, Yann Thorimbert, Sébastien

Leclaire, Sha Li, Francesco Marson, Jonathan Lemus, Christos Kotsalos, Raphaël Conradin, Christophe Coreixas, Rémy Petkantchin, Franck Raynaud, Joël Beny, Bastien Chopard, Palabos: Parallel Lattice Boltzmann Solver, Computers & Mathematics with Applications, Volume 81, 2021.

[36] : G. K. Batchelor, "An Introduction to Fluid Dynamics," Cambridge University Press, Cambridge, 1967.

[37] : Abas A, Gan ZL, Ishak MHH, Nasip NS, Fuat Khor S (2015) Lattice Boltzmann study of vortex street in pressurized underfill manufacturing process.

[38] : Schäfer M, Turek S, Durst F, Krause E, Rannacher R (1996) Benchmark computations of laminar flow around a cylinder.