# POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea Specialistica in Ingegneria Meccanica


Lattice Boltzmann Method for Three Dimensional
Fluid Flow Simulation

Relatore: Prof. Gianluca D'ERRICO
Corellatore: Prof. Guan YEOH

Tesi di Laurea Specialistica di:
Andrea Angelo LOMAZZI
Matr. 753219

Anno Accademico 2010 - 2011

## Abstract

An analysis of the Lattice Boltzmann Method (LBM) has been developed as an alternating Computational Fluid Dynamic (CFD) method based on the collective behavior of microscopic particles. This numerical method contrasts with the traditional approaches which usually consider macroscopic descriptions by using Navier-Stokes (NS) equations.

The specific purpose of the present thesis is to supply a comprehensive description of the field providing a source code for practical applications under GNU copyright.

In the first part the review of the fundamental macroscopic variables is emphasized to outline the NS equations with advantages and drawbacks.

Numerical methods are investigated in the second part, evolving from Cellular Automata, the most recent Lattice Gas Cellular Automata and specially its extension, the LBM.

The last part of this thesis shows the results of the method comparing to a traditional Navier-Stokes solver. It is also pointed out the advantages adopted with LBM approach in nano-scale multiphase fluid flow under complex boundary conditions.

## Acknowledgments

I would like to express my gratitude to Prof. Guan Yeoh for the advices, support and freedom I have enjoyed during my exchange in Australia that allowed me to follow my interests and points of view.

I would like to extend my gratitude to Prof. Gianluca D'Errico for supporting me through the course of this thesis work from Italy.

I also would like to thank the School of Mechanical and Manufacturing Engineering at UNSW and the researchers for helping me for my future prospects with support, collaboration, and friendship.

My warm thanks to Luigi Dolci and Amie Baines for the huge amount of food they friendly always shared with me at 56, Flinders. Thanks Luigi for being the big brother I have never had.

The welcome tranquility of Sydney, one of the world's greatest city, the gardens, the national parks, the ocean, the beaches, the easygoing and friendly Sydneysiders, and the Oz Lifestyle, all contributed significantly to provide inspiration to see this work through to its conclusion.

The Lobby, Candy's Apartment, and NSW State League Football Referees federation supported me through my Aussie Exchange Program with financial support.

Last but not the least, I would like to thank my family for their love all these years of studies specially for teaching me how to stand directly after falling and finding always the bright side of the life.  Finally, I would like to dedicate this thesis to them.

## Contents

# Introduction

The Lattice Boltzmann Method represents a new powerful approach for a simulation of a wide range of complex flow problems in computational fluid dynamic. Based on mesoscopic kinetic equations, LBM constructs a simplified macroscopic model under the collective behavior of fluid particles.

Historically this new method belongs to the class of Cellular Automata where a physical system is discretized in space and time, made up of identical cells. Each cell is characterized by Boolean variables which evolves in function of the state of the neighbors.

The evolution of CA is described by Lattice Gas Cellular Automata introduced in 1973 by Hardy, Pomeau, and de Pazzis. The dynamic of the flow is simultaneously shown by the moving and colliding of point-particles. Particles are allowed to move on a discrete lattice and local collisions conserved mass and momentum.

From LGCA, the LBM was developed by the description of the evolution of the single-particle distribution function. Each of these particles is given a discrete set of velocities for traveling from one node on the grid to another. The particles are redistributed on each node according to the rules that recover the collision process. Through the simplification of the collision terms proposed by Bhatnagar, Gross, and Krook (BGK), Lattice Boltzmann BGK was improved using an approximation to the equilibrium distribution function in order to cut down the cost of computation. An important feature of this approximation is the conservation of mass, momentum, and energy.

The model used in the present thesis is D3Q27, twenty-seven nodes in a three dimension mesh, which represents the evolution of the D2Q9, nine nodes in a two-dimension domain. An improvement in the source code is outlined by the size of the mesh which is a user-defined variable. The results obtained show that the method chosen for writing down the computational code is one of the most efficiency and straightforward until now.

# Chapter 1

# Fundamentals Fluid Mechanics

A review of the fundamentals is outlined in this chapter in order to be familiar with the basic concepts of fluid mechanics. Details are avoided, giving more emphasis to the physics.

## 1.1 Macroscopic Description of Fluid Flows

Fluid flows provide opportunities for mathematical modeling. Continuum approximation is applied taking into account the macroscopic flow configuration.

Let $L$ denote the characteristic length associated with the length of the domain and $L_\Delta$ the size of the infinitesimally small volume elements; $L_\Delta \ll L$. The hydrodynamic variables like density $\rho$, pressure $p$, temperature $T$ and velocity $\vec{u}$ are meaningful at scale of the elementary fluid elements. They represent average values computed from particles.

The derivation of macroscopic fluid flow description gives the general conservation equations for the elementary fluid elements,

$$\partial_t \rho + \nabla(\rho u) = 0 \tag{1.1}$$

$$\partial_t(\rho u) + \nabla \prod = 0 \tag{1.2}$$

Where the partial derivatives $\partial_t = \partial/\partial t$ and $\partial_\alpha = \partial/\partial r_\alpha$ express variation in quantities with respect to time and space. The divergence, for instance, measures net variation of a quantity in spatial space. Equation (1.1) and (1.2) ensure mass and momentum conservation for elementary fluid elements. The momentum-flux tensor $\prod_{\alpha\beta}$ gives the flux of the $\alpha$ component of the momentum in $\beta$ direction.

In order to obtain a closed description of the fluid dynamic, additional restrictions have to be considered.

## 1.2 The Incompressible Navier-Stokes Equations

Considering a constant density, conservation equation is written as,

$$\nabla u = 0 \tag{1.3}$$

And

$$\rho \partial_t u + \rho u \cdot \nabla u = -\nabla p + \nabla \prod^{visc} + F \tag{1.4}$$

Where $F = \rho a$ refers to an external body force, for instance gravity. Rearranging the equation (1.4),

$$\rho\left(\partial_t u_\alpha + u_\beta \partial_\beta u_\alpha\right) = -\partial_\alpha p + \partial_\beta \prod_{\alpha\beta}^{visc} + F_\alpha \tag{1.5}$$

On the left hand side of the formula, the inertial of the volume is shown as, $D_t = \partial_t + u_\beta \partial_\beta$ and on the right he forces acting on the volume are summed.

Coming up to the dynamical description, another restriction has to be imposed on the viscous stresses. In a Newtonian fluid the viscous stresses $\prod_{\alpha\beta}^{visc}$ are linearly proportional to the strain rate defined as,

$$S_{\gamma\delta} = \left(\partial_\delta u_\gamma + \partial_\gamma u_\delta\right)\Big/2 \tag{1.6}$$

$$\prod_{\alpha\beta}^{visc} = \Phi_{\alpha\beta\gamma\delta} S_{\gamma\delta} \tag{1.7}$$

Where the four-rank tensor $\Phi_{\alpha\beta\gamma\delta}$ measures the viscosity of the medium as internal friction of the fluid in function of temperature and pressure.

An additional reduction is made by considering isotropic and incompressible fluids.

$$\nabla \cdot \prod^{visc} = \partial_\beta \prod_{\alpha\beta}^{visc} = \partial_\beta \mu\left(\partial_\beta u_\alpha + \partial_\alpha u_\beta\right) = \mu \partial_\beta \partial_\beta u_\alpha \tag{1.8}$$

Substituting this source term into the momentum equation (1.4), the Navier-Stokes equation for an incompressible, isotropic Newtonian fluid is,

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho}\nabla p + \upsilon \nabla^2 u + a \tag{1.9}$$

Where $\upsilon = \mu/\rho$ is the kinematic viscosity and the term involving the laplacian of the velocity field, $\upsilon\nabla^2 u$, represents diffusion of momentum. The second term on the right hand side is the only nonlinear ones; it is called convective term. Numerical methods are required to obtain approximate solutions for the equations (1.3) and (1.8) together with the boundary conditions.

Microscopic details of the fluid are completely ignored in the NS equations. For instance, viscosity is an important terms to study the interaction between

particles at the atomic level. In the Navier-Stokes equations, the pysical properties of the fluid are modeled with material parameters obtained from experimental measurements; that is a reason why NS equations are applicable to so many different fluids.

### 1.3 Reynolds Number

**The Navier-Stokes equations incorporates a very important mathematical property named dynamic similarity between fluids flow.**

This parameter is the Reynolds Number, Re, a dimensionless number that measures the effects of the inertia and viscosity within the flow field. Generally, it is used to characterize the behavior of the flow, such as laminar (Re<2300), transient (2300<Re<4000), or turbulent flow (Re>4000). Roughly, when an object is moved trough the ambience, the gas molecules of the atmosphere near to the object are disturbed and they generate forces. The variables that modify the magnitude of the forces are the macroscopic quantities of the object and the behavior of the gas.

Firstly, relevant dimensionless variables are defined as,

$$r_\alpha^* = \frac{r_\alpha}{L},\ u_\alpha^* = \frac{u_\alpha}{L},\ t^* = t\frac{U}{L},\ P^* = P\frac{1}{U^2} = \frac{p}{\rho}\frac{1}{U^2},\ a_\alpha^* = a_\alpha\frac{L}{U^2} \tag{1.10}$$

Where L and U refer to the characteristic lenght scale and fluid flow velocity of the macroscopic system. NS equations are easily transformed into dimensionless form considering L and U constants,

$$\frac{\partial u_\beta}{\partial r_\alpha} = \frac{\partial\left(u_\beta^* U\right)}{\partial\left(r_\alpha^* L\right)} = \frac{U}{L}\frac{\partial u_\beta^*}{\partial r_\alpha^*} \tag{1.11}$$

Hence, by using the definitions in equation (1.9), it is straightforward to obtain,

$$\frac{U^2}{L}\frac{\partial u^*}{\partial t^*}+\frac{U^2}{L}u^*\cdot\nabla_* u^*=-\frac{U^2}{L}\nabla_* P^*+\upsilon\frac{U}{L^2}\nabla_*^2 u^*+\frac{U^2}{L}a^* \tag{1.12}$$

$$\frac{\partial u^*}{\partial t^*}+u^*\cdot\nabla_* u^*=-\nabla_* P^*+\frac{1}{Re}\nabla_*^2 u^*+a^* \tag{1.13}$$

$$Re=\frac{\rho Lu}{\mu}=\frac{Lu}{\upsilon} \tag{1.14}$$

Where $Re$ provides an estimate for the ratio of inertial forces to viscous forces quantifying their relative importance for given flow conditions. For instance, if Re number is high, inertial forces dominate the viscous forces and the flow becomes unstable.

### 1.4 Poiseuille Flow

The Poiseuille Flow is an important type of flow driven by a pressure gradient between two parallel surfaces. The flow is characterized by a parabolic velocity profile with a maximum value in the middle of the pipe and a value of zero at the walls (no-slip boundaries). The parabolic velocity profile is given by,

$$u\left(x\right)=\frac{G^*}{2\mu}\left(a^2-u^2\right) \tag{1.15}$$

where $G^*$ is the linear pressure gradient $\left(P_{in}-P_{out}\right)\Big/L$ .

Code 1.1 shows how to initialize the simulation of the velocity, `uProf(y)`, to Poiseuille profile in programming language. `yDim` is the maximum dimension of the domain and `uMax` is the maximum velocity defined a priori.

```
!         ========================================================
!         Computation of Poiseuille profile for the inlet/outlet
!         ========================================================
FUNCTION uProf(y)
    USE simParam, ONLY: yDIm, uMax
    implicit none

    integer, INTENT(IN):: y
    double precision:: radius, uProf

    radius = dble(yDim-1) * 0.5d0
    uProf  = -uMax * ((abs(1 - dble(y-1) / radius))**2 - 1.0d0)
END FUNCTION uProf
```

Code 1.1. Computation of Poiseuille profile for the inlet/outlet.

# Chapter 2

# Cellular Automata

Cellular Automata (CA) are discrete dynamical systems that model complex behavior based on simple rules animating cells on a lattice.

## 2.1 Introduction

While most of the systems can be broken down into identical components, each of them with the same rules, the cellulars act together in order to create the complex behavior of the system [1].

In the simplest CA form, space is represented by a uniform discretized grid, where each cell has a finite state memory, the automata.

At each time step, all the cells change their current status synchronously to the next state according to a set of “local rules”.

The complexity of the pattern generated is the main reason that makes this method interesting. CA can be seen as an algorithm imposed on a group of cellulars that occupy a proper position on a grid [2].

## 2.2 History

The history of CA dates back the nineteen-forties, when the Polish-Jewish mathematician Stanisław Ulam started to study the evolution of graphic construction under simple rules. His main goal was to construct self-replicating machines, building copies of themselves. In the nineteen-fifties, Von Neumann suggested to consider a lattice space where each point had a finite number of connections to certain of its “neighbors” [3]. The two mathematicians worked together during this period to develop an automata system.

Thanks to the lattice network studied with Enrico Fermi, Ulam realized a lattice model for studying crystal growing governed by simple rules. He found out that simple rules may generate complex and repetitive figure.

At the same time Von Neumann, involved in developing the self-reproducing machines, figured out a new way to represent a physical universe with a set of elementary algorithms. He designed a two-dimensional self-reproduction automata in which each cell would change its twenty-nine states according to rules of the nearest neighboring cells.

After the diffusion of the Cellular Automata, other people developed research in this field; the German civil engineer, Konrad Zuse, began to create a binary calculating machine which involved program control and floating point arithmetic. The new idea of cellular computer started with high-capacity memory and relays. Zuse published also a monograph concerning the application of CA to physical problems such as hydrodynamics, electrodynamics and quantum theory.

The English mathematician John Horton Conway tried to set out simple rules for a two-dimensional, two state automata, self-replicating machine. His work, following Ulam’s ideas, was called the “Game of Life” in which he simulated the evolution of a real society from a causal initial configuration through easy principles.

In 1973, Hardy, Pomeau, and De Pazzis simulated a cellular automata in a fluid flow. The model, called HPP from the initials of the three authors, is characterized by a mass and momentum conservation. However, it does not yield the desired Navier-Stokes equations in the macroscopic field.

After more than a decade, in 1986, Frisch, Hasslacher, and Pomeau tried a higher symmetry with CA over a hexagonal lattice called FHP. They finally were able to lead to the Navier-Stokes equations.

At the beginning of the eighties, the British scientist, Stephen Wolfram, carried out a one-dimensional CA where each cell may be interpreted as a k-digit number with two states. With his research, he came up with a series of rules able to change the state of the cells, from active to inactive and vice versa, as a function of the neighborhood.

In recent years, CA provides a basis and a new perspective for mathematical models for a wide variety of complex and natural phenomena described before by partial differential equations.

## 2.3 One-dimensional Cellular Automata

The basic idea behind the one-dimensional model consists of an infinite grid where each of the cells has a specific *k-value*, active or inactive, occupied or empty, alive or dead, black or white. According to the $\Phi_1$ rule, the value $a_i$ of position $i$ evolves as a function of the neighbors $a_{i-1}^t$ and $a_{i+1}^t$.

$$a_i^{(t+1)} = \Phi_1\left[a_{i-1}^t, a_i^t, a_{i+1}^t\right] \tag{2.1}$$

For instance, a system used for one-dimensional CA is the Wolfram Code, or Totalistic Cellular Automaton, where “totalistic” means described by the total or the average of the neighborhood.

## 2.4 Two-dimensional CA

In a two-dimensional cellular automata, the state of the cells is based on a $m \times n$ binary matrix. Mathematically, the state $a_{i,j}^{(t+1)}$ of the $i$ and $j$ cell is given by the formula,

$$a_{ij}^{(t)} = \Phi_2 \left[ a_{i-1,j-1}^t, a_{i-1,j}^t, a_{i-1,j+1}^t, a_{i,j-1}^t, a_{i,j+1}^t, a_{i+1,j-1}^t a_{i+1,j}^t, a_{i+1,j+1}^t \right] \quad (2.2)$$

where $\Phi_2$ is a Boolean function of eight variables (*Moore neighborhood*).

A easy way to illustrate two dimensional automata concept is described by the “wave” in a sport stadium; each person reacts with the state of his neighbor, standing up or sitting down.

### 2.4.1 The “Game of Life”

The Game of Life is a zero-player game that use an artificial intelligence simulating the evolution of a society of organisms basing on a certain rules. It was introduced by Conway in the nineteen seventies as a two-dimensional CA.

This mathematical game is a very simple and powerful demonstration of how even few rules may not only cause order, created out of chaos from an initial condition, but also give rise to self-replicating structures.

The universe is defined as an infinite two-dimensional orthogonal grid made by square cells each of them have two possible states: live, shown by a marker, or dead, empty square. At each time step, based on the interaction between neighbors, we have the following four rules:

- any live cell with two or three neighbors stay alive (generation);

- any dead cell with three live neighbors around becomes a live cell (reproduction);
- any live cell with more than three neighbors dies (overcrowding);
- any live cell with less than two neighbors dies (underpopulated);

It is a fascinating game that give us a chance to observe what it will happen in the future; life is full of surprises.

## 2.5 From PDEs to CA

A different approach to describe and simulate the predicting behavior of natural phenomena may be seen using numerical method.

CA represent a discrete counterpart of Partial Differential Equations (PDEs) with the benefit of being easily simulated, highly parallelizable [4], changing the perspective of the model to a local view.

Taking into account a general PDE (2.3), $u$ defines the variable with respect to $i$, time, and $j$, space; $m\left(u_{i,j}\right)$ contains all the $u_{i,j}$ terms; $n\left(u_{i,j}\right)$ defines all the Laplacian terms with respect to space; all the gradients applied are within $o\left(u_{i,j}\right)$ term.

$$f\left(u_{i,j}\right)=\frac{\partial u}{\partial t}=m\left(u_{i,j}\right)+\nabla_x^2 n\left(u_{i,j}\right)+\nabla_x o\left(u_{i,j}\right) \tag{2.3}$$

In order to transform the PDE (2.3) to CA, two different approaches can be followed: Backward Euler's Method and Forward Euler's Method.

The approximation of a differential equation through Forward Euler's Method is given by the formula below,

$$u_{i+1,j} = u_{i,j} + h_t f\left(u_{i,j}\right) \tag{2.4}$$

Where the variable $h_t$ defines the step size between two time values. Substituting the partial differential equation into the equation (2.4),

$$u_{i+1,j} = u_{i,j} + h_t\left( m\left(u_{i,j}\right) + \frac{\left(n\left(u_{i,j+1}\right) - 2n\left(u_{i,j}\right) + n\left(u_{i,j-1}\right)\right)}{h_x^2} + \frac{o\left(u_{i,j+1}\right) - o\left(u_{i,j-1}\right)}{2h_x} \right) \tag{2.5}$$

A more stable approach for larger $h_t$ values is outlined by Backeard Euler's Method using a first order Taylor series,

$$u_{i+1,j} = u_{i,j} + \frac{h_t f\left(u_{i,j}\right)}{1 - h_t \left.\dfrac{\partial f(u)}{\partial u}\right|_{i,j}} \tag{2.6}$$

Substituting the partial differential equation (2.3) into equation (2.6) the CA model is obtained by,

$$u_{i+1,j} = u_{i,j} + \frac{h_t}{1 - h_t \left.\dfrac{\partial m(u)}{\partial u}\right|_{i,j}} \left( m\left(u_{i,j}\right) + \frac{n\left(u_{i,j+1}\right) - 2n\left(u_{i,j}\right) + n\left(u_{i,j-1}\right)}{h_x^2} + \frac{o\left(u_{i,j+1}\right) - o\left(u_{i,j-1}\right)}{2h_x} \right) \tag{2.7}$$

Where the term $h_x$ is the step size variable for space.

Through this model, we obtain a much more simple mathematical method than the differential equation, just including addition, subtraction, multiplication and division operations.

New studies are developing in this research field, mostly in order to set up a software tool transforming PDE to CA and performing simulation.

## 2.6 CA Applications

We have already demonstrated in the “game of life”, chapter 2.4.1, that complexity can arise from very simple rules.

Several interesting applications have been proposed for CA method such as cryptography and simulation of single particle system.

Usually this method is worth when the inherent parallelism improves the runtime of algorithms, but the ability to implement models efficiently remains the best behavior which drives applications.

In recent years, SIMD (single instruction multiple data) hardware makes the CA such as compelling model for 3D graphics cards; it allows to a really good performance in the simulation of cellular space [5].

Another application is described by Hartka [6] in which CA is used in order to optimize circuitry.

De Garis, Korkin, Peréz-Uribe and Sanchez [7] tried to implement a CA model in neural networks using programmable hardware.

Burstedde et al. [8] implemented CA in the simulation of pedestrian with a non-deterministic model; this approach can be useful to optimize buildings for fast evacuation.

A novel approach at object recognition in image processing is outlined by Fey and Schmidt [9], where they have an CA overlaid on an image sensor. The sensor changes the state of each cell in function of the brightness of the light. In only 10 milliseconds, their chip is able to process a single image of

640x480 pixels, extracting the positions and types of simple geometric shapes in taken images.

# Chapter 3

# Lattice Gas Cellular Automata

Lattice Gas Cellular Automata (LGCA) are discrete molecular models based on the movement of particles on a lattice in a microscopic level, mimicking a fully dynamic fluid model. The derivation of a mesoscopic description of fluid flows usually starts from the atomic perspective. It's easily demonstrable that LGCA technique can be reproduced by macroscopic Navier-Stokes equations.

## 3.1 Introduction to LGCA

Lattice Gas automata have been introduced in the early nineteen seventies by the French Yves Pomeau. By the late seventies, the Information Mechanics Group at MIT developed a new idea of building a special-purpose machine to simulate physic models [10]. During this time, S. Wolfram visited the MIT group and, stimulating by their work, started to investigate on statistical mechanics for large fluid systems.

In the nineteen seventy three, Hardy, De Pazzis, and Pomeau introduced the so called HPP Lattice Gas Cellular Model using a square lattice [11]. The implementation of the motion of the particles was the interesting reason of making this model one of the greater than traditional CA. Due to the inadequate symmetry of HPP, this method failed simulating the Navier-Stokes Equations.

Frisch, Hasslacher and Pomeau set up a new model based on hexagonal lattice-gas model (FHP) [12]. This new model recovered isotropic flow in the continuum limit with a triangular mesh holding six momentum states.

This methodology was extended to a three-dimensional model developed with a hypercubic lattice. The Face Centered Hyper-Cubic lattice (FCHC) is a regular crystal lattice that leads to isotropic Navier-Stokes flows. An important tool to characterize and analyze this problem was found using a four-dimensional Face Centered Hyper Cubic lattice with twenty-four nearest neighbors. Research is still underway due to the large demand for local memory needed to initialize properly the complex iterations.

## 3.2 The HHP Lattice Gas Cellular Automata

Many macroscopically observable phenomena related to fluid dynamics have their origin in the underlying microscopic world. For instance, surface tension represents the cohesive forces between molecules responsible for phase separation in the macroscopic scale. With an additional procedure, for instance with averaging, hydrodynamic variables are then computed from the variables of intermediate scale. This alternating modeling philosophy is inherent in the mesoscopic description of the fluid flows.

The first and simplest model for the simulation of gas particles in a microscopic point of view, was called HPP, named according to the initials of the three authors.

This model is a two-dimensional LGCA model with a discrete time, space and velocity and unit link length. It is defined over a square lattice where the particles, viewed as made of “Boolean molecules”, move on the grid with a constant mass and velocity.

The behavior of the model is characterized by different steps; first of all a analysis of the model with a microscopic point of view; then, updating synchronously all the cells; finally taking into account local iterations with neighborhood and obtaining macroscopic quantities such as density and

velocity of the flow. Through a microscopic model we are able to describe the macroscopic evolution of the fluid dynamic.

### 3.2.1 Model

The square lattice of the HPP model has four cells each of them associated to the nearest neighbor. A cell is connected with each link at all nodes and it may be empty, defined by the number 0, or occupied, defined by the number one (Boolean variables).

There are up to four molecules (particles), one at each vertex, with equal mass, unit speed whose velocities point in one of the four link directions.

The particles move along the edges of the lattice at each time step; each site (node) is characterized by the Pauli exclusion principle[1]. The evolution of the particles brings to local *collisions* and *streaming* along the direction of their velocity. When two particles enter in the same node and with opposite direction, they rotate by 90° conserving the mass and the momentum.

Collision rules are showed in the figure 3.1; first case defined as “a”, shows the *streaming* step: the particle moves on the lattice following his original path. From the second and the third it is shown how local *collision* between two particles occurred.

[1] The Pauli exclusion principle defined that the simultaneous occupation of a vertex by identical molecules is forbidden.

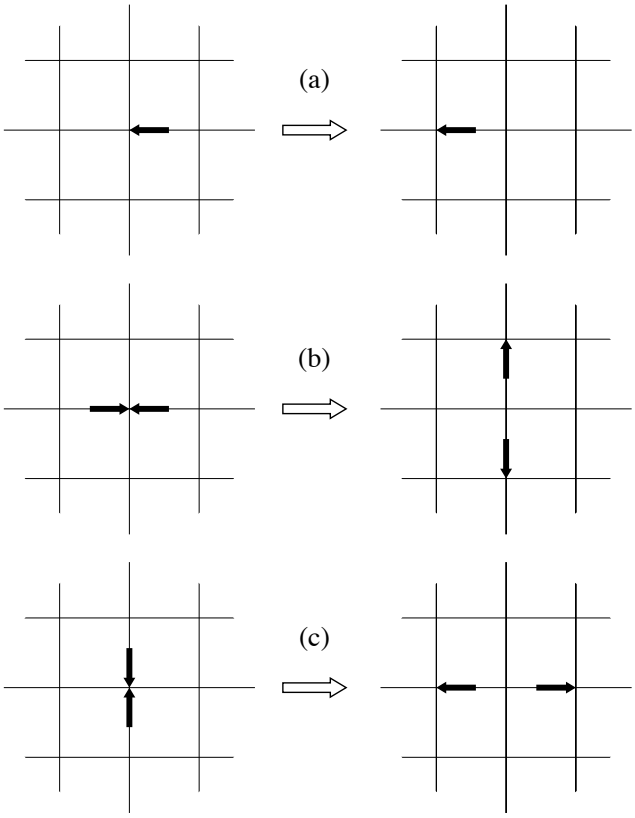


Figure 3.1 Streaming and Collision rules for colliding particles.

As already mentioned before, HPP model suffers some limits; first of all the total number of particles moving on the lattice is fixed. That does not represent the real continuum fluid, obtaining an undesirable feature of the model. A insufficient degree of rotational symmetry does not lead to the desired Navier-Stokes equations.

### 3.2.1 Coarse Graining

Coarse Graining is a method studied in order to calculate at least the mean value for mass and momentum density. This method is used not only in thermodynamic or fluid dynamic applications but, biomolecular simulations as well [13].

Looking at the behavior of this method, the initial step is based on the initialization of each node. Usually the entire domain is divided into subdomains, 32 or 64 times bigger than the size of the square lattice, where the mass and momentum density are approximated. Averaging over neighboring nodes $n_i$ (occupation number), the mean occupation number $N_i$, which may be 0 or 1, is obtained with respect to the coordinates of the nodes $\vec{r}$ and the time $t$.

$$N_i\left(\vec{x},t\right)=\left\langle n_i\left(\vec{r},t\right)\right\rangle \tag{3.1}$$

The index $i$ indicates the four possible nodes in the square lattice ($i=1,2,3,4$).

The mass density $\rho\left(\vec{x},t\right)$ and momentum density $\vec{j}\left(\vec{x},t\right)$ are obtained by the following formulas,

$$\rho\left(\vec{x},t\right)=\sum_{i=1}^{4}N_i\left(\vec{x},t\right) \tag{3.2}$$

$$\vec{j}\left(\vec{x},t\right)=\rho\vec{u}=\sum_{i=1}^{4}c_iN_i\left(\vec{x},t\right) \tag{3.3}$$

where $\vec{u}$ is the flow velocity with the direction $c_i$.

The purpose of this method is to realized a reliable and low noisy model averaging on large enough subdomains under the limit of the core memory.

### 3.3 The FHP Lattice Gas Cellular Automata

FHP, named after the initials of the three authors, is a two-dimensional model based on an hexagonal lattice. The particles are able to move on six sites fig. 3.2, all located at the same distance to the central node respectively, holding six possible lattice velocities. This moving is defined by a discrete time step with a constant speed.

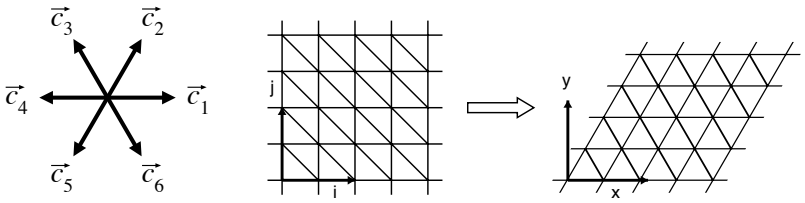


Fig. 3.2 The hexagonal lattice.

This model observes the hexagonal symmetry which prevents the model from yielding the Navier-Stokes equations in macroscopic limit [14]. On the contrary, the restrictions of the model are due to the conservation laws. The Pauli exclusion principle ensures that all the cells may be empty or occupied by at most one particle (Boolean variables). Mass and momentum density is conserved and the number of particles as well.

When two or three particles collide in the same site, as I describe in the previous chapter, all the particles are reflected. The deflection of two particles leads to a randomly clockwise or counterclockwise rotation by 60°; instead, if three particles meet at a node in a symmetry configuration they just invert their direction by 120° (Figure 3.3).

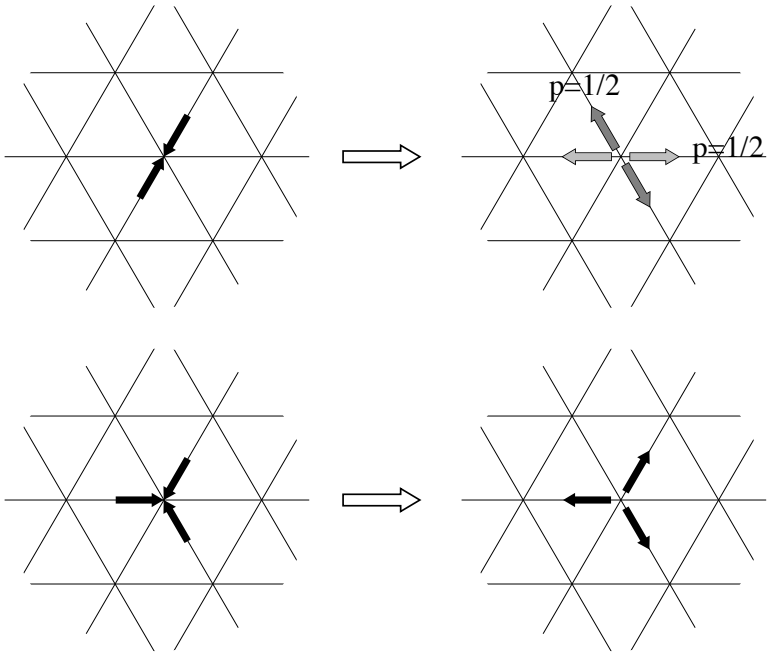


Fig. 3.3 FHP collision rules.

### 3.4 Dynamic of LGCA model

Taking into account the FHP model, the microdynamic of the system is defined by the *evolution operator* $\varepsilon$ splitted in two different steps: *streaming* (or propagation) $S$ and *collision* $C$.

$$\varepsilon = S \circ C \tag{4.4}$$

The *streaming* identifies the movement of the particles entering in the site $\vec{r}$ (position vector) at time $t$ with the direction $\vec{c}_i$ where $i$ represents one of the six lattice directions ($i = 1,2, \ldots ,6$ ).

$$S = n_i\left(\vec{r},t\right) \tag{3.5}$$

Only six possible velocities are available in function of $\Delta\vec{r}$ , lattice spacing, and $\Delta t$ , time step.

$$v_i = \frac{\Delta\vec{r}}{\Delta t}\vec{c}_i \tag{3.6}$$

Therefore, the propagation equation (3.7) with only the streaming step displays that a particle will continue in a straight line with the same direction of motion at the next time step.

$$n_i\left(\vec{r} + \Delta\vec{r}\cdot\vec{c_i},t + \Delta t\right) = n_i\left(\vec{r},t\right) \tag{3.7}$$

where $\vec{r} + \Delta\vec{r}\cdot c_i$ is the next particle site at his nearest neighbor at $t + \Delta t$ time step.

Collisions take place synchronously at every node and transform an initial state of a node *s* to a final state *s′* according to local rules (Fig. 3.2). It is a local step in fact only particles of a single node are involved.

Due to *collision*, the particle can change the original direction with a new $\vec{c}_i$ .

The collision term $\Omega_i\left(n_i\left(\vec{r},t\right)\right)$ is summed to the occupation number in order to obtain the LGCA equation (3.8).

$$n_i\left(\vec{r}+\Delta\vec{r}\cdot\vec{c_i},t+\Delta t\right)=n_i\left(\vec{r},t\right)+\Omega_i\left(n_i\left(\vec{r},t\right)\right) \tag{4.8}$$

### 3.5 Advantages and disadvantages

The FHP model is the advance step after the HPP model. Advantages are hold in the geometry construction and in the dynamic of the model. One of the main assets is due to the Boolean states giving the exact computing without any round-off error. It is also possible to improve the simulation with parallel computing obtaining faster results.

The main problem of this approach is the statistical noise as random fluctuations that disappear in the continuum limit. That is the general problem when it is tried to recover macroscopic quantities from microscopic simulations. Another disadvantage is due to the complexity to expand the model in three-dimensional problems keeping the sufficiently symmetric grid. The limited low-Mach number doesn't let to simulate such models with high velocity required.

# Chapter 4

# Bolzmann Gas Concepts

The main idea of Boltzmann Gas Concepts is characterized by the behavior of an ideal gas composed on a set of randomly moving and interacting particles.

Ludwig Boltzmann, in the eighteen seventies, played important rules with a new idea based on the description of ideal gases through classical mechanics. Basically he thought that a statistical treatment of ideal gases is necessary and appropriate in order to reproduce the behavior of the real flow.

Statistical mechanics is a branch of physics that applies probability theory using mathematical tools on a large complex systems. One of its most successful applications is the kinetic theory of gases.

## 4.1 Kinetic Theory of Gases

Kinetic Theory of Gases is the simplest approach that studies the macroscopic properties of a large number of particles in terms of molecular motion [15].

In the eighteen fifty-nines, Maxwell realized that dealing with a huge amount of particles is useless and even difficult for the calculations. The new Maxwell's idea focalized the attention on the averaging of the number of particles contained in an ideal gas.

A typical volume of gas, for instance, contains $10^{23}$ particles; too many particles and collisions between them. It clearly appears to approach the collective behavior of molecules from a statistical point of view.

Firstly, basic assumptions are supposed in order to describe the dynamic of ideal gas, defined as a gas with low density and temperature:

- Identical particles are treated in a finite domain.
- Particles are modeled as finite spheres, all of them with unit diameter dimension.
- Collision between particles and walls of the container are dealt elastically.
- There is no long-range forces between particles.

The $N$ particles in a microstate system at time $t$ are described by using two quantities: positions $\vec{r}_i(t)$, and momentum $\vec{p}_i(t)$.

Calling $f\left(\vec{r},\vec{p},t\right)$ the *one-particle* distribution function, the probability to find $N$ number of particles with a given position and momentum in a volume $dr^3$ and $dp^3$ is shown by the double integral,

$$N = \iint f\left(\vec{r},\vec{p},t\right) dr^3\, dp^3 \tag{4.2}$$

According to the kinetic energy, equation (4.2) is described in function of velocity in the three different directions $x$, $y$ and $z$.

$$\iiint f\left(c_x\right) f\left(c_y\right) f\left(c_z\right) dc_x dc_y dc_z = 1 \tag{4.3}$$

Where the function $f\left(c_i\right)$ is the fraction of the particles having velocity within the interval $\left\{c_i, c_i + dc_i\right\}$.

In order to find out the distribution function, one of the standard form for the distribution of an amount of energy between identical particles is described as,

$$f\left(c_x\right) = Ae^{-Bc_x^2} \tag{4.4}$$

Where A and B are two constants.

The product of the three distribution functions in $x$, $y$ and $z$ direction gives the probability to find particles with a speed between $c$ and $c + dc$ in a three dimensional space $\left(c_x, c_y, c_z\right)$.

$$f\left(c\right) = Ae^{-Bc_x^2} Ae^{-Bc_y^2} Ae^{-Bc_z^2} = A^3 e^{-Bc^2} \tag{4.5}$$

Therefore, the particles with that speed lyes between two shells with radius $c$ and $c + dc$. The volume of the spherical shell is defined as,

$$V = 4\pi c^2 dc \tag{4.6}$$

Thus, the probability density function is given by combination between the equations (4.5) and (4.6).

$$f\left(c\right)dc = 4\pi c^2 A^3 e^{-Bc^2} dc \tag{4.7}$$

The mass is the same for all the particles, thus the kinetic energy is in function only on the speed. For that reason we are able to define the constants A and B, calculated integrating the distribution function over all possible speeds as,

$$\frac{1}{2}mc^2 = \frac{\int_0^\infty \frac{1}{2}mc^2 f(c)dc}{\int_0^\infty f(c)dc} \tag{4.8}$$

Where $\int_0^\infty \frac{1}{2}mc^2 f(c)dc$ is the total energy of the system and $\int_0^\infty f(c)dc$ the total energy of the particles.

Substituting the value of $f(c)$ in (4.8), yields,

$$\frac{1}{2}mc^2 = \frac{3m}{4B} \tag{4.9}$$

But kinetic energy is equal to energy in terms of temperature of the gas,

$$\frac{1}{2}mc^2 = \frac{3}{2}kT \tag{4.10}$$

It is easy to find out the constant B substituting equation (4.10) in (4.9).

$$B = \frac{m}{3kT} \tag{4.11}$$

The constant A is determined by finding the proportionality between $f(c)$ and $c^2 e^{-\frac{mc^2}{2kT}}$.

Thus, the result of the distribution of molecular velocities in a gas is given by the equation (4.12),

$$f(c) = 4\pi\left(\frac{m}{2\pi kT}\right)^{\frac{3}{2}} c^2 e^{-\frac{mc^2}{2kT}} \tag{4.12}$$

The function increases parabolically from 0 (low speed) to a maximum value and then decreases exponentially. The position of the maximum depends on the temperature. The area below the curve is always equal to one. Roughly, the Maxwell-Boltzmann distribution function (4.12) shows how the speed of a moving particles changes at a particular temperature.

The Maxwell-Boltzmann distribution function is considered as the cornerstone of the kinetic theory. From this starting point, Ludwig Boltzmann was able to derive the evolution of $f$ in terms of microdynamic interactions.

## 4.2 The Boltzmann Equation

The Boltzmann Equation, as outlined in the previous chapter, describes the evolution of a single-particle distribution function. The statistical description of the number of particles (4.2) with a given position and velocity does not change if an external force acts on the gas. The force $F$ modifies the position $x$ to $x + c\mathrm{d}t$ and the velocity $c$ to $c + F\mathrm{d}t$.

$$f(r + c\mathrm{d}t, c + F\mathrm{d}t, t + \mathrm{d}t)\mathrm{d}r\mathrm{d}c - f(r, c, t)\mathrm{d}r\mathrm{d}c = 0 \tag{4.13}$$

If collisions between particles take place, the rate of change (4.13) is subjected to modify its status.

$$f(r + c\mathrm{d}t, c + F\mathrm{d}t, t + \mathrm{d}t)\mathrm{d}r\mathrm{d}c - f(r, c, t)\mathrm{d}r\mathrm{d}c = \Omega(f)\mathrm{d}r\mathrm{d}c\mathrm{d}t \tag{4.14}$$

Dividing the equation (4.14) by $\mathrm{d}r\mathrm{d}c\mathrm{d}t$,

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \Omega(f) \tag{4.15}$$

Where $\mathrm{d}f/\mathrm{dt}$ is equal to,

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial r}\frac{\mathrm{d}r}{\mathrm{d}t} + \frac{\partial f}{\partial c}\frac{\mathrm{d}c}{\mathrm{d}t} + \frac{\partial f}{\partial t}\frac{\mathrm{d}t}{\mathrm{d}t} = \frac{\partial f}{\partial r}c + \frac{\partial f}{\partial c}a + \frac{\partial f}{\partial t} \tag{4.16}$$

The $a$ term is equal to the acceleration and, related with force $F$ by the Newton's Second Law, $a = F/m$ ; thus, the Boltzmann Transport Equation is written as,

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial r}c + \frac{F}{m}\frac{\partial f}{\partial c} = \Omega \tag{4.17}$$

or

$$\frac{\partial f}{\partial t} + c \cdot \nabla f = \Omega \tag{4.18}$$

Equation (4.18) is the integro-differential equation from which is easy to draw out the macroscopic quantities, such as,

$$\rho(r,t) = m\int f(r,c,t)\,\mathrm{d}c \tag{4.19}$$

$$\rho(r,t)u(r,t) = m\int cf(r,c,t)\,\mathrm{d}c \tag{4.20}$$

$$\rho(r,t)e(r,t) = \frac{1}{2}m\int u_a^2 f(r,c,t)\,\mathrm{d}c \tag{4.21}$$

Where $m$ is the mass of a particle, $\rho$ the fluid density, $u_a$ the macroscopic speed flow defined as $u_a = c - u$ and $\rho e$ the energy density. All those equations (4.19, 4.20 and 4.21) are conservative mass, momentum and energy.

### 4.3 The Collision Operator: BGK Approximation

The collision operator is the main model in statistical physics for describing the iteration between colliding particles. It is a complicated integral and a simplification is needed in order to facilitate numerical and analytical solutions.

Bhatnagar, Gross and Krook (BGK) in the nineteen fifty-four, and at the same period Welander [16], proposed a simplified linear model for Collision Operator in which a simple way to ensure a correct relaxation is to image that each collision changes the distribution function $f$ by an amount proportional to the difference between $f$ and a local Maxwellian $f^{eq}$.

$$\Omega = \omega\left(f^{eq} - f\right) = \frac{1}{\tau}\left(f^{eq} - f\right) \tag{4.22}$$

The coefficient $\omega$ is called collision frequency; the time-scale relaxation factor $\tau$ describes the rate at which particles relax back to their equilibrium position; when $\tau > 1,$ it is called subrelaxation as the particle distribution is not completely relaxed to equilibrium. Indeed, overrelaxation when $\tau < 1,$ since the particle distribution is moved beyond equilibrium. On the other hand $\tau$ cannot be lower than 0.5 otherwise numerical instabilities occur. The term $f^{eq}$ represents the local Maxwellian equilibrium distribution function given by local conserved quantities such as, density, speed and temperature (4.12).

Since collision operator must preserve both mass and momentum, the formulas are,

$$\sum_i \Omega_i = 0 \tag{4.23}$$

$$\sum_i \vec{c}_i \cdot \Omega_i = 0 \tag{4.24}$$

Substituting the equation (4.22) in (4.18), the Boltzmann equation becomes a non-linear differential equation:

$$\underbrace{\frac{\partial f}{\partial t} + c \cdot \nabla f}_{Streaming} = \underbrace{\frac{1}{\tau}\left(f^{eq} - f\right)}_{Collision} \tag{4.25}$$

The discretization of the (4.23) in time and space along specific directions can be written as:

$$f_i\left(r + c_i \mathrm{d}t, c_i + F\mathrm{d}t, t + \mathrm{d}t\right) = f_i\left(r, c_i, t\right) + \frac{\mathrm{d}t}{\tau}\left[f_i^{eq}\left(r, c, t\right) - f_i\left(r, c, t\right)\right] \tag{4.26}$$

This is the starting point of the Lattice Boltzmann Method that replaces the Navier-Stokes equation in fluid dynamic simulations. In order to solve the partial differential equation, the domain needs to be divided into lattices where the particles as distribution functions move along specific directions (*streaming*) and collide each other (*collision*).

## 4.4 Chapman-Enskog Expansion

Chapman-Enskog Expansion is a useful tool available for multi-scale analysis. This procedure is based on a double expansion in the dependent and independent space-time $(x,t)$ variables.

Considering the single-particle distribution function, the formal expansion can be written as,

$$f_i = \sum_{n=0}^{\infty} \varepsilon^n f_i^{(n)} \tag{4.27}$$

The former can be simplify as:

$$f = f^{(0)} + \varepsilon f^{(1)} \tag{4.28}$$

Where $f_i^{(0)} = f_i^{eq}$ is the equilibrium distribution function and $f^1$ departure from this local equilibrium; $\varepsilon$ is a parameter proportional to *Knudsen number*[2] $K_n$; if $K_n \ll 1$, $f$ may be approximated with only few degrees of freedom.

The basic idea of multi-scale analysis is to represent space and time variables in terms of scale.
Since the final aim of the LBM is the macroscopic hydrodynamics, only the first term represents the equilibrium component and the others is hold in a final term called $f^{neq}$ (non-equilibrium).

$$f_i = f_i^{eq} + f_i^{neq} \tag{4.29}$$

---

[2] *Knudsen number* is a dimensionless number defined as the ratio between the mean free path of moleculae (lattice in LBM analysis) and the characteristic lenght scale .

The non-equilibrium components have the order $o(k)$ with respect to the equilibrium one and $k_n$ describes the Knudsen number of the flow.

Similar expansion for time and space derivates assumes the following forms,

$$\partial_t = \varepsilon \partial_{t_1} + \varepsilon^2 \partial_{t_2} \tag{4.30}$$

$$\frac{\partial}{\partial t} = \sum_{n=1}^{\infty} \varepsilon^n \frac{\partial}{\partial t^{(n)}} \tag{4.31}$$

And,

$$\partial_r = \varepsilon \partial_{r_1} \tag{4.32}$$

$$\frac{\partial}{\partial r} = \sum_{n=1}^{\infty} \varepsilon^n \frac{\partial}{\partial r_i^{(n)}} \tag{4.33}$$

Referring to equation (3.4), streaming operator need to be proportional to Chapman-Enskog second-order space representation.

$$S_t \sim \varepsilon \partial_{t_1} + \varepsilon^2 \partial_{t_2} + \varepsilon v_a \partial_{r_{1a}} + \frac{1}{2} \varepsilon^2 v_a v_b \partial_{r_{1a}} \partial_{r_{1b}} \tag{4.34}$$

Similar for the collision operator,

$$C[f] \sim C\left[f^0\right] + \varepsilon C' f^1 \tag{4.35}$$

The distribution functions $f_i^{(n)}$

depend only on time via local density, macroscopic velocity and internal energy.

Taking into account only the fist order terms in the expansion, the Boltzmann Equation become,

$$\frac{\partial}{\partial t}\left(f_i^{(0)}+\varepsilon f_i^{(1)}+...\right)+c\frac{\partial}{\partial r}\left(f_i^{(0)}+\varepsilon f_i^{(1)}+...\right)+\frac{F}{m}\frac{\partial}{\partial c}\left(f_i^{(0)}+\varepsilon f_i^{(1)}+...\right)=-\frac{1}{\tau}\sum_{n=0}^{\infty}\varepsilon^n f_i^{(n)}$$

(4.36)

Thanks to Chapman-Enskog tool analysis, we are able to define a numerical code in order to extract macroscopic quantities. On the other hand it is worth mentioning that the Chapman-Enskog perturbative treatment of the Boltzmann equation is generally not convergent exposed to severe numerical instability.

# Chapter 5

# Lattice Boltzmann Method

The Lattice Boltzmann Method (LBM) approximates the continuous Boltzmann equation to a discrete physical space with lattice nodes and a velocity space by a set of microscopic speed vectors [17].

## 5.1 Models

For the simulation of three-dimensional flow we will use the so called D3Q27 square lattice model with twenty-seven discrete velocities. The common terminology referred to the lattice geometry is D*n*Q*m*, where *n* represents the possible dimension of the model and *m* refers to the number of speeds.

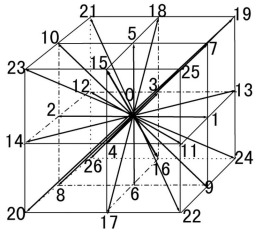


Fig. 5.1 D3Q27 square lattice model.

Each node has twenty-six neighbors connected by links. The distribution functions associated with the particles move move along these links in unit time step.

## 5.2 Equilibrium Distribution Function

Taking into account the particles moving in a medium with macroscopic velocity $u$, the Maxwellian Distribution Function (4.12) is written as,

$$f = \frac{\rho}{2\pi/3} e^{-\frac{3}{2}\left(\vec{c}-\vec{u}\right)^2} = \frac{\rho}{2\pi/3} e^{-\frac{3}{2}c^2} e^{3\left(\vec{c}\cdot\vec{u}-u^2\right)} \tag{5.1}$$

Using the Taylor Series expansion for $e^{-x}$ and ending up at the third order,

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} \tag{5.2}$$

Hence, equation (5.1) is expanded around the stationary state as,

$$f = \frac{\rho}{2\pi/3} e^{-\frac{3}{2}c^2} \left[ 1 + 3\left(\vec{c}\cdot\vec{u}\right) - \frac{3}{2}u^2 + \frac{9}{2}\left(\vec{c}\cdot\vec{u}\right)^2 \right] \tag{5.3}$$

Studying the general form of the equilibrium distribution function, yields,

$$f_i^{eq} = \rho \cdot \omega_i \left[ A + Bc_i \cdot u + C\left(c_i \cdot u\right)^2 + Du^2 \right] \tag{5.4}$$

The constants A,B,C, and D can be determined basing on the mass, momentum and energy conservation principles.

The density, scalar parameter, is equal to the sum of all the distribution functions used in the geometry model as,

$$\rho = \sum_{i=0}^{26} f_i^{eq} \tag{5.5}$$

If, for instance, the flow is stagnant, the velocity will be equal to zero and then the Equilibrium Distribution Function will be equal to the first Taylor Expansion term, A.

$$f_i^{eq} = \rho A \omega_i \tag{5.6}$$

## 5.3 Governing Equations

Lets focus our attention on the one-dimensional diffusion equation, such as,

$$\frac{\partial \phi}{\partial t} = \alpha \frac{\partial \phi^2}{\partial x^2} \tag{5.7}$$

Where the variable $\phi$ may be temperature or momentum or density and the $\alpha$ parameter stands for thermal or mass diffusion coefficient or kinematic viscosity for mass and momentum diffusion.

Following with finite difference approximation for one-dimensional diffusion problem, the domain needs to be discretized in square grids. Approximating the diffusion equation at a node *i*, we obtain,

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \alpha \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta x^2} \tag{5.8}$$

Rearranging the above equation,

$$T_i^{n+1} = T_i^n + \frac{\alpha \Delta t}{\Delta x^2}\left(T_{i+1}^n - 2T_i^n + T_{i-1}^n\right) \tag{5.9}$$

And reformulating as,

$$T_i^{n+1} = T_i^n \left( 1 - \frac{2\alpha\Delta t}{\Delta x^2} \right) + \frac{2\alpha\Delta t}{\Delta x^2} \left( \frac{T_{i+1}^n + T_{i-1}^n}{2} \right) \tag{5.10}$$

Substituting $\omega$ into equation (5.10),

$$\omega = \frac{2\alpha\Delta t}{\Delta x^2} \tag{5.11}$$

$$T_i^{n+1} = T_i^n \left(1 - \omega\right) + \omega \left( \frac{T_{i+1}^n}{2} + \frac{T_{i-1}^n}{2} \right) \tag{5.12}$$

For stability conditions $\Delta t \leq \frac{\Delta x^2}{2\alpha}$ , hence the term $\left(1 - \omega\right)$ must be greater or equal to zero.

Examining equation (5.12), the last term $\frac{1}{2}\left(T_{i+1}^n + T_{i-1}^n\right)$ is the equilibrium value simplified by $T_i^{eq}$ calculated by averaging the temperature around $T_i$ .

$$T_i^{n+1} = T_i^n \left(1 - \omega\right) + \omega T_i^{eq} \tag{5.13}$$

Moving on the Lattice Distribution Function for D3Q27 model and neglecting the force acting on the particles, we yield,

$$\frac{\partial f_i(r,t)}{\partial t} + c_i \frac{\partial f_i(r,t)}{\partial x} = \Omega_i \tag{5.14}$$

Where $i = 0,1,2,...,26$ defines the twenty-seven possible speeds. The term $c_i$ is in function of the lattice length and the time step.

$$c_i = \frac{\Delta r}{\Delta t} \tag{5.15}$$

Taking into account the BGK approximation for the rate of change of distribution function $\Omega_i$, the equation (5.14) becomes,

$$\frac{\partial f_i(r,t)}{\partial t} + c_i \frac{\partial f_i(r,t)}{\partial t} = -\frac{1}{\tau}\left[ f_i(r,t) - f_i^{eq}(r,t) \right] \tag{5.16}$$

Applying the time discretization integrating over a time step, the equation becomes as,

$$\frac{\partial f_i}{\partial t} \cong \frac{f_i(r,t+\Delta t) - f_i(r,t)}{\Delta t} \tag{5.17}$$

And the space step,

$$\frac{\partial f_i}{\partial x} \cong \frac{f_i(r+\Delta r,t+\Delta t) - f_i(r,t+\Delta t)}{\Delta x} \tag{5.18}$$

Substituting equation (5.17) and (5.18) in the Boltzmann Equation, the time and space discrete formula is written as,

$$f_i(r+\Delta r,t+\Delta t) - f_i(r,t) = -\frac{\Delta t}{\tau}\left[ f_i(r,t) - f_i^{eq}(r,t) \right] \tag{5.19}$$

Reformulated the (5.17) in order to find the similarity with equation (5.13), yields,

$$f_i\left(r+\Delta r,t+\Delta t\right)=f_i\left(r,t\right)\left[1-\omega\right]+\omega f_i^{eq}\left(r,t\right) \tag{5.20}$$

Where the relaxation time $\omega$ is written as,

$$\omega=\frac{\Delta t}{\tau} \tag{5.21}$$

## 5.4 D3Q27 Model

For the three-dimensional flow, D3Q27 model represents a good way in order to describe the fluid in the near equilibrium state of low-Mach number hydrodynamic.

The discrete set of velocities $e_i$ is defined as a vector with the following values for each node,

$$e_i=\begin{cases}(0,0,0), & \rightarrow\ i=0\\ (\pm1,0,0)c,(0,\pm1,0)c,(0,0,\pm1)c, & \rightarrow\ i=1,2,3,4,17,18\\ (\pm1,\pm1,0)c,(0,\pm1,\pm1)c,(\pm1,0,\pm1)c, & \rightarrow\ i=5,6,7,8,9,10,11,12,13,14,15,16,17,18\\ (\pm1,\pm1,\pm1)c, & \rightarrow\ i=19,20,21,22,23,24,25,26\end{cases} \tag{5.22}$$

```
!        Nodes positions
    v(0:26,0) = (/0,1,0,-1, 0,1,-1,-1, 1,1,0,-1, 0, 1, 0,-1, 0,0,     &
                & 0,  1, -1, -1, 1, 1, -1, -1, 1/)
    v(0:26,1) = (/0,0,1, 0,-1,1, 1,-1,-1,0,1, 0,-1, 0, 1, 0,-1,0,     &
                & 0,  1,  1, -1,-1, 1,  1, -1,-1/)
    v(0:26,2) = (/0,0,0, 0, 0,0, 0, 0, 0,1,1, 1, 1,-1,-1,-1,-1,1,     &
                & -1,  1,  1,  1, 1,-1, -1, -1,-1/)
```

Code 5.1 Position of the nodes.

Neglecting external forces, the transport equation for $f(r,c,t)$ is expressed by the Boltzmann equation as equation (5.16) where the collision term is approximated with BGK model. The equilibrium distribution function is given by,

$$f_i^{eq} = \rho\omega_i\left[1+\frac{3}{c^2}\left(e_i\cdot\vec{u}\right)+\frac{9}{2c^4}\left(e_i\cdot\vec{u}\right)^2-\frac{3}{2c^2}\left(\vec{u}\cdot\vec{u}\right)\right] \tag{5.23}$$

Where $c$ is the discrete particle velocity as (5.15). In twenty-seven square lattice, the equilibrium function for each node is,

$$\begin{cases} f_i^{eq} = \rho\omega_i\left[1-\frac{3}{2}\left(\vec{u}\right)^2\right] & \rightarrow\ i=0 \\ f_i^{eq} = \rho\omega_i\left[1+3\left(e_i\cdot\vec{u}\right)+\frac{9}{2}\left(e_i\cdot\vec{u}\right)^2-\frac{3}{2}\left(\vec{u}\right)^2\right] & \rightarrow\ i=1,2,...,26 \end{cases} \tag{5.24}$$

The algorithm is as follows,

```
!         ========================================================
!         Compute equilibrium distribution
!         ========================================================
SUBROUTINE computeFeq(fEq,rho,u,uSqr)
    USE D3Q27COnst, ONLY: t, v
    USE simParam, ONLY: xDim, yDim, zDim
    implicit none

    double precision, INTENT(IN):: rho(zDim,yDim,xDim), &
         & uSqr(zDim,yDim,xDim), u(zDim,yDim,xDim,0:2)
    double precision, INTENT(INOUT):: fEq(zDim,yDim,xDim,0:26)
    integer:: i, x, y, z
    double precision:: uxyz
```

```
    do i = 0, 26
        do x = 1, xDim
            do y = 1, yDim
                do z = 1, zDim
                   uxyz = u(z,y,x,0) * v(i,0) + u(z,y,x,1) * v(i,1) + &
                          & u(z,y,x,2) * v(i,2)
                   fEq(z,y,x,i)=t(i)*rho(z,y,x)*(1.0d0+3.0d0* &
                          & uxyz+4.5d0*uxyz*uxyz-1.5d0*uSqr(z,y,x))
                end do
            end do
        end do
    end do
END SUBROUTINE computeFeq
```

Code 5.2 Equilibrium Distribution Function.

The weight factor $\omega_i$ is defined as a vector with the follow values for each node,

$$\begin{cases} \omega_i = 8/27 & \rightarrow i = 0 \\ \omega_i = 2/27 & \rightarrow i = 1,2,3,4,17,18 \\ \omega_i = 1/54 & \rightarrow i = 5,6,7,8,9,10,11,12,13,14,15,16,17,18 \\ \omega_i = 1/216 & \rightarrow i = 19,20,21,22,23,24,25,26 \end{cases} \tag{5.25}$$

```
!         D3Q27 Weights
    double precision,parameter:: t(0:26) =                          &
         &(/8.0d0/27.0d0,2.0d0/27.0d0,2.0d0/27.0d0,2.0d0/27.0d0     &
         &,2.0d0/27.0d0,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0      &
         &,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0      &
         &,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0      &
         &,1.0d0/54.0d0,2.0d0/27.0d0,2.0d0/27.0d0,1.0d0/216.0d0     &
         &,1.0d0/216.0d0,1.0d0/216.0d0,1.0d0/216.0d0,1.0d0/216.0d0 &
         &,1.0d0/216.0d0,1.0d0/216.0d0,1.0d0/216.0d0/)
```

Code 5.3 Weight for D3Q27 model.

# Chapter 6

# Boundary Conditions

Boundary Conditions (BCs) are an essential part of any numerical method. For LBM they represent a critical part due to the microscopic point of view [18]; BCs need to be translated from macroscopic hydrodynamic quantities to microscopic BCs for particle distribution.

BCs are classified in four categories:

1) Periodic Boundary Conditions, easiest but sometimes not realistic.
2) Inflow Boundary Conditions, a desired velocity or pressure (density) set the inlet distribution functions.
3) Outflow Boundary Conditions, very difficult to deal with and they usually involve a combination of macroscopic quantities with distributions functions.
4) Wall Boundary Conditions, which include slip and non-slip BCs, shows the iteraction between solid obstructions and fluid.

## 6.1 Periodic Boundary Conditions

Periodic BCs represents the simplest approach useful only in preliminary test as it implies hight symmetry of the flow domain [19].

$$f_i = f_{eq} \tag{6.1}$$

It is applied directly to the particles distribution functions requiring only to copying the distribution function from the opposite boundary.

```
do i = 0, 26
     do x = 1, xDim
         do y = 1, yDim
             do z = 1, zDim
                 f(z,y,x,i) = fEq(z,y,x,i)
             end do
         end do
     end do
 end do
```

Code 6.1 Periodic Boundary Conditions.

## 6.2 Velocity Boundary Conditions

When we model a flow with a prescribed velocities or pressure profiles, we need to specify macroscopic quantities such as velocity or pressure (density). Dealing with the pressure is equivalent to imposing density conditions thanks to the equations of state [20].

Considering a D3Q27 model, the velocities $u_x$ , $u_y$ and $u_z$ are determined studying the distribution functions.

$$\rho = \sum_{i=0}^{26} f_i \tag{6.2}$$

Knowing the density, the velocities are set equal to the difference between inlet and outlet nodes.

$$u_x = \frac{\left(f_1 + f_5 + f_8 + f_9 + f_{13} + f_{19} + f_{23} + f_{26} + f_{22} - \left(f_3 + f_6 + f_7 + f_{11} + f_{15} + f_{20} + f_{24} + f_{25} + f_{21}\right)\right)}{\rho} \tag{6.3}$$

$$u_y = \frac{\left(f_2 + f_5 + f_6 + f_{14} + f_{10} + f_{19} + f_{20} + f_{24} + f_{23} - \left(f_4 + f_7 + f_8 + f_{16} + f_{12} + f_{22} + f_{21} + f_{25} + f_{26}\right)\right)}{\rho}$$

(6.4)

$$u_z = \frac{\left(f_9 + f_{17} + f_{11} + f_{10} + f_{12} + f_{19} + f_{20} + f_{21} + f_{22} - \left(f_{13} + f_{18} + f_{15} + f_{14} + f_{16} + f_{23} + f_{24} + f_{25} + f_{26}\right)\right)}{\rho}$$

(6.5)

```
!         ============================================================
!         Compute velocity from distribution functions
!         ============================================================
SUBROUTINE computeMacros(image,f,rho,u,uSqr)
    USE simParam, ONLY: xDIm, yDim, zDim
    implicit none
    integer, INTENT(IN):: image(zDim,yDim,xDim)
    double precision, INTENT(IN):: f(zDim,yDim,xDim,0:26)
    double precision, INTENT(INOUT)::u(zDim,yDim,xDim,0:2),     &
           &   rho(zDim,yDim,xDim),uSqr(zDim,yDim,xDim)
    integer:: x,y,z

    do x = 1, xDim
        do y = 1, yDim
            do z = 1, zDim
              rho(z,y,x)  = (f(z,y,x,0) + f(z,y,x,1) + f(z,y,x,2)   +  &
           & f(z,y,x,3)  + f(z,y,x,4)  + f(z,y,x,5)  + f(z,y,x,6)  +  &
           & f(z,y,x,7)  + f(z,y,x,8)  + f(z,y,x,9)  + f(z,y,x,10) +  &
           & f(z,y,x,11) + f(z,y,x,12) + f(z,y,x,13) + f(z,y,x,14) +  &
           & f(z,y,x,15) + f(z,y,x,16) + f(z,y,x,17) + f(z,y,x,18) +  &
           & f(z,y,x,19) + f(z,y,x,20) + f(z,y,x,21) + f(z,y,x,22) +  &
           & f(z,y,x,23) + f(z,y,x,24) + f(z,y,x,25) + f(z,y,x,26))

              u(z,y,x,0) = ((f(z,y,x,1) - f(z,y,x,3) + f(z,y,x,5)   - &
            & f(z,y,x,6)  - f(z,y,x,7)  + f(z,y,x,8)  + f(z,y,x,9)  - &
            & f(z,y,x,11) + f(z,y,x,13) - f(z,y,x,15) + f(z,y,x,19) - &
            & f(z,y,x,20) + f(z,y,x,23) - f(z,y,x,24) + f(z,y,x,26) - &
            & f(z,y,x,25) + f(z,y,x,22) - f(z,y,x,21)) / rho(z,y,x))

              u(z,y,x,1) = ((f(z,y,x,2) - f(z,y,x,4) + f(z,y,x,5)   + &
           & f(z,y,x,6)  - f(z,y,x,7)  - f(z,y,x,8)  + f(z,y,x,14) - &
           & f(z,y,x,16) + f(z,y,x,10) - f(z,y,x,12) + f(z,y,x,19) - &
           & f(z,y,x,22) + f(z,y,x,20) - f(z,y,x,21) + f(z,y,x,24) - &
           & f(z,y,x,25) + f(z,y,x,23) - f(z,y,x,26)) / rho(z,y,x))
```

```
              u(z,y,x,2) = ((f(z,y,x,9) - f(z,y,x,13) + f(z,y,x,17) - &
            & f(z,y,x,18) + f(z,y,x,11) - f(z,y,x,15) + f(z,y,x,10) - &
            & f(z,y,x,14) + f(z,y,x,12) - f(z,y,x,16) + f(z,y,x,19) - &
            & f(z,y,x,23) + f(z,y,x,20) - f(z,y,x,24) + f(z,y,x,21) - &
            & f(z,y,x,25) + f(z,y,x,22) - f(z,y,x,26)) / rho(z,y,x))

                uSqr(z,y,x) = u(z,y,x,0) * u(z,y,x,0) + u(z,y,x,1) *  &
            & u(z,y,x,1) + u(z,y,x,2) * u(z,y,x,2)
            end do
        end do
    end do
END SUBROUTINE computeMacros
```

Code 6.2 Macroscopic variables from distribution functions.

## 6.3 Bounce-Back Boundary Conditions

Bounce-Back *no-slip* BCs is one of the most important benefits of the LBM. Basically, the distribution functions propagating towards the solid boundary are simply scattered back into the fluid domain along a direction rotated by 180° radiants.

If the boundary is placed on the solid boundary, bounce-back gives only first order accuracy [21]; second order accuracy is achieved when the wall is placed half-way from the lattice sites.

For instance, if the distribution functions $f_7$, $f_4$ and $f_8$ are known, after the streaming process boundary conditions is applied as,

$$\begin{cases} f_7 = f_5 \\ f_4 = f_2 \\ f_8 = f_6 \end{cases} \tag{6.6}$$

Bounce-Back BCs ensures mass and momentum conservation at the boundary. It may be apply to all the lattices on the solid surface in modeling flow over an obstacle.

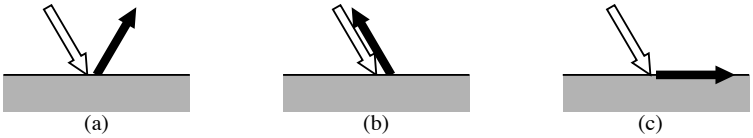


Fig. 6.1 Bounce-Back rules: (a) Specular reflection, (b) bounce-back condition and (c) trapping wall condition.

Bounce-Back may be *Slip* conditions, it means that the Discrete Functions are reflected to a mirror-like fashion instead of being bounce-back. Slip occurs when the fluid particles are specularly reflected and the tangential velocity is preserved; the wall would not exert tangential stresses on the fluid and slippage would be observed.

## 6.4 Zou-He Pressure and Velocity Boundary Conditions

Underlining a new method proposed by Zou Qisu and He Xiaoyi [22] using an idea of Bounce-Back of non-equilibrium distribution, we are able to obtain excellent results shown by a second-order accuracy.

Assumptions, such as steady flow independent of x, need in order to satisfy the differential equation which is a second-order approximation of the Navier-Stokes equations.

Taking into account the bottom node, the boundary is aligned in x-direction with $f_4, f_7, f_8$ pointing into the wall. Boundary conditions are calculated after streaming step, thus, $f_0, f_1, f_3, f_4, f_7, f_8$ are known. If $u_x$ and $u_y$ are specified on the wall, $f_2, f_5, f_6$ and $\rho$ need just to be determined from equations (6.2).

$$f_2 + f_5 + f_6 = \rho - \left(f_0 + f_1 + f_3 + f_4 + f_7 + f_8\right) \tag{6.7}$$

$$f_5 - f_6 = \rho u_x - \left(f_1 - f_3 - f_7 + f_8\right) \tag{6.8}$$

$$f_2 + f_5 + f_6 = \rho u_y + \left(f_4 + f_7 + f_8\right) \tag{6.9}$$

Substituting equation (6.7) in equation (6.9),

$$\rho = \frac{1}{1-u_y}\left[f_0 + f_1 + f_3 + 2\left(f_4 + f_7 + f\right)\right] \tag{6.10}$$

We assume that the bounce-back rule is still correct for the non-equilibrium part of the particle distribution, thus, $f_2 - f_2^{(eq)} = f_4 - f_4^{(eq)}$ . Knowing $f_4$, the distribution functions $f_2$, $f_5$, $f_6$ are figure out as,

$$f_2 = f_4 + \frac{2}{3}\rho u_y \tag{6.11}$$

$$f_5 = f_7 - \frac{1}{2}\left(f_1 - f_3\right) + \frac{1}{2}\rho u_x + \frac{1}{6}\rho u_y \tag{6.12}$$

$$f_6 = f_8 - \frac{1}{2}\left(f_1 - f_3\right) + \frac{1}{2}\rho u_x + \frac{1}{6}\rho u_y \tag{6.13}$$

Taking into account pressure Boundary Condition and supposing a flow boundary along y-direction, the pressure at the inlet and the velocity along y-direction is known, it is clear how to find out $f_1, f_5, f_8$ ,

$$f_1 + f_5 + f_8 = \rho_{in} - \left(f_0 + f_2 + f_3 + f_4 + f_6 + f_7\right) \tag{6.14}$$

$$f_1 + f_5 + f_8 = \rho_{in} u_x + \left(f_3 + f_6 + f_7\right) \tag{6.15}$$

$$f_5 - f_8 = f_2 - f_4 + f_6 - f_7 \tag{6.16}$$

Substituting equation (6.14) in (6.16),

$$u_x = 1 - \frac{\left[ f_0 - f_2 + f_4 + 2\left( f_3 + f_6 + f_7 \right) \right]}{\rho_{in}} \tag{6.17}$$

Using the bounce-back rule for the non-equilibrium part normal to the inlet ($f_1 - f_1^{(eq)} = f_3 - f_3^{(eq)}$), the unknown distribution functions are calculated as,

$$f_1 = f_3 + \frac{2}{3}\rho_{in} u_x \tag{6.18}$$

$$f_5 = f_7 - \frac{1}{2}\left( f_2 - f_4 \right) + \frac{1}{6}\rho_{in} u_x \tag{6.19}$$

$$f_8 = f_6 - \frac{1}{2}\left( f_2 - f_4 \right) + \frac{1}{6}\rho_{in} u_x \tag{6.20}$$

A special treatment must be considered for the corner node. After the streaming step, we find out that,

$$f_1 = f_3 + \left( f_1^{(eq)} - f_3^{(eq)} \right) = f_3 \tag{6.21}$$

$$f_2 = f_4 + \left( f_2^{(eq)} - f_4^{(eq)} \right) = f_4 \tag{6.22}$$

$$f_5 = f_7 \tag{6.23}$$

$$f_6 = f_8 = \frac{1}{2}\left[ \rho_{in} - \left( f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_7 \right) \right] \tag{6.24}$$

Similar procedure is applied for the top inlet node and outlet corner nodes.

# Chapter 7

# Numerical Results

A brief explanation about numerical analysis is described in order to compared numerical solution using two different method: LBM and Finite Volume analysis. Incompressible, isothermal and unsteady fluid flow will be taken into account.

## 7.1 LBM Analysis

LBM serves as exceptional numerical laboratories for a wide number of processes in different fields []. The common idea with LBM is to approximate and solve the Boltzmann equation from the particle perspective and focus on the macroscopic quantities.

Here a discussion of the main characteristic of the 3D code is presented in order to clarify the practical use of the equation in computer language.

### 7.1.1 Flow Geometry

The model used is D3Q27 in three-dimension square lattice in a computational parallelepiped domain.

The dimension of the model is in function on the total length of the microchannel; the ratio between the length of the domain and the length of the mesh must be integer number.

## 7.2 The code

An overview of LBM algorithm is shown in figure 7.1. The code is divided in three steps: initialization, collision and streaming phase.

Initialization phase defines the macroscopic quantities such as velocity, density, Reynolds number, geometry domain and number of iterations we want to run for collision and streaming phase. Poiseuille profile is set up as BCs for the velocity at the inlet of our domain.

Collision phase is based on four sub steps; first of all we compute the inlet and Bounce-Back BCs. Then, we find out the macroscopic velocity through the particle distribution functions. After calculating the equilibrium distribution function, we are able to compute the collision operator.

Streaming phase is the last of the three steps: each population function is shifted one site along their corresponding lattice direction. Important to underline that the code does not need any temporary memory for this phase.

The result to analyze is the macroscopic velocity of fluid flow. The components x,y and z are printed out to a text file in two format: Tecplot and Matlab. For each mesh point we obtain three different velocities: $v_x$, $v_y$ and $v_z$.

```
!         ========================================================
!         Constants for simulation setup
!         ========================================================
MODULE simParam

!        Dimension of the domain
    integer, parameter:: xDim   = 250
    integer, parameter:: yDim   = 50
    integer, parameter:: zDim   = 50
    integer, parameter:: obstX  = xDim/5
    integer, parameter:: obstY  = yDim/2
    integer, parameter:: obstR  = yDim/10+1

!        Adimensional Mesh size:
    double precision, parameter:: deltaT = 0.25d0

!        Increment numbers
    double precision, parameter:: deltaX = xDim / deltaT
    double precision, parameter:: deltaY = yDim / deltaT
    double precision, parameter:: deltaZ = zDim / deltaT

!        Number of steps
    integer, parameter:: tMax = 500

!        Variables defined
    double precision, parameter:: uMax = 0.02d0
    double precision, parameter:: Re = 10.0d0
END MODULE simParam
```

Code 7.3 Domain variables

In order to reach the right accuracy the number of iterations might be around 500. This is due to the fact that for small lattice size, the time of CPU is usually low, and therefore, the time reduction is not very sensible. In contrast, if the lattice has a large size, the time reduction is sensible.

```
!         ========================================================
!         Construct an array the defines the flow geometry
!         ========================================================
SUBROUTINE constructImage(image)
    USE cellConst
    USE simParam, ONLY: xDim, yDim, zDim, obstX, obstY, obstR
    USE D3Q27Const, ONLY: v

    implicit none

    integer, INTENT(INOUT):: image(zDim,yDim,xDim)
    integer:: x,y,z

!        Definition of the Boundary
    image(:,:,:)     = fluid
    image(:,:,1)     = inlet
    image(:,:,xDim)  = outlet
    image(:,1,:)     = wall
    image(:,yDim,:)  = wall
    image(1,:,:)     = wall
    image(zDim,:,:)  = wall


!        Circular obstacle

    do x = 1, xDim
        do y = 1, yDim
            do z = 1, zDim
                if (((x-obstX)**2 + (y-obstY)**2) <= (obstR**2) ) &
                   &   image(z,y,x) = wall
            end do
        end do
    end do

END SUBROUTINE constructImage
```

Code 7.4 Flow geometry variables

Open and solid-wall BCs are implemented: Bounce-Back BCs is specified on the walls and Poiseuille flow at the inlet. At macroscopic level of fluid flow, velocity and Reynolds number is defined a priori.

## 7.3 Flow Simulation

For presenting an overall view of the flow simulation in the microchannel, the contours of velocity field have been shown in figure 7.1.

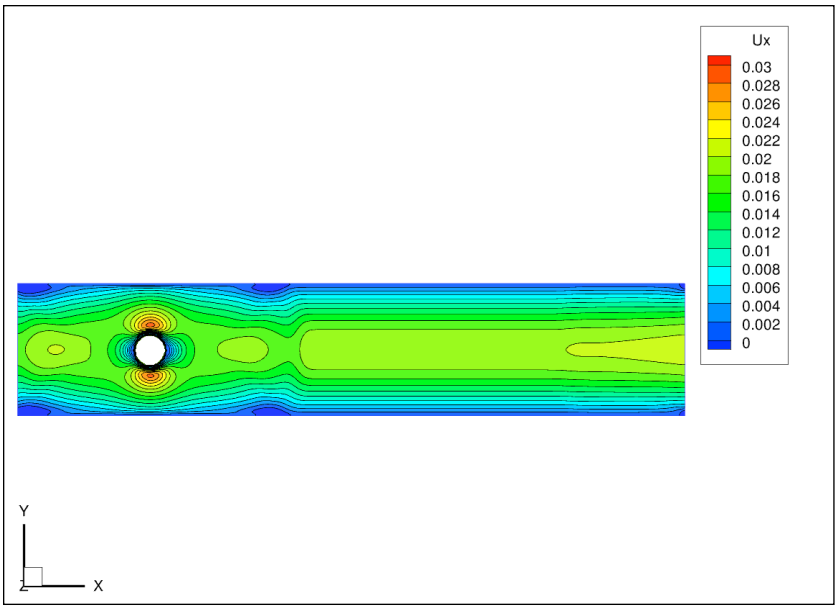


Figure 7.1 Velocity contour $u_x$ along x-axis.

Figure 7.1 show just the results in a middle plane of the model. The data obtained reflects the results we were supposed to get; an increasing of the velocity up to the double on the lateral surface of the obstacle and the stagnation point on the contact between fluid and obstacle.

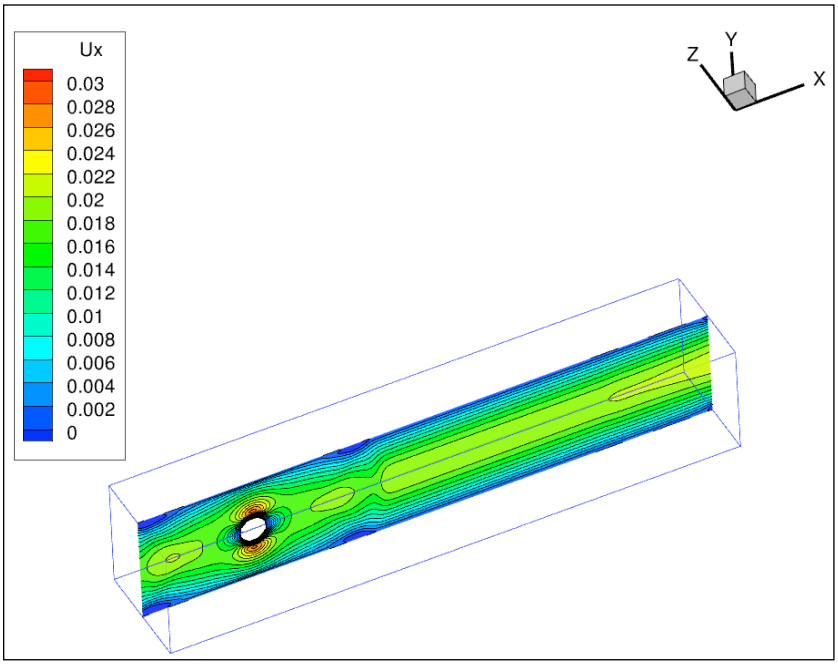


Fig. 7.2 3D view of the velocity contour $u_x$ along x-axis.

The velocity contour for $u_x$ shows the results of the LBM analysis along x-direction at z equal to 25 millimeters; at the inlet the Poiseuille profile is defined with a velocity range between 0 m/s on the borders and a maximum value given a priori. Close to the obstacle, a stagnation point has the right velocity equal to 0 m/s and on the surfaces at 90° with respected to the stagnation point the velocity has the maximum value equal to 0.03 m/s. Finally, at the output a parabolic velocity profile is outlined as aspected.

The velocity at the walls is shown in figure 7.3. The velocity is not equal to zero but tends to zero.

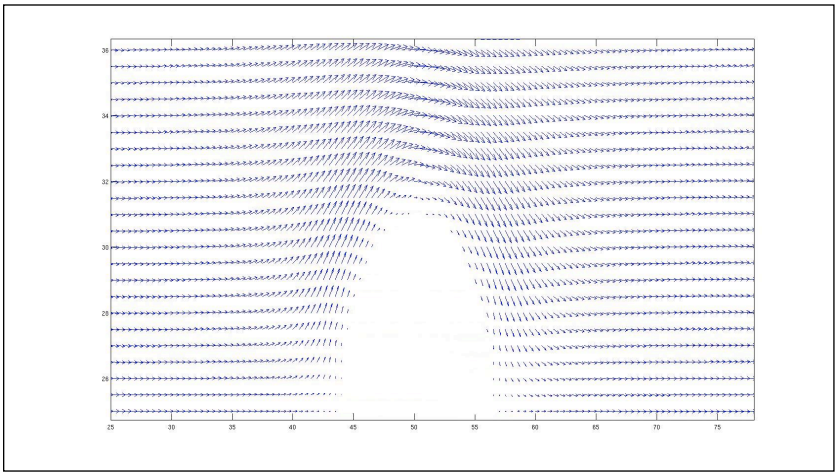

Fig. 7.3 Zoom in view of the stream vectors near the obstacle.

This is due to the limit of the 3D LBM mesh geometry; the center point of the mesh is positioned on the border of the domain, then, half of the cube points out of the domain and the other, respectively, inside.

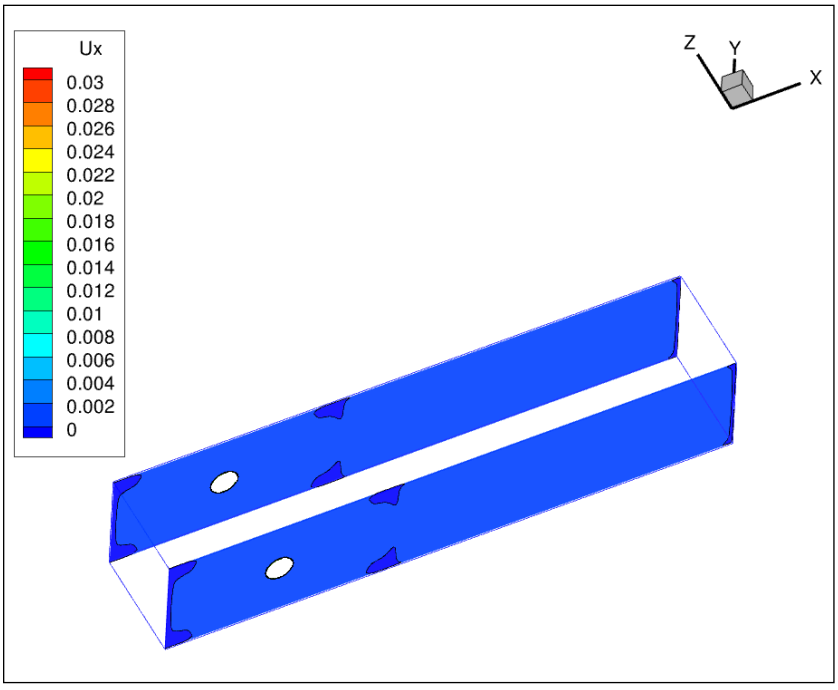


Fig. 7.3 3D view for the velocity contour $u_x$ on the borders.

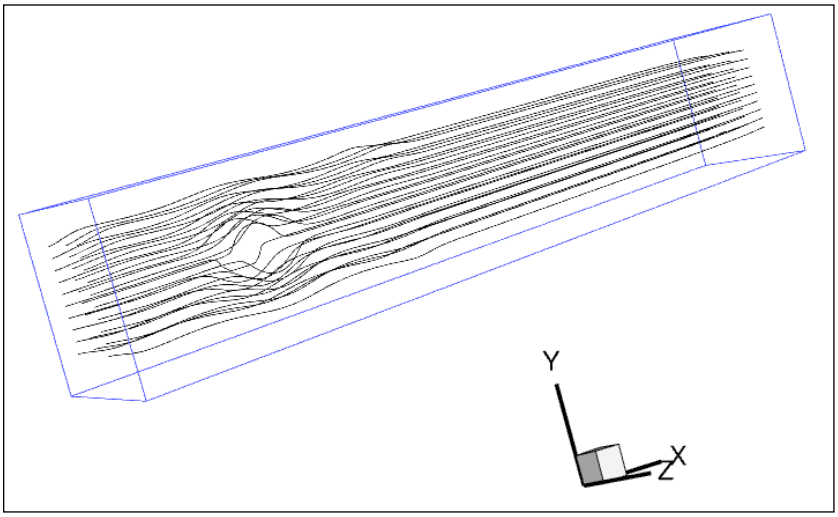


Fig. 7.4 Streamlines.

## 7.4 Finite Volume Results

A traditional Navier-Stokes solver will be used in order to compare the results obtained with LBM approach. We will shown that LBM results are satisfactory for the investigated case.

### 7.4.1 Ansys Preprocessing

The general-purpose commercial CFD/CAE package Ansys [23] is chosen in order to provide a basis of comparison for LBM results. Ansys is a Finite Volume Navier-Stokes solver and is usually utilized for the study of wide simulations of fluid flows . But not all of them are adequate for describing the flow; such as microscopic flows, with Knudsen number not infinitesimal, are described by particle-particle and particle-wall interactions. Knudsen number cannot be considered as small as for the propagation of ultrasound due to the high frequency of the sound wave; even shock waves, when a flow changes from supersonic to subsonic, including drastic changes in temperature, pressure, and density, cannot be sufficiently studied with NS

equations. Basically, Knudsen number is a appropriate variable in order to understand if a specific case may be modeled with NS or LBM approach. If the flow regimens have $k_n$ greater than 0.1, the Navier-Stokes equation fail; in that case, the gas must be described in greater detail, or by extended macroscopic models.

The model taken into account in the analysis is valid for both approaches, NS and LBM. For the present study, three-dimensional incompressible, unsteady flow is used in the simulation with Ansys software. The computational domain is discretized with cubic cells where an unstructured mesh grids with higher density is used close to the circular cylinder obstacle.

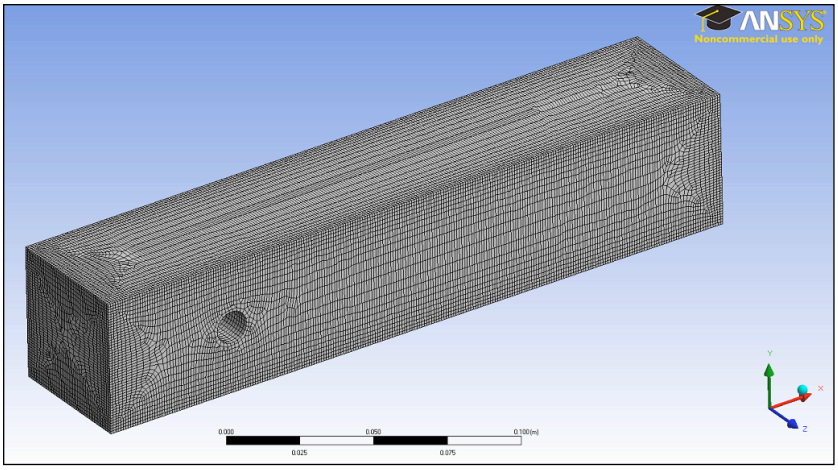


Fig. 7.5 Discrete nonuniform lattice mesh grids

The BCs taken into account are the same as LBM approach: no-slip, inlet and outlet BCs. The walls of the domain are classified as no-slip bounce-back BCs. At the inlet, constant velocity profile is set up with a velocity equal to 0.02 m/s normal to x-direction. Indeed, an average static pressure is defined at the outlet.

The fluid considered is isothermal air at 25°; the number of iterations are 1000 with a timescale factor equal to 1.0. The criteria of convergence is limited as $10^{-6}$ residual target.

### 7.4.2 Ansys Results

The reference solution was computed using a fine mesh close to the obstacle. It should be noted that the CPU time could probably be reduced by choosing an optimized mesh.

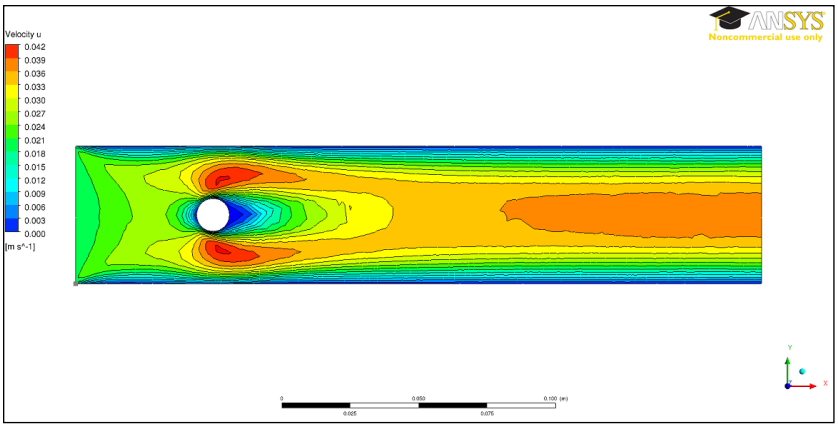


Fig. 7.6 Velocity profile $u_x$ using CFX compiler.

Figure 7.6 shows the velocity profile $u_x$ along the center plane of the channel; the maximum velocity close to the obstacle is equal to 0.04 m/s. At the point of impact between obstacle and flow, we are able to see the stagnation point with velocity equal to zero. After the obstacle the flow try to become steady with a maximum velocity equal to 0.03 m/s.

The results of streamlines plots, figure 7.7, clearly depicting the flow pattern structure in the cavity. This is to make a vision comparison between LBM and Finite Volume flows.

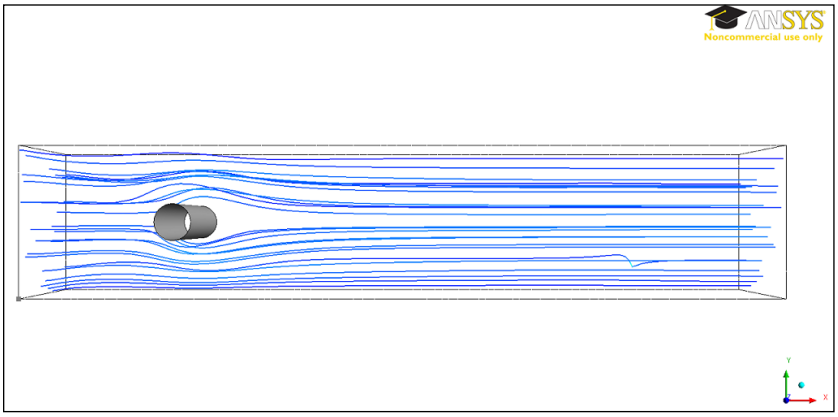


Figure 7.7 3D Streamline view with 25 seed points.

## 7.5 Comparison of the results

In this section, the provided computational results will be compared with regard to accuracy, performance, timing measurements and memory requirements.

The goal is to take into account four points within the domain; at each point the velocity will be studied in order to find out the difference between the two approaches. The reason of the position of the points is in function on the main peaks of speed. The closest point to the inlet is Point 1; it is defined in order to study the income velocity before the contact with the obstacle. Point 2 is the highest in y-direction; it is useful for studying the change of the velocity profile due to the boundary conditions and the obstacle. The middle point after the obstacle is Point 3; the inlet flow is deviated because of the obstacle. Point 3 is able to figure out the changes within the fluid flow as a result of the obstacle. The furthest point from the inlet but closest to the outlet is Point 4.

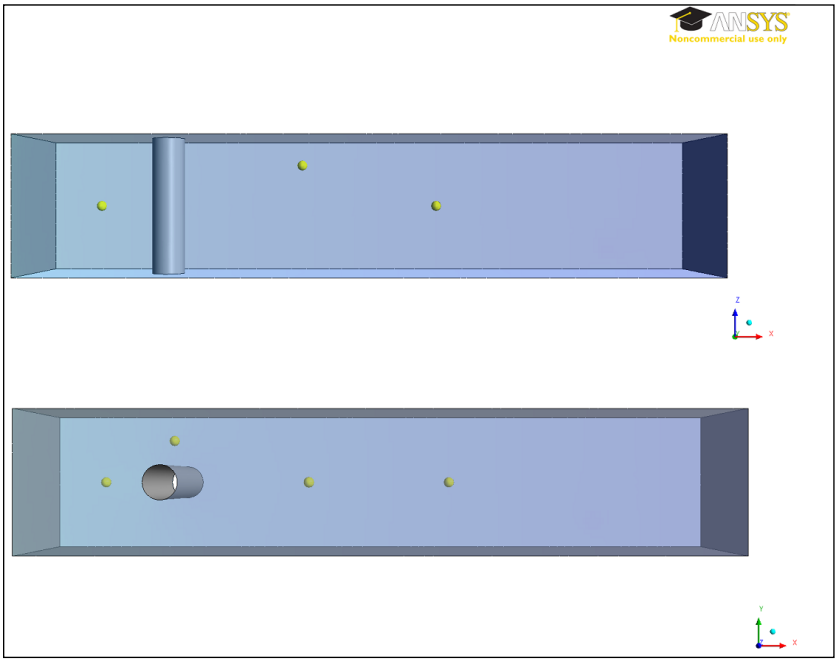


Figure 7.9 Position of the points.

Table 7.1 shows the results of the velocity for the components, x, y and z using Ansys. The last value **v** is the vectorial sum of the three components.

| **CFX** | **x** | **y** | **z** | $\mathbf{u_x}$ | $\mathbf{u_y}$ | $\mathbf{u_z}$ | **v** |
|---|---|---|---|---|---|---|---|
| **Point 1** | 25 | 25 | 25 | 2,49E-02 | -4,50E-05 | 1,42E-05 | 2,490E-02 |
| **Point 2** | 50 | 40 | 25 | 3,96E-02 | 4,05E-03 | 3,50E-05 | 3,981E-02 |
| **Point 3** | 150 | 25 | 25 | 3,56E-02 | 1,23E-06 | 1,83E-05 | 3,560E-02 |
| **Point 4** | 100 | 25 | 40 | 2,30E-02 | 4,81E-05 | 2,11E-04 | 2,300E-02 |

Table 7.1 Velocity results for CFX analysis.

| **LBM** | **x** | **y** | **z** | $\mathbf{u_x}$ | $\mathbf{u_y}$ | $\mathbf{u_z}$ | **v** |
|---|---|---|---|---|---|---|---|
| **Point 1** | 25 | 25 | 25 | 1,87E-02 | 2,41E-04 | -2,27E-06 | 1,872E-02 |
| **Point 2** | 50 | 40 | 25 | 1,96E-02 | 1,41E-04 | 1,28E-06 | 1,962E-02 |
| **Point 3** | 150 | 25 | 25 | 1,99E-02 | 1,13E-06 | 7,00E-07 | 1,986E-02 |
| **Point 4** | 100 | 25 | 40 | 1,64E-02 | 3,39E-06 | 3,04E-05 | 1,635E-02 |

Table 7.2 Velocity results for LBM analysis.

| | x | y | z | Δv |
|---|---|---|---|---|
| **Point 1** | 25 | 25 | 25 | 25% |
| **Point 2** | 50 | 40 | 25 | 51% |
| **Point 3** | 150 | 25 | 25 | 44% |
| **Point 4** | 100 | 25 | 40 | 29% |

Table 7.3 Comparison between CFX and LBM results.

The comparison between CFX and LBM values are shown in table 7.3. The percentage represents the error taking into account the Ansys' value as the reference.

Point 2, below the obstacle, has a difference equal to 51%; that is due to two main reasons. With the old release of Ansys, a parabolic velocity profile is unfeasible to set up at the inlet. Indeed, a constant velocity is clearly easy to define. The speed of the point (0,40,25) in the inlet is equal to 0.02 m/s, indeed, in the Poiseuille profile, it is equal to 0.0097 m/s, almost the half of the previous value. If we study the discrepancy between the two different velocities at the inlet, the difference in percentage is equal to 51%. Another difference is imposed by the mesh. Non-uniform triangular and quadrangular mesh is defined close to the circular obstacle, indeed, a square grid is set up in LBM analysis with an accuracy equal to the length of the cubic mesh. For instance, reducing the size of the D3Q27 model we are able to decrease the error up to 5 percentage points. This is a limitation of the commercial softwares; usually they are less flexible and adaptable to specific applications.

The performance in terms of time measurements and memory required is calculated using Linux-Debian operating system with a processor 2.53 GHz Intel Core 2 Duo and RAM 4 GByte 1067 MHz. The three-dimension LBM model is fixed with 625000 nodes with 500 steps in order to have a good accuracy. The time required to solve the model is 8 minutes and 32 seconds against the 15 minutes and 47 seconds of the CFX analysis.

## 7.6 Conclusion

Generally, in most practical situations, the systems are too complex to be reduced to a system of equations which are analytically tractable. PDEs are translated to algebraic difference equations which are then solved locally at each space sub-division generally speaking therefore of Finite-Difference or Finite-Element. LBM represents an additional approach based, first of all, on the advantage of the simplicity of coding. That is one the reason why LBM is preferred in the simulation of Rayleigh-Benard convection or the von Karman vortex street. Moreover, the LBM is a valid approach in order to study the nanoscale fluid flow such as blood analysis .

In terms of accuracy, it increases progressively from 3D models with 9 nodes reaching the greatest value with 27 nodes. Related to the state of the art, the mesh with 27 nodes is the best studied till now even if the differences between 27 and 19 nodes are not so relevant in terms of accuracy, memories required and computational time.

The order of convergence of LBM is exhibited for the second-order convergence with low memory required. The source code was developed without any temporary memory for the streaming and collision step. The memory required is in function of the order of convergence and the number of mesh; each mesh has 27 distribution functions stored in a swap memory.

Another great advantage is the timing measurements. The code written in Fortran90 compared with the finite volume analysis with the same number of mesh needs half of the time and lower CPU frequency. It is also true that a commercial software needs to run subroutines that are not needed for the model but implemented inside the commercial code.

In terms of accuracy, the commercial software needs almost the double time than the Lattice Boltzmann Method in order to get at least the same precision as the source code. This is due to the numerous modules that the software has to compile and analyze while the code compiles just what it is written into. Trying to change only the computer's hardware with a better processor's frequency up to 3.16 GHz Intel core 2 Duo and RAM 32 GByte, the time

required to obtain the same accuracy decreases much more for CFX analysis than LBM. This is due to the increasing of the RAM: if the RAM is not enough for the analysis, Ansys tries to storage data on a swap memory on the hard disk drive. Because of the mechanical design limitations the speed of the data access is lower than a volatile Random-Access memory modules. That is another reason why CFX needs more time to run.

One of the last aspect to take into account is the memory required; in both cases the memory is in function of the number of mesh and the number of loops of the code. If a better accuracy is required, the number of loops has to be increased and the memory rises. A great feature of Debian OS is the possibility to use swap memory on the hard disk drive. On the other hand the performance in frequency time measuring decreases exponentially due to the mechanical design limitation.

# Appendix A

## D3Q27 Lattice Boltzmann Code

The following code presented is the summary of the LBM in 3D with D3Q27 model written in Fortran90.

```
!          ========================================================
!          Constants that identify different cell-types according
!        to the dynamics they implement
!          ========================================================
MODULE cellConst
    integer, parameter:: fluid = 0, wall = 1, inlet = 10, outlet = 11
END MODULE cellConst


!          ========================================================
!          Lattice constants for the D3Q27 lattice
!          ========================================================
MODULE D3Q27Const

!          D3Q27 Weights
    double precision,parameter:: t(0:26) = (/              &
&  8.0d0/27.0d0,2.0d0/27.0d0,2.0d0/27.0d0,2.0d0/27.0d0     &
&, 2.0d0/27.0d0,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0     &
&, 1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0     &
&, 1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0,1.0d0/54.0d0     &
&, 1.0d0/54.0d0,2.0d0/27.0d0,2.0d0/27.0d0,1.0d0/216.0d0    &
&, 1.0d0/216.0d0,1.0d0/216.0d0,1.0d0/216.0d0,1.0d0/216.0d0 &
&, 1.0d0/216.0d0,1.0d0/216.0d0,1.0d0/216.0d0/)


!          D3Q27 Directions
    integer:: v(0:26,0:2)
!       = (/(/0,1,0,-1,0,1,-1,-1,1/),(/0,0,1,0,-1,1,1,-1,-1/)/)

!        D3Q27 Opposite Bounce Back
    integer, parameter:: opposite(0:26) = (/
              0,3,4,1,2,7,8,5,6,15,16,13,14,11,12,
              9,10,18,17,25,26,23,24,21,22,19,20/)

END MODULE D3Q27Const
```

```
!          ========================================================
!          Constants for simulation setup
!          ========================================================
MODULE simParam
!        Dimension of the domain
    integer, parameter:: xDim   = 250
    integer, parameter:: yDim   = 50
    integer, parameter:: zDim   = 50
    integer, parameter:: obstX  = xDim/5
    integer, parameter:: obstY  = yDim/2
    integer, parameter:: obstR  = yDim/10+1

!        Adimensional Mesh size:
    double precision, parameter:: deltaT = 0.3d0

!        Increment numbers
    double precision, parameter:: deltaX = xDim / deltaT
    double precision, parameter:: deltaY = yDim / deltaT
    double precision, parameter:: deltaZ = zDim / deltaT

!        Number of steps
    integer, parameter:: tMax = 700

!        Variables defined
    double precision, parameter:: uMax = 0.02d0
    double precision, parameter:: Re = 10.0d0
END MODULE simParam

!          ========================================================
!          The main program, implementing a flow past a cylinder
!          ========================================================
PROGRAM unsteady
    USE simParam, ONLY: xDim, yDim, zDim, tMax
    implicit none

    double precision:: omega, time1, time2, timeTot
    double precision, dimension(:,:,:,:), allocatable:: u, fEq, f
    double precision, dimension(:,:,:), allocatable:: rho, uSqr
    integer, dimension(:,:,:), allocatable:: image
    integer:: tStep,z,y,x,i

    allocate(f(zDim,yDim,xDim,0:26))
    allocate(fEq(zDim,yDim,xDim,0:26))
    allocate(u(zDim,yDim,xDim,0:2))
    allocate(uSqr(zDim,yDim,xDim))
    allocate(rho(zDim,yDim,xDim))
    allocate(image(zDim,yDim,xDim))

    CALL constructImage(image)
    CALL computeOmega(omega)
    CALL initMacro(rho,u,uSqr)
    CALL computeFeq(fEq,rho,u,uSqr)
```

```
    do i = 0, 26
        do x = 1, xDim
            do y = 1, yDim
                do z = 1, zDim
                    f(z,y,x,i) = fEq(z,y,x,i)
                end do
            end do
        end do
    end do

    timeTot = 0.0d0

    do tStep = 1, tMax
        CALL CPU_TIME(time1)
        CALL inletOutlet(f,rho,u,image)
        CALL boundaries(f,image)
        CALL computeMacros(image,f,rho,u,uSqr)
        CALL computeFeq(fEq,rho,u,uSqr)
        CALL collide(f,fEq,omega,image)
        CALL stream(f)
        CALL CPU_TIME(time2)
        write(*,*) 'tStep = ', tStep
        timeTot = timeTot + (time2-time1)
    end do

    CALL writeInput(omega)
    CALL writeImage(image)
    CALL writeOutput(u,0)
    CALL writeOutputMATLAB(u,0)
    write(*,*) dble(tMax) * (dble(yDim * xDim)) &
               & / timeTot ,'cells per second'

    deallocate(f)
    deallocate(fEq)
    deallocate(u)
    deallocate(uSqr)
    deallocate(rho)
    deallocate(image)

END PROGRAM unsteady

!          ========================================================
!          Compute the relaxation parameter from the Reynolds number
!          ========================================================
SUBROUTINE computeOmega(omega)
    USE simParam, ONLY: Re, uMax, obstR, deltaT

    implicit none

    double precision, INTENT(INOUT):: omega
    double precision:: nu

    nu    =  uMax * 2.0d0 * dble(obstR) / Re
    omega = 1.0d0 / (((3.0d0*nu)/deltaT)+0.5d0)

END SUBROUTINE computeOmega
```

```
!          ========================================================
!          Construct an array the defines the flow geometry
!          ========================================================
SUBROUTINE constructImage(image)
    USE cellConst
    USE simParam, ONLY: xDim, yDim, zDim, obstX, obstY, obstR
    USE D3Q27Const, ONLY: v

    implicit none

    integer, INTENT(INOUT):: image(zDim,yDim,xDim)
    integer:: x,y,z

!        Nodes positions
    v(0:26,0) = (/0,1,0,-1, 0,1,-1,-1, 1,1,0,-1, 0, 1, 0,-1, 0,0, &
                 & 0,  1, -1, -1, 1, 1, -1, -1, 1/)
    v(0:26,1) = (/0,0,1, 0,-1,1, 1,-1,-1,0,1, 0,-1, 0, 1, 0,-1,0, &
                 & 0,  1,  1, -1,-1, 1,  1, -1,-1/)
    v(0:26,2) = (/0,0,0, 0, 0,0, 0, 0, 0,1,1, 1, 1,-1,-1,-1,-1,1, &
                 & -1,  1,  1,  1, 1,-1, -1, -1,-1/)

!        Definition of the Boundary
    image(:,:,:)     = fluid
    image(:,:,1)     = inlet
    image(:,:,xDim)  = outlet
    image(:,1,:)     = wall
    image(:,yDim,:)  = wall
    image(1,:,:)     = wall
    image(zDim,:,:)  = wall

    do x = 1, xDim
        do y = 1, yDim
            do z = 1, zDim
                if (((x-obstX)**2 + (y-obstY)**2) <= (obstR**2) ) &
                   &   image(z,y,x) = wall
            end do
        end do
    end do

END SUBROUTINE constructImage
```

```
!          ============================================================
!          Initialize the simulation to Poiseuille profile at
!        an equilibrium distribution
!          ============================================================
SUBROUTINE initMacro(rho,u,uSqr)

    USE simParam, ONLY: xDim, yDim, zDim

    implicit none

    double precision, INTENT(INOUT)::rho(zDim,yDim,xDim), &
             & u(zDim,yDim,xDim,0:2),uSqr(zDim,yDim,xDim)
    double precision:: uProf

    integer:: y,z


    do z = 1, zDim
        do y = 1, yDim
            u(z,y,:,0) = uProf(y)
            u(z,y,:,1) = 0.0d0
            u(z,y,:,2) = 0.0d0
        end do
    end do
    rho  = 1.0d0
    uSqr(:,:,:) = u(:,:,:,0) * u(:,:,:,0) + u(:,:,:,1) * u(:,:,:,1) + &
                & u(:,:,:,2) * u(:,:,:,2)

END SUBROUTINE initMacro

!          ============================================================
!          Compute equilibrium distribution
!          ============================================================
SUBROUTINE computeFeq(fEq,rho,u,uSqr)
    USE D3Q27COnst, ONLY: t, v
    USE simParam, ONLY: xDim, yDim, zDim
    implicit none

       double precision, INTENT(IN):: rho(zDim,yDim,xDim), uSqr(zDim,yDim,xDim), u
(zDim,yDim,xDim,0:2)
    double precision, INTENT(INOUT):: fEq(zDim,yDim,xDim,0:26)
    integer:: i, x, y, z
    double precision:: uxyz

    do i = 0, 26
        do x = 1, xDim
            do y = 1, yDim
                do z = 1, zDim
                    uxyz = u(z,y,x,0) * v(i,0) + u(z,y,x,1) * v(i,1)  &
                    & + u(z,y,x,2) * v(i,2)
                    fEq(z,y,x,i) = t(i) * rho(z,y,x) * (1.0d0+3.0d0 * &
                    & Uxyz + 4.5d0 * uxyz * uxyz - 1.5d0 * uSqr(z,y,x))
                end do
            end do
        end do
    end do
END SUBROUTINE computeFeq
```

```
!          ========================================================
!          Compute velocity from distribution functions
!          ========================================================
SUBROUTINE computeMacros(image,f,rho,u,uSqr)
    USE simParam, ONLY: xDIm, yDim, zDim
    implicit none
    integer, INTENT(IN):: image(zDim,yDim,xDim)
    double precision, INTENT(IN):: f(zDim,yDim,xDim,0:26)
    double precision, INTENT(INOUT)::u(zDim,yDim,xDim,0:2), &
                & rho(zDim,yDim,xDim),uSqr(zDim,yDim,xDim)
    integer:: x,y,z

    do x = 1, xDim
        do y = 1, yDim
            do z = 1, zDim
                rho(z,y,x)=(f(z,y,x,0) + f(z,y,x,1) + f(z,y,x,2)    &
        & + f(z,y,x,3)  + f(z,y,x,4)  + f(z,y,x,5)  + f(z,y,x,6)    &
        & + f(z,y,x,7)  + f(z,y,x,8)  + f(z,y,x,9)  + f(z,y,x,10)   &
        & + f(z,y,x,11) + f(z,y,x,12) + f(z,y,x,13) + f(z,y,x,14)   &
        & + f(z,y,x,15) + f(z,y,x,16) + f(z,y,x,17) + f(z,y,x,18)   &
        & + f(z,y,x,19) + f(z,y,x,20) + f(z,y,x,21) + f(z,y,x,22)   &
        & + f(z,y,x,23) + f(z,y,x,24) + f(z,y,x,25) + f(z,y,x,26))  &


                u(z,y,x,0)=((f(z,y,x,1) - f(z,y,x,3) + f(z,y,x,5)   &
        & - f(z,y,x,6)  - f(z,y,x,7)  + f(z,y,x,8)   + f(z,y,x,9)   &
        & - f(z,y,x,11) + f(z,y,x,13) - f(z,y,x,15)  + f(z,y,x,19)  &
        & - f(z,y,x,20) + f(z,y,x,23) - f(z,y,x,24)  + f(z,y,x,26)  &
        & - f(z,y,x,25) + f(z,y,x,22) - f(z,y,x,21)) / rho(z,y,x))

                u(z,y,x,1)=((f(z,y,x,2) - f(z,y,x,4) + f(z,y,x,5)   &
        & + f(z,y,x,6)  - f(z,y,x,7)  - f(z,y,x,8)  + f(z,y,x,14)   &
        & - f(z,y,x,16) + f(z,y,x,10) - f(z,y,x,12) + f(z,y,x,19)   &
        & - f(z,y,x,22) + f(z,y,x,20) - f(z,y,x,21) + f(z,y,x,24)   &
        & - f(z,y,x,25) + f(z,y,x,23) - f(z,y,x,26)) / rho(z,y,x))

                u(z,y,x,2)=((f(z,y,x,9) - f(z,y,x,13) + f(z,y,x,17) &
        & - f(z,y,x,18) + f(z,y,x,11) - f(z,y,x,15) + f(z,y,x,10)   &
        & - f(z,y,x,14) + f(z,y,x,12) - f(z,y,x,16) + f(z,y,x,19)   &
        & - f(z,y,x,23) + f(z,y,x,20) - f(z,y,x,24) + f(z,y,x,21)   &
        & - f(z,y,x,25) + f(z,y,x,22) - f(z,y,x,26)) / rho(z,y,x))

                uSqr(z,y,x) = u(z,y,x,0) * u(z,y,x,0) + u(z,y,x,1)  &
        & * u(z,y,x,1) + u(z,y,x,2) * u(z,y,x,2)

            end do
        end do
    end do
END SUBROUTINE computeMacros
```

```
!          ========================================================
!          Implement Bounce-back on upper/lower boundaries
!          ========================================================
SUBROUTINE boundaries(f,image)
    USE D3Q27Const, ONLY: opposite
    USE cellConst, ONLY: wall
    USE simParam, ONLY: xDim, yDim, zDim
    implicit none

    integer, INTENT(IN):: image(zDim,yDim,xDim)
    double precision, INTENT(INOUT):: f(zDim,yDim,xDim,0:26)
    double precision:: fTmp(0:26)
    integer:: i, x, z, y
   do x = 1, xDim
        do y = 1, yDim
            do z = 1, zDim
                if (image(z,y,x) == wall) then
                    do i = 0, 26
                        fTmp(i) = f(z,y,x,opposite(i))
                    end do
                    do i = 0, 26
                        f(z,y,x,i) = fTmp(i)
                    end do
                end if
            end do
        end do
    end do
END SUBROUTINE boundaries

!          ========================================================
!          Use Zou/He boundary condition to implement Dirichlet
!        boundaries on inlet/outlet
!          ========================================================
SUBROUTINE inletOutlet(f,rho,u,image)
    USE cellConst, ONLY: inlet, outlet
    USE simParam

    implicit none

    double precision, INTENT(INOUT):: f(zDim,yDim,xDim,0:26), &
                  & u(zDim,yDim,xDim,0:2),rho(zDim,yDim,xDim)
    integer, INTENT(IN):: image(zDim,yDim,xDim)

    double precision:: uProf
    integer:: x, y, z

    do x = 1, xDim
        do y = 1, yDim
            do z = 1, zDim
                if (image(z,y,x) == inlet) then
                    u(z,y,x,0) = uProf(y)
                    u(z,y,x,1) = 0.0d0
                    u(z,y,x,2) = 0.0d0
```

```
                else if (image(z,y,x) == outlet) then
                    u(z,y,x,0) = uProf(y)
                    u(z,y,x,1) = 0.0d0
                    u(z,y,x,2) = 0.0d0
                end if
            end do
        end do
    end do
CONTAINS
!          ============================================================
!          Computation of Poiseuille profile for the inlet/outlet
!          ============================================================
FUNCTION uProf(y)
    USE simParam, ONLY: yDIm, uMax
    implicit none

    integer, INTENT(IN):: y
    double precision:: radius, uProf

    radius = dble(yDim-1) * 0.5d0
    uProf  = -uMax * ((abs(1 - dble(y-1) / radius))**2 - 1.0d0)
END FUNCTION uProf

!          ============================================================
!          Streaming step: the population functions are shifted
!        one site along their corresponding lattice direction
!        (no temporary memory is needed)
!          ============================================================
SUBROUTINE stream(f)
    USE simParam
    implicit none

    double precision, INTENT(INOUT):: f(zDim,yDim,xDim,0:26)
    double precision:: periodicHor(xDim), periodicVert(yDim), &
         & periodicWid(zDim)

!          ---------------------------------------
!          right direction
    periodicVert    = f(1,:,xDim,1)
    f(1,:,2:xDim,1) = f(1,:,1:xDim-1,1)
    f(1,:,1,1)      = periodicVert
!          ---------------------------------------
!          up direction
    periodicHor       = f(1,yDim,:,2)
    f(1,2:yDim,:,2)   = f(1,1:yDim-1,:,2)
    f(1,1,:,2)        = periodicHor
!          ---------------------------------------
!          left direction
    periodicVert      = f(1,:,1,3)
    f(1,:,1:xDim-1,3) = f(1,:,2:xDim,3)
    f(1,:,xDim,3)     = periodicVert
!          ---------------------------------------
!          down direction
    periodicHor        = f(1,1,:,4)
    f(1,1:yDim-1,:,4)  = f(1,2:yDim,:,4)
    f(1,yDim,:,4)      = periodicHor
```

```
!          ------------------------------------
!          up-right direction
    periodicHor          = f(1,yDim,:,5)
    periodicVert         = f(1,:,xDim,5)
    f(1,2:yDim,2:xDim,5) = f(1,1:yDim-1,1:xDim-1,5)
    f(1,1,1,5)           = periodicVert(yDim)
    f(1,2:yDim,1,5)      = periodicVert(1:yDim-1)
    f(1,1,2:xDim,5)      = periodicHor(1:xDim-1)
!          ------------------------------------
!          up-left direction
    periodicVert           = f(1,:,1,6)
    periodicHor            = f(1,yDim,:,6)
    f(1,2:yDim,1:xDim-1,6)   = f(1,1:yDim-1,2:xDim,6)
    f(1,2:yDim,xDim,6)       = periodicVert(1:yDim-1)
    f(1,1,xDim,6)            = periodicVert(yDim)
    f(1,1,1:xDim-1,6)        = periodicHor(2:xDim)
!          ------------------------------------
!          down-left direction
    periodicVert           = f(1,:,1,7)
    periodicHor            = f(1,1,:,7)
    f(1,1:yDim-1,1:xDim-1,7) = f(1,2:yDim,2:xDim,7)
    f(1,1:yDim-1,xDim,7)     = periodicVert(2:yDim)
    f(1,yDim,xDim,7)         = periodicVert(1)
    f(1,yDim,1:xDim-1,7)     = periodicHor(2:xDim)
!          ------------------------------------
!          down-right direction
    periodicVert         = f(1,:,xDim,8)
    periodicHor          = f(1,1,1:xDim,8)
    f(1,1:yDim-1,2:xDim,8) = f(1,2:yDim,1:xDim-1,8)
    f(1,1:yDim-1,1,8)      = periodicVert(2:yDim)
    f(1,yDim,1,8)          = periodicVert(1)
    f(1,yDim,2:xDim,8)     = periodicHor(1:xDim-1)
!          ------------------------------------
!        up-in direction
    periodicVert              = f(zDim,:,1,10)
    periodicWid               = f(:,yDim,1,10)
    f(2:zdim,2:yDim,:,10)     = f(1:zdim-1,1:yDim-1,:,10)
    f(1,2:yDim,1,10)          = periodicVert(1:yDim-1)
    f(1,1,1,10)               = periodicVert(yDim)
    f(2:zDim,1,1,10)          = periodicWid(1:zDim-1)
!          ------------------------------------
!        up-out direction
    periodicVert              = f(1,:,1,14)
    periodicWid               = f(:,yDim,1,14)
    f(1:zDim-1,2:yDim,:,14)   = f(2:zDim,1:yDim-1,:,14)
    f(zDim,2:yDim,1,14)       = periodicVert(1:yDim-1)
    f(zDim,1,1,14)            = periodicVert(yDim)
    f(2:zDim,1,1,14)          = periodicWid(1:zDim-1)
!          ------------------------------------
```

```
!       down-out direction
    periodicVert              = f(1,:,1,16)
    periodicWid               = f(:,1,1,16)
    f(1:zDim-1,1:yDim-1,:,16) = f(2:zDim,2:yDim,:,16)
    f(zDim,1:yDim-1,1,16)     = periodicVert(2:yDim)
    f(zDim,yDim,1,16)         = periodicVert(1)
    f(1:zDim-1,yDim,1,16)     = periodicWid(2:zDim)
!         ---------------------------------------
!       down-in direction
    periodicVert              = f(zDim,:,1,12)
    periodicWid               = f(:,1,1,12)
    f(2:zDim,1:yDim-1,:,12)   = f(1:zDim-1,2:yDim,:,12)
    f(1,1:yDim-1,1,12)        = periodicVert(2:yDim)
    f(1,yDim,1,12)            = periodicWid(zDim)
    f(2:zDim,yDim,1,12)       = periodicWid(1:zDim-1)
!         ---------------------------------------
!       in-right direction
    periodicWid          = f(:,1,xDim,9)
    periodicHor           = f(zDim,1,:,9)
    f(2:zDim,:,2:xDim,9) = f(1:zDim-1,:,1:xDim-1,9)
    f(2:zDim,1,1,9)      = periodicWid(1:zDim-1)
    f(1,1,1,9)           = periodicWid(zDim)
    f(1,1,2:xDim,9)      = periodicHor(1:xDim-1)
!         ---------------------------------------
!       in-left direction
    periodicWid             = f(:,1,1,11)
    periodicHor             = f(zDim,1,:,11)
    f(2:zDim,:,1:xDim-1,11) = f(1:zDim-1,:,2:xDim,11)
    f(2:zDim,1,xDim,11)     = periodicWid(1:zDim-1)
    f(1,1,xDim,11)          = periodicWid(zDim)
    f(1,1,1:xDim-1,11)      = periodicHor(2:xDim)
!         ---------------------------------------
!       out-left direction
    periodicWid               = f(:,1,1,15)
    periodicHor               = f(1,1,:,15)
    f(1:zDim-1,:,1:xDim-1,15) = f(2:zDim,:,2:xDim,15)
    f(1:zDim-1,1,xDim,15)     = periodicWid(2:zDim)
    f(zDim,1,xDim,15)         = periodicWid(1)
    f(zDim,1,1:xDim-1,15)     = periodicHor(2:xDim)
!         ---------------------------------------
!       out-right direction
    periodicWid               = f(:,1,xDim,13)
    periodicHor               = f(1,1,:,13)
    f(1:zDim-1,:,2:xDim,13) = f(2:zDim,:,1:xDim-1,13)
    f(1:zDim-1,1,1,13)      = periodicWid(2:zDim)
    f(zDim,1,1,13)          = periodicWid(zDim)
    f(zDim,1,2:xDim,13)     = periodicHor(1:xDim-1)
!         ---------------------------------------
!       in-right-up direction
    periodicVert               = f(zDim,:,xDim,19)
    periodicHor                = f(zDim,yDim,:,19)
    periodicWid                = f(:,ydim,xDim,19)
    f(2:zDim,2:yDim,2:xDim,19) = f(1:zDim-1,1:yDim-1,1:xDim-1,19)
    f(1,2:yDim,1,19)           = periodicVert(1:yDim-1)
    f(1,1,1,19)                = periodicVert(yDim)
    f(1,1,2:xDim,19)           = periodicHor(1:xDim-1)
    f(2:zDim,1,1,19)           = periodicWid(1:zDim-1)
```

```
!          ------------------------------------
!        in-left-up direction
    periodicVert                 = f(zDim,:,1,20)
    periodicHor                  = f(zDim,yDim,:,20)
    periodicWid                  = f(:,yDim,1,20)
    f(2:zDim,2:yDim,1:xDim-1,20) = f(1:zDim-1,1:yDim-1,2:xDim,20)
    f(1,2:yDim,xDim,20)          = periodicVert(1:yDim-1)
    f(1,1,xDim,20)               = periodicVert(yDim)
    f(2:zDim,1,xDim,20)          = periodicWid(1:zDim-1)
    f(zDim,yDim,1:xDim-1,20)     = periodicHor(2:xDim)
!          ------------------------------------
!        in-right-down direction
    periodicVert                 = f(zDim,:,xDim,22)
    periodicHor                  = f(zDim,1,:,22)
    periodicWid                  = f(:,1,xDim,22)
    f(2:zDim,1:yDim-1,2:xDim,22) = f(1:zDim-1,2:yDim,1:xDim-1,22)
    f(1,1:yDim-1,1,22)           = periodicVert(2:yDim)
    f(1,yDim,1,22)               = periodicVert(1)
    f(zDim,1,2:xDim,22)          = periodicHor(1:xDim-1)
    f(2:zDim,yDim,1,22)          = periodicWid(1:zDim-1)
!          ------------------------------------
!        in-left-down direction
    periodicVert                   = f(zDim,:,1,21)
    periodicHor                    = f(zDim,1,:,21)
    periodicWid                    = f(:,1,1,21)
    f(2:zDim,1:yDim-1,1:xDim-1,21) = f(1:zDim-1,2:yDim,2:xDim,21)
    f(1,1:yDim-1,xDim,21)          = periodicVert(2:yDim)
    f(1,yDim,xDim,21)              = periodicVert(1)
    f(1,yDim,1:xDim-1,21)          = periodicHor(2:xDim)
    f(2:zDim,yDim,xDim,21)         = periodicWid(1:zDim-1)
!          ------------------------------------
!        out-right-up direction
    periodicVert                 = f(1,:,xDim,23)
    periodicHor                  = f(1,yDim,:,23)
    periodicWid                  = f(:,yDim,xDim,23)
    f(1:zDim-1,2:yDim,2:xDim,23) = f(2:zDim,1:yDim-1,1:xDim-1,23)
    f(zDim,2:yDim,1,23)          = periodicVert(1:yDim-1)
    f(zDim,1,1,23)               = periodicVert(yDim)
    f(1,yDim,2:xDim,23)          = periodicHor(1:xDim-1)
   f(1:zDim-1,1,1,23)           = periodicWid(2:zDim)
!          ------------------------------------
!        out-left-up direction
    periodicVert                   = f(1,:,1,24)
    periodicHor                    = f(1,yDim,:,24)
    periodicWid                    = f(:,yDim,1,24)
    f(1:zDim-1,2:yDim,1:xDim-1,24) = f(2:zDim,1:yDim-1,2:xDim,24)
    f(zDim,2:yDim,xDim,24)         = periodicVert(1:yDim-1)
    f(zDim,1,xDim,24)              = periodicVert(yDim)
    f(zDim,1,1:xDim-1,24)          = periodicHor(2:xDim)
    f(1:zDim-1,1,xDim,24)          = periodicWid(2:zDim)
!          ------------------------------------
```

```
!        out-left-down direction
    periodicVert                     = f(1,:,1,25)
    periodicHor                      = f(1,1,:,25)
    periodicWid                      = f(:,1,1,25)
    f(1:zDim-1,1:yDim-1,1:xDim-1,25) = f(2:zDim,2:yDim,2:xDim,25)
    f(zDim,1:yDim-1,xDim,25)         = periodicVert(2:yDim)
    f(zDim,yDim,xDim,25)             = periodicVert(1)
    f(zDim,yDim,1:xDim-1,25)         = periodicHor(2:xDim)
    f(1:zDim-1,yDim,xDim,25)         = periodicWid(2:zDim)
!          ------------------------------------
!        out-right-down direction
    periodicVert                   = f(1,:,xDim,26)
    periodicHor                    = f(1,1,:,26)
    periodicWid                    = f(:,1,xDim,26)
    f(1:zDim-1,1:yDim-1,2:xDim,26) = f(2:zDim,2:yDim,1:xDim-1,26)
    f(zDim,1:yDim-1,1,26)          = periodicVert(2:yDim)
    f(zDim,yDim,1,26)              = periodicVert(1)
    f(zDim,yDim,2:xDim,26)         = periodicHor(1:xDim-1)
    f(1:zDim-1,yDim,1,26)          = periodicWid(2:zDim)


END SUBROUTINE stream

!          ========================================================
!          LBGK collision step
!          ========================================================
SUBROUTINE collide(f,fEq,omega,image)
    USE simParam, ONLY: xDim, yDim, zDim
    USE cellConst, ONLY: wall
    implicit none

    integer, INTENT(IN):: image(zDim,yDim,xDim)
    double precision, INTENT(IN):: fEq(zDim,yDim,xDim,0:26), omega
    double precision, INTENT(INOUT):: f(zDim,yDim,xDim,0:26)

    integer:: x,y,z,i

   do i = 0, 26
        do x = 1, xDim
            do y = 1, yDim
                do z = 1, zDim
                    if (image(z,y,x) /= wall) f(z,y,x,i) =  &
                    &  * f(z,y,x,i) + omega * fEq(z,y,x,i)
                end do
            end do
        end do
    end do

END SUBROUTINE collide
```

```
!          ============================================================
!          Write the components of the velocity to a text file,
!        with indices (x,y)
!          ============================================================
SUBROUTINE writeOutput(u,tStep)
    USE simParam, ONLY: xDim, yDim, zDim
    implicit none
    integer, INTENT(IN):: tStep
    double precision, INTENT(IN):: u(zDim,yDim,xDim,0:2)

    integer:: x,y,z
    character (LEN=100):: fileName

    write(fileName,*) tStep

    fileName = adjustl(fileName)
    open(13,file='output_Ux_Uy_Uz_3D_TECPLOT'//trim(fileName)//'.dat')

!-------number of field variables to be printed
          write(13,800)
800       format(' VARIABLES = X, Y, Z, Ux, Uy, Uz')
!-------field variables
          write(13,900) xDim,yDim,zDim
900  format(' ZONE T="VELOCITY"',' I=',i3,' J=',i2,' K=',i2,'  &
        & C= BLUE',' F=POINT')

    do z = 1, zDim
        do y=1, yDim
            do x=1, xDim
                write(13,102) x,y,z,u(z,y,x,0),u(z,y,x,1),u(z,y,x,2)
            end do
        end do
    end do
102     format(3i10,f20.10,f20.10,f20.10)
    close(13)
END SUBROUTINE writeOutput

!          ============================================================
!          Write the flow geometry to a file
!          ============================================================
SUBROUTINE writeImage(image)
    USE simParam, ONLY: xDim, yDim, zDim
    implicit none
    integer, INTENT(IN):: image(zDim,yDim,xDim)
    integer:: x,y,z

    open(13,file='output_Image_3D.dat')
    do x=1, xDim
        do y=1, yDim
            do z=1, zDim
                write(13,102) x,y,z,image(z,y,x)
            end do
        end do
    end do
102 format (3i10,3i10)
close(15)
END SUBROUTINE writeImage
```

```
!          ========================================================
!          MATLAB: Write the components of the velocity to a text
!        file, with indices (x,y)
!          ========================================================
SUBROUTINE writeOutputMATLAB(u,tStep)
    USE simParam, ONLY: xDim, yDim, zDim, tMax
    implicit none
    integer, INTENT(IN):: tStep
    double precision, INTENT(IN):: u(zDim,yDim,xDim,0:2)
    integer:: x,y,z
    character (LEN=100):: fileName,dir

    fileName = adjustl(fileName)

    open(15,file='output_Ux_Uy_Uz_3D_MATLAB_data_'// &
                & trim(fileName)//'.txt')
    do x=1, xDim
        do y=1, yDim
            do z=1, zDim
                write(15,102) x,y,z,u(z,y,x,0),u(z,y,x,1),u(z,y,x,2)
            end do
        end do
    end do
    close(15)

END SUBROUTINE writeOutputMATLAB

!          ========================================================
!          Print out simulation parameters to screen and create a
!        input_MATLAB.txt
!          ========================================================
SUBROUTINE writeInput(omega)
    USE simParam
    implicit none

    double precision, INTENT(IN):: omega
    character (LEN=100):: fileName

    write(*,*) 'xDim                = ', xDim
    write(*,*) 'yDim                = ', yDim
    write(*,*) 'zDim                = ', yDim
    write(*,*) 'Obstacle X          = ', obstX
    write(*,*) 'Obstacle Y          = ', obstY
    write(*,*) 'Obstacle Z          = ', zDim
    write(*,*) 'Obstacle Radius     = ', obstR
    write(*,*) 'Mesh Size           = ', deltaT
    write(*,*) 'tMax                = ', tMax
    write(*,*) 'uMax                = ', uMax
    write(*,*) 'Re                  = ', Re
    write(*,*) 'omega               = ', omega
```

```
    open(13,file='input_MATLAB.txt')
    write(13,*) xDim
    write(13,*) yDim
    write(13,*) zDim
    write(13,*) obstX
    write(13,*) obstY
    write(13,*) zDim
    write(13,*) obstR
    write(13,*) deltaT
    write(13,*) tMax
    write(13,*) uMax
    write(13,*) Re
    write(13,*) omega

    close(13)

END SUBROUTINE writeInput
```

## References

[1] D. Arumuga Perumal and Anoop K. Dass, *Simulation of Incompressible Flow in Two-Sided Lid-Driven Square Cavities. Part II* - LBM, Department of Mechanical Engineering, Indian Institute of Technology Guwahati.

[2] M.C. Sukop, D.T. Thorne Jr., *Lattice Boltzmann Modeling*, Springer.

[3] Ramon Gauci, *A History of Cellular Automata,* University of East London.

[4] B. Strader, K. Schubert, E. Gomez, J. Curnutt, P. Boston, *Simulating Spatial Differential Equations with Cellular Automata*, Department of Computer Science and Engineering, California State University, San Bernanrdino, CA, USA, Department of Earth and Environmental Science, New Mexico Tech, Socorro, NM, USA

[5] Harald Niesche, *Introduction to Cellular Automata*, Seminar "Organic Computing", 2006

[6] T.R. Hartka, *Cellular Automata for Structural Optimization on Reconfigurable Computers*, Master's thesis, Virginia Polytechnic Institute and State University, 2004.

[7] A. Peréz-Uribe, E. Sanchez, *FPGA implementation of an adaptable-size neural network*, Logic System Laboratory, Computer Science Department, Swiss Federal Institute of Technology-Lausanne.

[8] C. Burstedde, A. Kirchner, K. Klauck, A. Schadshneider, J. Zittartz, *Cellular Automata Approach to Pedestrian Dynamics - Applications*, 2001.

[9] D. Fey, D. Schmidt, *Marching-pixels: A new organic computing paradigm for smart sensor processor arrays*, ACM International Conference on Computing Frontiers, 2005.

[10] T. R. Kirkpatrick, M. H. Ernst, *Kinetic Theory for Lattice-Gas Cellular Automata*, Pysical Review A, Volume 44, Number 12, December 15th, 1991.

[11] J. Hardy, O. De Pazzis, Y. Pomeau, *Molecular dynamics of a classical lattice gas: transport properties and time correlation functions*, Phys. Rev. A 13, 1949-1961 (1976).

[12] U. Frisch, B. Hasslacher, and Y. Pomeau, *Lattice-Gas Automata for Navier-Stokes Equations*, Physical Review Letters, Volume 56, Number 14, April 7th, 1986.

[13] V.Tozzini, *Coarse-Grained models for proteins*, Elsevier.

[14] C. Moore, M. G. Nordahl, *Predicting Lattice Gases is P-complete*, Santa Fe Institute and Institute of Theoretical Physics, Chalmers University of Technology.

[15] Xiaoyi He, Gary D. Doolen, *Thermodynamic Foundations of Kinetic Theory and Lattice Boltzmann Models for Multiphase Flows*, Journal of Statistical Physics, Vol. 107, Nos 1/2, April 2002.

[16] Yong Shi, T.S. Zhao, and Z. L. Guo, *Thermal lattice Bhatnagar-Gross-Krook model for flows with viscous heat dissipation in the incompressible limit*, Phys, Rev. E, Volume 70 Issue 6.

[17] R. R. Nourgaliev, T.N. Dinh, T.G. Theofanous, and D, Joseph, *The Lattice Boltzmann Equation Method: Theoretical Interpretation, Numerics and Implications*, University of California, Santa Barbara, USA, University of Minnesota, USA.

[18] R. S. Maier, R. S. Bernard, and D. W. Grunau, *Boundary conditions for the Lattice Boltzmann Method*, American Institute of Physics.

[19] Joris C.G. Verschaeve, *Analysis of the lattice Boltzmann-Gross-Krook no-slip boundary conditions: Ways to improve accuracy and stability*, Department of Energy and Process Engineering, Norwegian University of Science and Technology.

[20] C. F. Ho, C. Chang, K. H. Lin, and C. A. Lin, *Consistent Boundary Conditions for 2D and 3D Lattice Boltzmann Simulations*, Tech Science Press, CMES, vol. 44 no. 2

[21] R. S. Maier, R. S. Bernard, D. W. Grunau, *Boundary conditions for the lattice Boltzmann method*, Phys. Fluids, Vol. 8, No. 7, July 1996.

[22] X. He and Q. Zou, *Analysis and boundary condition of the lattice Boltzmann BGK model with two velocity components*, Los Alamos preprint, LA-UR-95-2293.

[23] http://www.ansys.com/Industries/Academic, Ansys, Inc. Southpointe, Canonsburg.