REAL-TIME SIMULATION OF INDOOR AIR FLOW USING THE LATTICE BOLTZMANN METHOD ON GRAPHICS PROCESSING UNIT

NICOLAS DELBOSC

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

UNIVERSITY OF LEEDS

School of Mechanical Engineering
Faculty of Engineering

September 2015

ACKNOWLEDGMENTS

I would like to thank my supervisors, Dr. J.L. Summers, and Dr. A.I. Khan, for their continued support and invaluable discussions throughout my time as a PhD student at the University of Leeds.

Furthermore, I would like to extend my gratification to my fellow PhD students who helped make my time at the School of Mechanical Engineering both a rewarding and enjoyable experience.

Finally, I would like to thank NVIDIA for donating two K40 GPUs through the hardware donation program.

ABSTRACT

This thesis investigates the usability of the lattice Boltzmann method (LBM) for the simulation of indoor air flows in real-time. It describes the work undertaken during the three years of a Ph.D. study in the School of Mechanical Engineering at the University of Leeds, England.

Real-time fluid simulation, i.e. the ability to simulate a virtual system as fast as the real system would evolve, can benefit to many engineering application such as the optimisation of the ventilation system design in data centres or the simulation of pollutant transport in hospitals. And although real-time fluid simulation is an active field of research in computer graphics, these are generally focused on creating visually appealing animation rather than aiming for physical accuracy. The approach taken for this thesis is different as it starts from a physics based model, the lattice Boltzmann method, and takes advantage of the computational power of a graphics processing unit (GPU) to achieve real-time compute capability while maintaining good physical accuracy.

The lattice Boltzmann method is reviewed and detailed references are given a variety of models. Particular attention is given to turbulence modelling using the Smagorinsky model in LBM for the simulation of high Reynolds number flow and the coupling of two LBM simulations to simulate thermal flows under the Boussinesq approximation.

A detailed analysis of the implementation of the LBM on GPU is conducted. A special attention is given to the optimisation of the algorithm, and the program kernel is shown to achieve a performance of up to 1.5 billion lattice node updates per second, which is found to be sufficient for coarse real-time simulations. Additionally, a review of the real-time visualisation integrated within the program is presented and some of the techniques for automated code generation are introduced.

The resulting software is validated against benchmark flows, using their analytical solutions whenever possible, or against other simulation results obtained using accepted method from classical computational fluid dynamics (CFD) either as published in the literature or simulated in-house. The LBM is shown to resolve the flow with similar accuracy and in less time.

PUBLICATIONS & TALKS

Some ideas and figures have appeared previously in the following publications and conference talks:

PUBLICATIONS

- A.I. Khan, N. Delbosc, J.L. Summers, C.J. Noakes.
  Real-time flow simulation of indoor environments using the Lattice Boltzmann Method.
  Journal of Building Simulation. August 2015, Volume 8, Issue 4, Pages 405–414.
  http://dx.doi.org/10.1007/s12273-015-0232-9

- N. Delbosc, J.L. Summers, A.I. Khan, N. Kapur, C.J. Noakes.
  Optimised Implementation of the Lattice Boltzmann Method on a Graphics Processing Unit Towards Real-Time Fluid Simulation.
  Computer & Mathematics with Applications. February 2014, Volume 67, Issue 2, Pages 462–475.
  http://dx.doi.org/10.1016/j.camwa.2013.10.002

CONFERENCE TALKS

- N. Delbosc, K.H. Luo, J.L. Summers.
  Lattice Boltzmann Method for Turbulence : Real-Time Simulation and Big Data Processing.
  Whither Turbulence and Big Data for the 21st century, Corsica, France, April 20–24, 2015.

- N. Delbosc, A.I. Khan, J.L. Summers.
  Saving Energy in Data Centers Using Real-Time Simulation.
  GPU Technology Conference (GTC 2015), San Jose, California, USA, March 16–20, 2015.

- N. Delbosc.
  Real-Time Fluid Simulation.
  Villanova University, Philadelphia, USA, March 10, 2015. (invited talk)

- N. Delbosc, J. Summers, A. Khan, C. Noakes.
  Real-time simulation and visualisation of air flow in datacenters and hospitals.
  23rd International Conference on Discrete Simulation of Fluid Dynamics (DSFD 2014), Paris, France, July 28 – August 1, 2014.

- N. Delbosc, G de Boer, J.L. Summers.
  Managing data centres with the aid of real-time airflow simulations.
  Datacentre Transformation Conference (DTC 2014), Manchester, U.K., July 8, 2014

- N. Delbosc, A.I. Khan, J.L. Summers, C.J. Noakes.
  Real-time Simulations and Visualisation of Air Flow in Datacenters and Hospitals.
  GPU Technology Conference (GTC 2014), San Jose, California, USA, March 24–27, 2014.

- N. Delbosc, J.L. Summers, A.I. Khan.
  Real-time Simulation of Air Flow in Datacenters and Hospitals.
  International Conference for Mesoscopic Methods in Engineering and Science (ICMMES 2013), Oxford, U.K., July 22–26, 2013.

- N. Delbosc, J.L. Summers, A.I. Khan.
  Real-Time Indoor Air Flow Simulation Using Lattice Boltzmann Method on GPU.
  International Conference for Mesoscopic Methods in Engineering and Science (ICMMES 2012), Taipei, Taiwan, July 23–27, 2012.

- N. Delbosc, J.L. Summers.
  Nanofluid Simulation Using the Lattice Boltzmann Method on Graphics Processing Unit.
  Workshop on Nanomaths. Centre de Recerca Matemàtica (CRM), Bellaterra, Barcelona, Spain, July 11-13, 2012.

CONTENTS

1 INTRODUCTION

1.1 BACKGROUND AND MOTIVATIONS

Simulations are at the core of all scientific modelling.

The word *simulation* comes from the Latin adjective *similis,* meaning *similar*. In this context, *simulating* a physical system can be regarded as *making something similar* to it, whether it is building a scaled experiment or solving the equations that model its behaviour. This can be done for a variety of reasons: to gain a better understanding of its functioning, to study alternative conditions, and more... but in general, the goal is to improve a particular aspect of the system. The recourse to simulations can become necessary when the real system cannot be directly accessed or modified or when its manufacturing cost would be too high.

Often, computer simulations are used to study a model, and they have become essential in the modelling of many natural systems in physics, chemistry and biology, but also human systems in economics or social science. Numerical simulations are nowadays an integral part of engineering, as the equations that describe the motion of an aircraft or the stiffness of a bridge are often too complex to be solved without the help of computers.

But despite the growth in both speed and memory of computers over the last decades, computer simulations are still regarded as a complicated task. And engineering problems commonly require large simulations that run for many hours on a dedicated cluster of computers, called a *supercomputer*. In particular, the domain of *computational fluid dynamics* (CFD, see section 1.2), that is the numerical simulation of fluid flows, has become associated with long running programs consuming a lot of computational resources. This is because CFD usually involves complex time dependant non-linear partial differential equations that are difficult to solve, even with the best computers.

The goal of this thesis is to develop new simulation tools, based on novel numerical models and novel computer architectures, that will allow for a real-time and interactive simulation of various fluid flow problems applicable to engineering, and in particular *indoor airflows*. The study, prediction and control of indoor airflows is of prime importance in engineering and sprouts to many fields of applications, from air conditioning for human comfort to thermal management for equipments (eg, data centre) and to air quality control for manufacturing processes (eg, clean room) and for environments with a high risk of contagion (eg, hospitals).

Because of excessive computation time, the majority of CFD models available today only consider steady-state scenarios which are not able to capture the transient effects (such as varying load for a data centre or the movement of people for a hospital) and are hence limited in their prediction capabilities. This work is seen as an important keystone towards the development of a tool for the real-time control of indoor ventilation systems where, ultimately, sensors and actuators placed within the flow would be synchronised with the boundary conditions of the simulation.

1.2 COMPUTATIONAL FLUID DYNAMICS (CFD)

Computational fluid dynamics, or CFD for short, is a area of research in fluid mechanics that uses numerical methods and algorithms to study fluid flows. CFD is useful both in theoretical research, to help with the development of new models, as in engineering to ease the conception of new designs, and usually at lower cost than the experiment and sometimes at a better accuracy.

The field of CFD is usually associated with the resolution of the *Navier-Stokes equation*, a partial differential equation (PDE) which expresses a local conservation law for the momentum in the system. Due to the generality of this equation, it can be applied to model a large variety of fluids, such as liquids (like water and oil) or gases (like air). Indeed, although liquids and gases are two very distinct types of fluid, their bulk behaviour can be described using the same equation. When the compressibility effects are neglected, the difference between the two fluids boils down to a single physical property: the *kinematic viscosity*, which describes the proficiency of the fluid to diffuse momentum. Viscosity is the reason why the movement of a fluid layer can induce movements in the neighbouring layers. It can be shown that the viscosity characterises the rate of energy dissipation in a fluid subject to a deformation, which is why stirring a spoon in honey is harder than in tea (honey has a higher viscosity than water, and therefore tea).

Henri Navier
(1785–1836)
Source: École des Ponts, ParisTech

Sir Georges Stokes
(1819–1903)
Source: Popular Science Monthly, vol. 7, p. 641, 1875.

The Navier-Stokes equation was named after two physicists: one French, Henri Navier and one Irish, Sir George' Stokes who studied and published (separately) studies on the law of momentum of fluids during the first half of the 19th century. Solving the Navier-Stokes equation can be a very challenging task; actually, an analytical solution is only available for simple problems of limited use. However, in total disregard for the lack of mathematical proof[1], engineers have been computing approximated numerical solutions of the equa-

1 As a pure mathematical problem, the proof of the existence and smoothness of a solution for the three-dimensional Navier-Stokes equation is still an unsolved problem. It is one of the seven *Millennium Prize Problems* by the *Clay Mathematics Institute* which offers a $1,000,000 price to the first person providing a solution.

tion for a long time (with the help of computers and special algorithms).

In its simplest incompressible form, the Navier-Stokes equation can be written as

$$\frac{\partial \vec{u}}{\partial t} + \left(\vec{u} \cdot \vec{\nabla}\right) \vec{u} = -\frac{1}{\rho} \vec{\nabla} p + \nu \vec{\nabla}^2 \vec{u} \tag{1}$$

where $\vec{u}$, p and ρ represents the velocity, pressure and density of the fluid, respectively, and ν is the kinematic viscosity of the fluid. The equation can also include additional stresses and body force terms. The velocity and pressure in the fluid are functions of space $\vec{x}$ and time t, i.e. $\vec{u} = \vec{u}\,(\vec{x}, t)$ and $p = p\,(\vec{x}, t)$, they are each represented by a field of values rather than a single value. Air is a compressible fluid, which means its density can also vary, however for indoor airflows, the variations of density are small enough to be neglected and the flow is assumed to be incompressible. For some fluids, known as *non-Newtonian*, the viscosity can also vary with shear stress; these types of fluids are significantly harder to study.

As written in (1), this equation constitutes a largely simplified model for a viscous , single component fluid in an environment without density or temperature variations. Although it only partially addresses the complexity of real fluids of interest in engineering applications, it can still lead to accurate predictions of the flow, in agreements with the results of physical experiments.

The Navier-Stokes equation alone does not form a closed system. There are 4 unknowns (u_x, u_y, u_z and p) but only 3 equations (equation (1) can be projected against each of the 3 axis). In order to close the problem, another equation needs to be considered.

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot (\rho \vec{u}) = 0 \tag{2}$$

This equation is known as the *continuity equation*. It describes the conservation of mass in the system. For an incompressible flow, the density ρ is a constant, thus the continuity equation takes a simpler form (which shows that the velocity filed of an incompressible flow is *divergence-free*, also known as *solenoidal*).

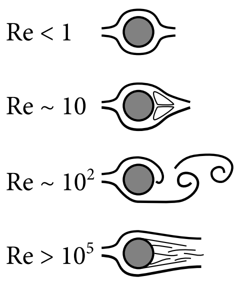
$$\vec{\nabla} \cdot \vec{u} = 0 \tag{3}$$

It should be noted that to truly close the above equations, the boundary conditions need to be addressed, more details on this are available in section 3.2.

Through a process of dimensional analysis (see Appendix B), it is common practice to re-express the physical terms in the Navier-Stokes equation using dimensionless quantities along with a characteristic lengh, velocity and time scales. This process is known as the *non-dimensionalisation* of an equation. When applied to the Navier-Stokes equation, an important dimensionless number naturally appears, the *Reynolds* number (Re).

$$\mathrm{Re} = \frac{LU}{\nu} \tag{4}$$

where L is a characteristic length for the system, for instance it could be the radius of a cylinder, U is a characteristic speed for the system, for instance the speed of the cylinder, and ν is the kinematic viscosity of the fluid as introduced previously.



Qualitative description of the fluid flow over a cylinder for a large range of Reynolds number.

There are over dimensionless quantities that are used to characterise fluid flows, but the Reynolds number is of particular importance because it describes what is the regime of the flow, such as *laminar*, *transient* or *turbulent*, and it is what justifies using a scaled model of an aeroplane during an experiment by matching the Reynolds number of the experiment and that of the aeroplane. By its construction, the Reynolds number indicates the relative importance of inertial to viscous forces, for small Reynolds number, viscous effects are predominant and the flow is laminar and for large Reynolds number, inertial effects are predominant and the flow becomes turbulent (see illustration in the margin).

1.2.1 *Numerical Methods for CFD*

For many engineering problems, an analytical solution to the Navier-Stokes equations cannot be found and it becomes necessary to resort to numerical methods. Several numerical methods can be used to obtain a numerical approximation to the Navier-Stokes equations, but at their core, most of them rely on the discretisation of the equations on a finite grid that can be solved numerically so that the value at each grid point (or cell) matches that of the "exact" solution evaluated at that point (with a certain truncation error).

The three methods most commonly used in engineering are (by order of complexity) : the finite difference, the finite volume and the finite element methods.

The *finite difference* methods are the most straightforward : the derivatives at one node are computed in terms of a truncated Taylor series expansion using the values at neighbouring nodes. The size of the stencil (i.e., the amount of neighbouring nodes) used in the computation determines the order of the truncation, and higher order schemes will converge faster (give more accurate results) as the resolution (i.e., the number of nodes in the domain) is increased. These methods are simpler to implement but they are commonly restricted to Cartesian grids which limit their application to complex geometries.

Rather than considering the nodes, the *finite volume* and *finite element* schemes are both based on dividing the flow domain into a (large) number of small cells, or volumes. Theses cells can be of any shape (tetrahedra, prisms, cubes, ...) which allow the use of unstructured meshes finely refined around the object's body.

The *finite volume* methods are based on the integration of the governing equations over each cells. This allows the divergence terms of the equations to be transformed into a surface integral using the divergence theorem.

Finite element approaches are traditionally used in solid mechanics but can be adapted to fluid problems. These methods approximates the unknown field by piecewise functions valid over each cell whose coefficients are to be determined. The residuals, that is the difference between the function and the exact solution, are minimised by multiplying them with a weighting function and integrating; which results in simple algebraic equations for the coefficients of the approximating function.

Each of these methods has several variants with different orders of convergence, stabilities, advantages and disadvantages. It is also possible to make problem specific simplifications in order to speed up the computations. For instance, if the flow is axisymetrical (invariant under any rotation around a specific axis), then it can be solved as a 2D problem with the appropriate corrections on the acceleration. Another example, if the flow is supposed to be steady, then the velocity does not vary over time, and the Navier-Stokes equation can be simplified by removing the first term. However, for time dependant flows, two different schemes can be used for the resolution: explicit and implicit schemes. In explicit schemes, the state of the system at a later time is directly computed from the state at the current time, these schemes require the use of a significantly smaller time-step in order to insure the stability. For implicit schemes, the new state is found by solving an equation involving both the current and new states, these schemes require the inversion of larger matrices and are significantly slower, but they tend to be more stable.

Two other methods are worth mentioning : the spectral method and the vorticity-streamfunction formulation whose principles are to reformulate the Navier-Stokes equation in another basis or using different variables (respectively) in order to make them easier to solve.

The spectral methods uses a transformation (usually the Fourier transform) to bring variables in a new space (called the spectral space) where derivations and integrations become simple multiplication and divisions. Once solved, the transformed solution can be converted back to the original space using the inverse transformation. The problem of such methods is in the choice of the right basis functions, which can be difficult when dealing with complex geometries. However, provided a good basis is available, spectral methods are notorious to achieve a high degree of fidelity of the solution, and can solve the Navier-Stokes equation up to machine accuracy. As a result, spectral methods are often confined to more academic problems like the study of homogeneous isotropic turbulence.

The vorticity-streamfunction formulation is particularly interesting in 2D where it allows to reduce the dimensionality of the problem by reformulating the Navier-Stokes equation in terms of two scalar quantities (the vorticity and the stream function) instead of one vector quantity (the velocity) and one scalar quantity (the pressure). However, as for spectral methods, it can be difficult to define complex boundary conditions within this context, and the range of applications of this method is thus limited.

1.2.2 *Thermal Flows*

At 0 K, i.e. −273.15 °C, all the microscopic particles are perfectly still, thus it is the smallest possible temperature. It is known as the absolute zero.

Thermal fluctuations play an important role in the dynamics of a flow. When the temperature of a volume of fluid increases, the random motion of its constituent microscopic particles increases and as a result, the volume increases hence, as the total mass is conserved, the density decreases. If this volume of hot fluid is surrounded by a colder fluid, then its density is locally lower, and this volume will rise due to *buoyancy*. In the opposite, a volume of fluid colder that its surrounding would fall because its density is relatively higher.

In a fluid flow, there are two processes by which the temperature can transported:

- *thermal advection*, i.e., *convection*, as the fluid moves it carries with it the particles of fluid and their temperature,
- *thermal diffusion*, the temperature of a fluid layer can propagate to the neighbouring layers. This process also exist in solids where it is know as *thermal conduction*.

The Péclet number (Pe) is a dimensionless number that can be used to indicate the relative importance of advection to diffusion transport rates.

$$\mathrm{Pe} = \frac{LU}{\alpha} \tag{5}$$

where L and U are still the characteristic length and speed of the system. And α is the coefficient of thermal diffusivity of the fluid.

For further details on the equations modelling thermal effects in a fluid, see section 2.3.5.

1.2.3 *Multiphase Flows*

The simulation of multiphase flows, that is a system containing different fluid phases such as liquids and gases interacting together (and also with solids), is a notoriously challenging area of CFD. Indeed, although the bulk of each phase can be individually simulated using the Navier-Stokes equation, there are also some crucial physical phenomenon happening at the interfaces, such as the *surface tension* for example, that need to be modelled carefully. The modelling of multiphase flows becomes even more challenging when temperature variations comes into play, introducing phase changes like droplet condensation and evaporation. It is worth noting that there is also a similar category of fluid flow problems involving a mixture comprised of different fluid species (e.g. water and oil) that can be either miscible or immiscible. This type of flows are known as multi-component flows. Multiple engineering problems involve multiphase and multicomponent flows and constitute active research fields. The mixing of two fluids for manufacturing, droplet impingement for inkjet printing, droplet extraction for filtering system, droplet evaporation for spray cooling, etc... are a few examples.

There are various strategies to simulate multiphase flow that aims at simplifying the problem. For instance, for large scale water-air multiphase flow, like a dam break, the surface tension effects are small and the air phase can be neglected. This type of problem is known as free-surface flows. The volume of fluid (VOF) method is a numerical technique that was designed to solve such problems. It combines the simulation of the liquid phase using the Navier-Stokes equation with an interface tracking technique to precisely solve the location of the free surface. However at small scale, like in microfluidics, the surface tension forces cannot be neglected and classical CFD techniques struggle to capture the effects of varying contact angles or phase changes. On the other hand, the lattice Boltzmann method (see 1.4) due to its kinetic nature can incorporate inter-molecular interactions in a straightforward way and is able to capture the effects effectively.

1.2.4 *Turbulence Modelling*

For high Reynolds number, the inertial effects are predominant and the flow becomes turbulent. A turbulent flow is a highly complex flow showing many irregular and random (chaotic) features and composed of many eddies of largely different sizes. The largest eddies have a size similar to the characteristic length of the domain (integral scale). These eddies obtain energy from the mean flow. By interacting together, the energy is passed down sequentially from the larger eddies gradually to the smaller ones, down to the smallest scale (Kolmogorov scale) at which

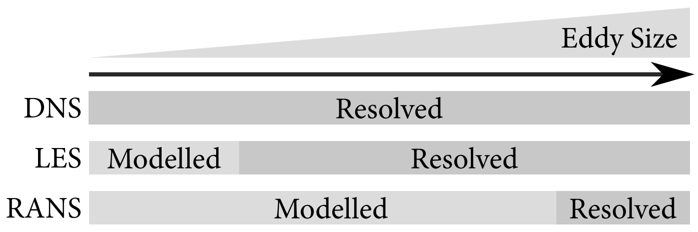


Figure 1: Each turbulence model resolves different scales.

the energy is dissipated by viscosity. This process is know as the *turbulent energy cascade*.

The variety of scales and the complexity of turbulent flows makes the modelling of turbulence an extremely challenging topic in CFD. Although there are many approaches to the numerical simulation of turbulence, they can be classified into three categories, each solving the flow down to a different scale (see figure 1).

DIRECT NUMERICAL SIMULATION (DNS) :
: the DNS solves the Navier-Stokes equation without any specific model for turbulence, which means the whole range of eddies scales needs be resolved and the spatial resolution in DNS must be of the same order as the Kolmogorov scale. Turbulence theory shows that this scale is around $\mathrm{Re}^{\frac{3}{4}}$ smaller than the integral length scale, thus the number of cells required in three-dimensional DNS increases with the Reynolds number as $\mathcal{O}(\mathrm{Re}^{9/4})$. Therefore, the computational cost of DNS is tremendous for high Reynolds number flows. Nevertheless, the computational power of today's supercomputers allows to realise the DNS simulation of turbulent flows up to $\mathrm{Re} = 10^4 - 10^5$ using billions (i.e. 10^9) of cells [1, 2].

LARGE EDDY SIMULATION (LES) :
: the view of LES is that large eddies are usually more energetic and contribute more to the transport than the smaller ones, so it sufficient to only solve accurately for the large scale of the flow and the effect of the unresolved small scales on the fluid motion is modelled using a sub-grid model. In practice, simple turbulence models, like the Smagorinsky model (see section 2.5.1), suppose that the energy dissipation by the small scale is isotropic and can be represented using a local eddy viscosity which is added to the fluid kinematic viscosity. LES simulations are still expensive, but not as much as DNS and can give higher accuracy than RANS, especially when the transient behaviour of the flow is of importance.

REYNOLDS-AVERAGED NAVIER STOKES (RANS) :
: the RANS approach is to only simulate the mean flow averaged in time and to model the fluctuations using additional transport equations. It is the most computationally efficient way to simulate turbulent flows in engineering applications. However, a single model cannot address all the complexity of turbulence, and RANS simulation should be regarded as an engineering approximation of a complex turbulent flow. Amongst the various RANS models, the $k - \epsilon$ model is is one of the most popular in engineering. It introduces 2 additional transport equations to model the turbulent kinetic energy k and the turbulent energy dissipation ϵ.

1.2.5 *Commercial CFD Software.*

A introduction to CFD could not be complete without citing some of the most used CFD software in engineering. The three following software packages are used routinely by many engineers at the *School of Mechanical Engineering* at the *University of Leeds*, and can be taught in classes. They have been used to give comparable simulation results throughout the Ph.D.

OPEN FOAM : Open Foam[2] is a free, open source CFD software package developed by a commercial company, OpenCFD Ltd, now a subsidiary of the ESI group. It is based on the finite volume method and due to its openness, it allows its user to customise their own solver for an extensive range of problems from complex fluid involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetism.

FLUENT : with its first release in 1988, Fluent is one of the longest running commercial CFD software. It was acquired in 2006 by the company ANSYS, Inc. and is now part of ANSYS product line[3]. The software is based on the finite volume method and benefits of a simple graphical user interface and integrated pre and post processing tools. It includes a large variety of models and dynamics that allows to simulate a large range of fluid problems but is not as flexible as its competitors and access to the details of the working of the solver is not possible.

COMSOL MULTIPHYSICS : as its name suggest, COMSOL Multiphysics[4] strength lies in its capability for coupled physics phenomena modelling. While a large variety of phenomena are predefined through equation-based model templates, it allows the user to modify them with their own equations for their specific multiphysics model. It also provides a modern graphical interface with integrated pre and post processing tools.

Some CFD software choose to specialise to solve a specific type of problem. For instance, software like 6Sigma[5] and floVENT[6] are dedicated to the simulation of HVAC (short for Heating, Ventilation and Air Conditioning) and have been optimised to do it simply and efficiently.

2 http://www.openfoam.com/
3 http://www.ansys.com/Products
4 http://www.comsol.com/comsol-multiphysics
5 http://www.futurefacilities.com/software/6SigmaDCOverview.php
6 http://www.mentor.com/products/mechanical/flovent/

Real-time fluid simulation, i.e., computing the movement of a virtual fluid as fast as the real physical fluid would evolve, is an active field of research in Computer Graphics. While the numerical methods introduced in the previous chapter can give very accurate solution to the Navier-Stokes equations, they usually take a very long time to converge and are not amenable to real-time simulation. On the other hand, computer graphics and in particular computer games, have been including in the recent years more and more real-time physical simulations[7]. Some of these games showcase simple real-time fluid simulations, but they are more focused on creating visually appealing animations rather than aiming for physical accuracy. Nevertheless, it is important to understand if these techniques could be useful to engineering-type CFD and what the capability for real-time simulation could bring to the field.

First and foremost, a real-time capability would give a completely new way to approach CFD problems. Nowadays a simulation with a typical CFD software is comprised of three key steps: (1) pre processing, (2) numerical simulation of the flow and (3) post-processing.

1. During the *pre-processing* step, the engineer defines the characteristic of the flow problem to be simulated by the solver in the next step. This is when the solution domain is defined, the physical properties of the fluid are selected, the boundary conditions are specified and more importantly when the mesh required by the solver is generated. In itself, pre-processing can be a challenging task, and a wrong choice of any of the aforementioned parameters might cause the solver to not properly converge.

2. The *simulation* is usually the longest step. The solver takes all the parameters, boundary conditions and mesh from the preprocessor and simulate the flow using the selected equations and numerical method. It can range from a few minutes to several days for complex high resolution engineering applications.

3. Finally, the results of the solver are analysed during the *post-processing* step. By its nature, CFD generates a large amount of data which requires specialised tool to dissect. A post-processor is basically a visualisation tool that can display informations in various forms such as vector, contour and surface plots but also compute additional flow informations that are not necessarily know to the solver such as vorticity, streamlines and streaklines. The post-processor generates images and animation and it is the role of the engineer to interpret the results of the simulation from them.

7 http://www.geforce.com/hardware/technology/physx/games

Each of these steps can easily take from a few minutes to a few hours depending on the application, but in a real-time context these steps need to happen several times per second. The geometry can be updated as the solver is running and the boundary conditions or even the fluid properties can be modified on the fly. A real-time visualisation need to be connected to this process to inform the user on the changes that he is performing and allow him to inspect the resulting flow structures as they are being simulated. Typically, to avoid jittering and provide a smooth experience, a real-time fluid simulation need to be iterated at a minimum of 25 times per second, or frame per second (FPS) as for a film. Therefore, each simulation-step must take a maximum of 40 milliseconds to compute. The time constraints for real-time CFD are extremely tight. The Figure 2 provides a representation of the design cycles using traditional CFD techniques against real-time CFD. It should be noted that the computing time figures are purely illustrative and would vary heavily depending on the application.

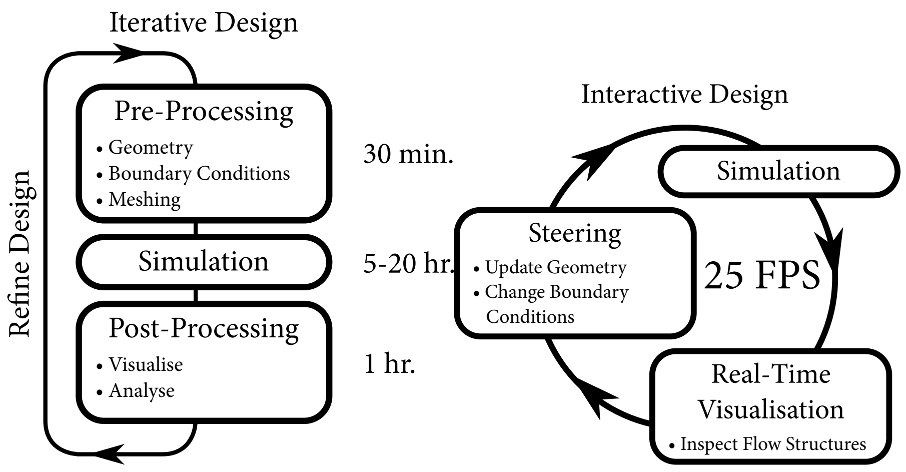
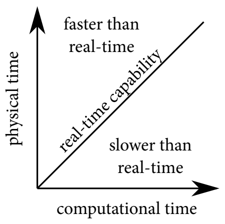


Figure 2: Design cycles using traditional CFD versus real-time CFD. The computational times provided are merely estimations for a simple indoor air flow simulation, and in practice they largely vary depending on the complexity of the simulation.

Real-time CFD and even "faster than real-time" CFD (the top triangle on the figure in the margin) is necessary to inform early warning systems. An example of such system, which is also amongst the only one to use predictive CFD, is weather forecasting. Weather forecasting relies on sensors for temperature, pressure and velocity spread across the surface of the country (but also in the air). These sensors are used to initialise the simulation, then the flow is solved on large clusters of computers, several times per day for both short-term and long-term forecasts. While the numerical methods used in weather forecasting vary depending on the country, in the UK the Met Office predictions come from a *Unified Model* which couples different simulations running on different grids at different resolutions (and poten-



Physical time of a problem versus the computational time required to solve it.

tially with different numerical schemes). At its core, the solver uses a semi-implicit semi-lagrangian discretisation of the fully compressible Navier-Stokes equation on a rotating sphere, and meanwhile several additional processes operate to warm (or cool) and moisten (or dry) the atmosphere, to form clouds and precipitation.

Another area that has been hinted at and where real-time CFD would really help is in design optimisation. CFD is very often used by engineers in order to optimise one or several parameters of a model, from optimising the lift of an aerofoil to finding the optimal location for the ventilation vents in a room. Optimisation in CFD is very challenging because it requires to compute multiple simulations of the same problem, where the parameters to optimise are varied; and then, using clever techniques, guess the optimum based on interpolation of the limited set of simulations. There is an entire field of CFD devoted to that. Using real-time CFD, an engineer could vary the parameters to optimise by hand while monitoring the quantities of interest and quickly reach the optimal value for his problem. This process of modifying the simulation while it is running is known as *computational steering*.

Finally real-time CFD could bring a lot of benefits to teaching. Allowing the teacher to illustrate abstract concepts of fluid mechanics using real-time interactive simulations of real flows.

Maybe paradoxically, real-time simulation is easier to achieve for larger systems than for small ones. Indeed, while it is easy to solve planetary rotations due to gravity in real-time (one year to solve one rotation of the planet Earth!) it would be much harder (impossible) to solve microscopic particle collisions in real-time as they happen on very short time scales. The same is true with fluids, while it is possible (yet difficult) to simulate airflows around the globe in real-time, it would be harder to simulate a micro droplet impacting a surface in real-time. It is unlikely that real-time will ever be possible for such small time scales, and even if was it would not be very useful as the user would have to slow down the simulation to inspect the results. For such cases, it is more convenient to consider *interactive CFD* instead which still yields many of the benefits detailed above.

In 1965, the engineer Gordon Moore predicted that the doubling of component density on an integrated circuit (and thus the performance) every two years. Fifty years later, the law still holds, but its end might be near as components size approach that of the atom. Photo: Intel.

Nowadays, computers are very powerful machines and a single top-end desktop computer can easily achieve several teraFLOPS, that is several millions of millions of floating-point calculations per second. And yet, this is still insufficient to allow real-time simulation using traditional CFD methods (finite difference, volume, element), although it might become possible one day due to Moore's law. In the meantime, it is necessary to come up with alternative methods, and maybe trade off some of the physical accuracy for higher computational speed. The gaming industry and the computer graphics community do not prioritise the physical accuracy, instead they use special numerical methods (usually wave-based or particle-based) that looks convincing but do not resolve the Navier-Stokes equation. Although, in some cases, the semi-lagrangian scheme, which crudely approximate this equation, is used.

The most common way of faking a fluid in video games uses wave functions to apply oscillations on a surface, and create the illusion of a moving flow, but this hardly qualifies as a simulation. The second most commonly used technique for large water bodies solves for the *shallow-water equation*. When solely the surface of the water is of interest and the heights of the fluctuations are small compared the size of the water pool, the Navier-Stokes equation can be simplified by integrating along the depth. This technique allows to simulate convincingly waves propagating on the surface of the water and video games might use it to simulate waves on a pool around a character's body, for instance. However, it is not amenable to indoor airflows.

Another way of approximating free-surface flows is using particles like in the *smooth particle hydrodynamics* (SPH) method. That technique has gained popularity recently in video-games, despite its high computational cost, and NVIDIA, a large video-game company, recently developed a tool[8] for real-time simulation using particles to represent both fluids, solids and soft bodies. The idea of SPH is to represent the fluid volume with particles that are advected using the simple Newton's law and the particles are subjects to an interaction potential that tries to mimic surface tension. The technique is quite good at representing free surface flows like a dam breaking and its advantage is that it does not require a mesh (it is a meshless technique) as the particles are allowed to move freely until they hit an obstacle. On the other hand, it does require a large number of particles to accurately represent the flow and boundary conditions such as inlet/outlets are difficult to implement as particles have to enter/exit the flow domain. It is therefore not amenable to indoor airflows either.

Finally, the *fast fluid method*, also known as *semi-lagrangian*, is a grid-based method that relies on the Navier-Stokes equation where the advection term is approximated by guessing where the velocity of each cell is coming from, following the flow trajectory and using an interpolation scheme. The method is very interesting due to its good computational efficiency and unconditional stability, as showcased by its successful use in weather forecasting. However, it is relatively inaccurate in its standard form but there are several possible modifications that aims at improving its accuracy (although they require to loose some of its simplicity, efficiency and stability). More details on the method will be given in Appendix C.

Another contender for real-time CFD, although it is very rarely used in computer graphics, is the *lattice Boltzmann method* (LBM). As a numerical method, the LBM can be regarded as an explicit scheme

8 https://developer.nvidia.com/physx-flex

second order in space and first order in time. Its explicit nature makes the method very computationally efficient, but its real particularity is that it does not solve directly for the Navier-Stokes equation or for an approximated version of it. Instead, the LBM solves for a discrete version of the Boltzmann equation, an equation coming from the kinetic theory of gases. The LBM shows several advantages : it is simple, fast (especially on parallel processors, see chapter 4), stable and it can accurately solve for thermal and turbulent air-flows making it an ideal candidate as the preferred numerical scheme in the real-time simulation of indoor air-flow for this Ph.D. As a result, after introducing the method in the next section, chapter 2 will give a detailed review of the method, including some of its many variants, chapter 3 identifies various boundary conditions and chapter 4 will discuss its optimised implementation on a Graphics Processing Unit.

1.4 LATTICE BOLTZMANN METHOD (LBM)

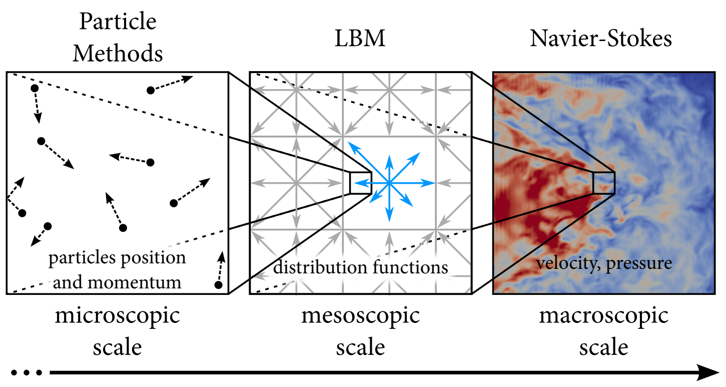


Figure 3: A fluid flow can be studied from various scales.

Three length scales (or time scales) can be defined for a fluid flow : the microscopic scale, the macroscopic scale and between the two an intermediary scale named : *mesoscopic scale*. These scales are represented in figure 3.

MICROSCOPIC SCALE This is the scale of the molecules of fluid. At this scale, particles have ballistic trajectories (Brownian motion) with an average microscopic speed given by the temperature. This is the scale that molecular dynamics and smooth particle hydrodynamics try to replicate to some extent.

MESOSCOPIC SCALE When the number of particles becomes large, it becomes more convenient to average the particles over a volume. The kinetic theory studies the evolution of particle's *distribution function*. Distribution functions exist in phase space, for instance $f(\vec{x}, \vec{u}, t)$ represents the number of particles per unit volume having the velocity $\vec{u}$ within the volume surrounding the location $\vec{x}$ and at time t. It is this mesoscopic viewpoint that is taken by the LBM.

MACROSCOPIC SCALE This is the scale of the variations of vectors fields (like the fluid speed $\vec{u}$) and scalar fields (like the pressure p or the temperature T). This scale is large enough compared to the microscopic and the mesoscopic scale that the fluid can be considered as a continuum substance, so that macroscopic quantities are defined on every point $\vec{x}$ and every time t. For instance, the speed can be written as $\vec{u}\,(\vec{x}, t)$ and the pressure as $p\,(\vec{x}, t)$. The behaviour of these macroscopic fields is described accurately by the Navier-Stokes equation.

While it might seem strange that the LBM, focused on particle's distribution function, can be used to simulate macroscopic flows, such as indoor airflows, that are normally described by the Navier-Stokes equation, the two methods do in fact produce the same results, and with a similar accuracy, see chapters 6 and 7. Actually, it has been shown that the Navier-Stokes equation can be derived from the lattice Boltzmann equation through a low Mach number expansion known as the Chapman–Enskog procedure [3].

It is important to note that the LBM is not a single method but rather a category or group of methods that can be used to solve diverse partial differential equations (not just the incompressible Navier-Stokes). This can make LBM confusing to the beginner, as it can be difficult to choose which variant of the method to use for which problem. On the other hand, it is this flexibility that make the LBM so powerful allowing to simulate the incompressible Navier-Stokes equation for isothermal single phase flow but also other problems like multiphase flows using an interaction potential or completely different problems such as solving the light transport equation [4, 5] or the Schrödinger equation for quantum dynamics [6].

One of the strong point of LBM compared to traditional CFD (based on Navier-Stokes) is that it does not require to solve a Poisson equation for the pressure. This allows the LBM to be intrinsically local, i.e., each node can be computed independently from its neighbours, which in turn results in very high performance when implemented on a massively parallel processor like a GPU (see next section). A downside though, is that it comes at the price of a generally less accurate pressure field and small variations in density that can cause pressure waves. In this regard, the LBM is an artificially compressible scheme to solve the incompressible Navier-Stokes equation, rather like the *artificial compressibility method* (ACM) [7].

On the other hand, one of the downsides of LBM is its *youth* compared to most other numerical methods in CFD. A direct consequence of that is that the LBM still requires some development work (for instance in the modelling of the turbulence) as well as a serious work of *classification* and *standardisation*. Notably on the implementation of the boundary conditions or on the process of unit conversion which is too often overlooked (that is the process of transforming the macroscopic variables defining a problem into an initial state and boundary conditions for the distribution functions used in the simulation).

The LBM is a very active field of research, and as such, there are two yearly international conferences dedicated to it, hosted by different universities each year. The Discrete Simulation in Fluid Dynamics[9] (DSFD), which is more focused on the theoretical development of the method. And the International Conference for Mesoscopic Methods in Engineering and Science[10] (ICMMES) which features relatively more engineering-focused applications. In the United Kingdom, the UK Consortium on Mesoscale Engineering Science[11] (UKCOMES) aims to bring together the expertise from multiple universities in the UK and to maintain DL_MESO[12], a simulation package based on LBM (free for academics).

Commercial LBM software

Due to its originality and youth, the LBM is not yet very popular amongst CFD engineers, as a result there are only a few commercial CFD software based on LBM.

ULTRAFLUIDX [13] : developed by the German company FluiDyna GmbH which also develops an SPH software, a finite volume software and a library for GPU acceleration of OpenFOAM. The company is investing in GPU computing and all its software feature GPU acceleration, allowing large performance increase. The ultraFluidX software is mostly dedicated to the simulation of airflows around moving vehicles, it features turbulence modelling, a mesh refinement technique and it allows for in-situ visualisation.

POWERFLOW [14] : developed by the American company Exa Corporation and also focuses on the simulation of airflows around vehicles. The company product line include a tool to study the noise created by the interaction of the air and the moving vehicle and a tool for thermal management as well as several dedicated software for pre or post processing.

XFLOW [15] : developed by the Spanish company Next Limit S.L which also develops fluid simulation software for the graphics indus-

9 http://www.dsfd2015.ed.ac.uk/home
10 https://www.icmmes.org
11 http://www.ukcomes.org/
12 http://www.stfc.ac.uk/SCD/support/40694.aspx
13 http://www.fluidyna.com/content/ultrafluidx
14 https://www.exa.com/powerflow.html
15 http://xflowcfd.com/

try and a physically based image renderer for architecture applications. The solver is based on the factorised central moment LBM and features an LES turbulence model (with special wall modelling). It allows moving boundaries with adaptive wake refinement and can simulate multiple fluid physics such as compressibility, thermal fluctuations, turbulence, multiphase flows, etc...

PALABOS [16] : Palabos is an open-source solver developed by the a commercial company FlowKit Ltd. from Switzerland. It has a reasonably large community from academia with an active forum[17] and a wiki[18] full of useful information for beginners looking to get started with LBM. The source code is written in the C++ programming language and relies heavily on templates for compile time optimisation. It features a large variety of models such as multi-phase/free-surface flow, thermal flow, fluid-particle interaction and immersed moving object. It uses a multi-block technique for mesh refinement and can scale well with the number of cores, however it does not feature GPU acceleration.

Due to the simplicity of its algorithm LBM also spawned a large amount of open-source implementations, usually developed by a single person and thus not very well maintained and limited in reach. Nonetheless, the Sailfish[19] project is particularly interesting as it uses Python (a scripting language) to maintain the ease of use and generates an compilable code (C or CUDA) to achieve high performances, and it works on GPU. A similar approach was used for the development of the simulation code for this Ph.D.

1.5 GRAPHICS PROCESSING UNIT (GPU)

A graphics card, also known as graphics processing unit, or GPU for short, is a recent type of processor that focuses on parallel computing. Traditionally, this type of computing device was found in video game consoles and in gaming PC to accelerate the graphics computations. Their functioning is significantly different to that of a central processing unit (CPU) which handles of all of a computer's processes. While the CPU has to handle a large variety of tasks in a iterative way, the GPU operations are more specialised and done in parallel so that every pixels of the display is computed at the same time, for instance. Over the last fifteen years, the architecture of the GPU has evolved independently from that of the CPU and they are nowadays

The GTX Titan X from NVIDIA. With 6 teraFLOPS of compute power, it is amongst the fastest computer chip to date. Photo: NVIDIA.

16 http://www.palabos.org/
17 http://www.palabos.org/forum/
18 http://wiki.palabos.org/
19 http://sailfish.us.edu.pl/

much more powerful than their CPU counterpart. Both in term of raw compute power, as they pack a massive amount of cores (a few thousands versus a dozen for a CPU), but also in term of memory bandwidth, that is the speed at which data can be accessed, using their own dedicated memory. They can come in multiple forms or shape, size, performance and price, but overall they are more interesting than CPUs in term of performance per watt (power usage) and performance per price.

On the other hand, a program needs to be written carefully in a special programming language in order to run efficiently on a GPU, it is not possible to just run a program compiled for a CPU. Even so, the LBM can be implemented very efficiently on the GPU (as described in chapter 4), achieving a performance of more than a billion cell updates per second for a 3D problem, opening the door to the simulation of complex indoor airflows in real-time.

1.6 OBJECTIVES OF THE THESIS

This thesis is in line with two research themes from two different institutes at the University of Leeds. Namely the Institute of Thermofluids in the school of Mechanical Engineering looking at the thermal management of data centres and the Institute for Public Health and Environmental Engineering with an interest on the control and the optimisation of hospital ventilation.

The goal of this project is to develop a real-time capability for the simulation of thermal and turbulent indoor air flows, running on a single (high-end) desktop computer.

To this purpose, the lattice Boltzmann method is used as the main numerical method for the solver, as it is more efficient than methods that discretise the Navier-Stokes equation, while being as accurate.

Moreover the solver is implemented and optimised on a GPU, allowing a computational speed increase of about two orders of magnitude compare the same implementation on the CPU.

The software needs to be able to handle the complex geometries of indoor environment as well as being able to accurately model the thermal and turbulent fluid behaviours while maintaining the real-time capability. In addition, it needs to provide a facility for an easy definition of the fluid flow problem with its geometry and appropriate boundary conditions together with an in-situ visualisation tool to allow for the real-time study of the flow as it is simulated and to allow for computational steering from user inputs.

The finished software could have multiple applications in engineering but two applications are of particular interest. (1) The prediction and control of thermal loads in *data centres* (see section 7.3) as well as the optimisation of the ventilation system design in order to reduce the cost of the cooling. The tool could be combined one day with sensors and actuators located inside a data centre to allow for dynamic adaptive cooling based on the servers load and the temperature profile across the room. (2) The simulation of the ventilation in *hospital* wards (see section 7.4) in order to improve the thermal comfort of the patients and predicting the transport of pollutants (such as dust, bacteria and viruses) in the hope of creating one day smart hospitals that would be able to respond quickly to the spread of a contamination.

1.7 THESIS OUTLINE

Chapter 2 of the thesis provides an extensive description of the field of LBM in its current state, with details on multi-physics modelling (multi-phase or thermal coupling), on improved collision operators for increased stability, on turbulence modelling and on techniques to approach non-uniform grids.

Chapter 3 then extend the description with an important discussion on the issues of initial and boundary conditions that any LBM implementation must deal with.

Chapter 4 introduces the graphics processing unit and the concepts of parallel programming. It describes the implementation of each LBM model on the GPU and their optimisation in order to achieve the highest computational throughput. This implementation is then tested in chapter 5 for its performances and chapter 6 for its accuracy against various benchmark problems.

Chapter 7 applies the described software and algorithms to the simulation of indoor airflows by looking at a benchmark room for the ventilation and at a the modelling of a small data centre.

Finally, chapter 8 showcases how the program has the potential to be applied to a wider range of applications and presents some results on multiphase flows, on fluid-structure interactions and on the parameter search for enhancing the stability in some moment-based models.

2 THE LATTICE BOLTZMANN METHOD

2.1 INTRODUCTION

This chapter delves into the lattice Boltzmann method (LBM), its origins as a cellular automaton, its foundations in kinetic theory, and the general framework of the method; more details on the algorithm and boundary conditions will be given in the next chapter. The current chapter is an attempt at a comprehensive classification of the large variety of models developed over the years for multi-physics applications. It includes a presentation of the alternative collision operators designed to improve the stability and accuracy over the standard LBM, as well as a discussion on some of the techniques for turbulence modelling and mesh refinement, both important topics for indoor air flows modelling.

2.1.1 *Historical background*

The Boltzmann equation (see details in section 2.1.3) describes gas behaviour through the statistical distribution of fluid particles. In 1973, Hardy, Pomeau, and de Pazzis [8] had the idea to render both velocities and locations of the particles discrete by using a cellular automaton, known nowadays as the *lattice gas automaton*. Later, in 1986, Frisch, Hasslacher and Pomeau [10] provided the first lattice gas automaton that was able to recover the Navier-Stokes equation. They showed that when the collision rules conserve mass and momentum, and if the underlying lattice has a sufficient symmetry (at least hexagonal in two dimensions) then the lattice gas automaton can lead to the Navier-Stokes equation in the macroscopic limit [11].

A cellular automaton, like the popular Conway's Game of Life *[9], consist of a regular grid of* cells, *each which a finite number of* states. *Complex behaviours can emerge from the simple updating rules.*

In order to fix the issue of the high statistical noise with the lattice Boltzmann automaton (i.e. one of its main drawbacks), a new model was first introduced by McNamara and Zanetti in 1988 [12]: the Lattice Boltzmann Equation. The basic idea is simple: replace the Boolean variables (corresponding to a single Boolean molecule) of the cellular automaton by floating point variables (corresponding to the distribution function of molecules). Then in 1989, Higuera et al. [13, 14] showed that using a linearised form of the collision operator increases significantly the accessible Reynolds numbers.

While by then, the core of what would become the *lattice Boltzmann method* (LBM) was founded, a very significant progress has been made over the last 25 years, both in term of the theory, applications and performances. While the lattice gas automaton started as an empirical model to replicate fluid flow behaviour, the theoretical foundations of the LBM are now sound : it is now clear in which limits the Navier-Stokes equations are recovered [15, 16, 17] and the role of the artificial compressibility is better understood [7]. Many models have been proposed [18, 19, 20, 21] to increase the accuracy and numerical stability of the LBM over the original implementation. This has somehow fragmented the LBM into many different models, as section 2.4 tries to convey, so much that it seems necessary to address the LBM as the lattice Boltzmann method*s* (with a plural). These LBM(s) have been employed in many areas with success, such as flows in porous media [22, 23], turbulent flows [24, 25, 26, 27, 28], multiphase flows [29], thermally driven flows [30, 31, 32, 33], compressible flows [34], electronic kinetic flows [35, 36], magnetohydrodynamics [37], non-Newtonian flows [38], soft matter systems [39], shallow water flows [40], reaction and combustion [41], radiation heat transfer [42], relativistic hydrodynamics [43], quantum mechanics [6] and so on.

The last twenty years have seen a massive improvements on the attainable computational throughput as the machines, algorithms, and implementations have evolved [44, 21, 45]. In the recent years, the rise of the graphics processing unit as a general purpose computing platform has opened the road to accurate fluid simulation in real-time and in an interactive way [46].

Despite the scepticism at its beginnings (and some that still persists today), the LBM is now accepted by the scientific community as a viable way to simulate fluid flows, although its adoption by the engineering community is rather limited, mostly due to the lack of established commercial software. Its numerical stability, physical accuracy, adaptivity to multiple problems and its numerical efficiency on multi-core architecture makes it the ideal candidate for the future generation of fluid flow solvers [47].

2.1.2 *Kinetic Theory*

Kinetic theory is the branch of statistical mechanics dealing with the dynamics of non-equilibrium processes and their relaxation to thermodynamic equilibrium. At its foundation, the theory is based on the *molecular hypothesis* of matter, which postulates that matter is not continuous but is composed of a large (but finite) number of small bodies, called *molecules*. The theory explains macroscopic properties of gases, such as pressure, temperature, viscosity, thermal conductivity, etc., by considering the microscopic motion of the constituent molecules.

In ordinary mechanics, the aim is usually to determine the events that follow from prescribed initial conditions. This approach is not viable for the kinetic theory of gas because (i) the detailed state of motion of every molecules at an initial instant is unknown and (ii) the sheer number of molecules is too large (in the order of the Avogadro constant) to allow to follow the subsequent motion of all the many molecules that compose a gas. Hence kinetic theory does not even attempt to consider the fate of individual molecules, but instead is only interested in their statistical properties, such as the mean number, momentum or energy of the molecules within an element of volume, averaged over a short time interval. This point of view is necessary because of the computational limitations but it is also physically adequate, as experiments only measure such averaged properties.

The Avogrado constant, that is the number of molecules in one mole, is of the order of 10^{23} which largely exceeds the capacity of even the largest super-computers in the world (around 10^{16} flops).

Hence, in statistical mechanics a gas is described by the distribution function of its molecules $f(\vec{x}, \vec{u}, t)$ which represents the probability of finding a molecule at the position $\vec{x}$, with the velocity $\vec{u}$, at the time t. Macroscopic laws of the evolution of gases can be predicted by the molecular description of kinetic theory and one its main success was to predict the second law of thermodynamics, one of the most fundamental laws of nature which states that the entropy of a closed system is always increasing.

2.1.3 *The Boltzmann Equation*

The Boltzmann Equation is the cornerstone of kinetic theory, it describes the evolution of the distribution function $f(\vec{x}, \vec{u}, t)$ in terms of micro-dynamic interactions. This equation was derived in 1872 by the Austrian scientist Ludwig Boltzmann. And although this equation was established more than a century ago, the formal mathematical proof of global existence and rapid decay to equilibrium of classical solutions was only achieved recently, in 2011 [48].

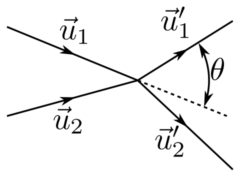
Ludwig Eduard Boltzmann (1844–1906) was an Austrian physicist whose greatest achievement was in the development of statistical mechanics, which explains and predicts how the properties of atoms determine the visible properties of matter. Image source: The Dibner Library Portrait Collection - Smithsonian Institution Libraries.

The Boltzmann Equation for the one-body distribution function $f(\vec{x}, \vec{u}, t)$ can be written as:

$$\left(\partial_t + \vec{u}.\nabla_{\vec{x}} + \vec{F}.\nabla_{\vec{p}}\right) f(\vec{x}, \vec{u}, t) = C_{12} \tag{6}$$

where the left-hand side represents the streaming motion of the molecules along the trajectories associated with the force field $\vec{F}$ (a straight line if $\vec{F} = \vec{0}$) and C_{12} represents the effect of (two-body) intermolecular collisions taking molecules in and out of the streaming trajectory. It is assumed that encounters with other molecules occupy a small part of the lifetime of a molecule, this implies that only binary encounters are important.

The term C_{12} represents the collision integral of two particles, hence the name *collision operator*; it acts only on the velocity $\vec{u}$ and is local in the position $\vec{x}$ and time t. While the Boltzmann collision operator can take various forms, as the Boltzmann Equation is applicable to a large variety of systems, for a dilute gas of point-like and structure-less molecules interacting via a short-range two-body potential, it can be expressed as:



A pair of particles before and after collision.

$$C_{12}\left(\vec{u}_1\right) = \int_{R^3}\int_{S^2} K\left(\left|\vec{u}_1 - \vec{u}_2\right|, \theta\right)\left(f\left(\vec{u}_1'\right) f\left(\vec{u}_2'\right) - f\left(\vec{u}_1\right) f\left(\vec{u}_2\right)\right) d\Omega d\vec{u}_2 \tag{7}$$

where $\vec{u}_1$, $\vec{u}_2$ and $\vec{u}_1'$, $\vec{u}_2'$ are the velocities of a pair of particles before and after collision. The Boltzmann collision kernel $K\left(\left|\vec{u}_1 - \vec{u}_2\right|, \theta\right)$ only depends on the relative velocity $\left|\vec{u}_1 - \vec{u}_2\right|$, and the deviation angle θ.

2.1.4 *The BGK Approximation*

In order to simplify the Boltzmann equation (with the goal of finding numerical or analytical solutions) without spoiling the physics, Bhatnagar, Gross and Krook introduced in 1954 [49] a single relaxation time operator to replace the complex non linear integral collision operator. They stated that the distribution function $f\left(\vec{x}, \vec{u}, t\right)$ is close to a *local* equilibrium distribution function $f^{(eq)}\left(\vec{x}, \vec{u}, t\right)$ and that it decays towards this equilibrium with some characteristic time τ.

The notion of locality *is very important to the method, as it allows for an efficient parallel implementation (see Chapter 4 for more details)*

$$C_{BGK}\left(f\left(\vec{x}, \vec{u}, t\right)\right) = -\frac{f\left(\vec{x}, \vec{u}, t\right) - f^{(eq)}\left(\vec{x}, \vec{u}, t\right)}{\tau} \tag{8}$$

The lattice BGK equation drastically simplifies the collision operator by assuming a single value for the relaxation scale τ, which should be in principle a functional of $f\left(\vec{x}, \vec{u}, t\right)$. Some models opt instead for a collision involving multiple relaxation times (see next section), but due to its simplicity, the lattice BGK equation has become the most popular lattice Boltzmann model. While simplified, collision operator in (8) retains some generality, as the local equilibrium $f^{(eq)}\left(\vec{x}, \vec{u}, t\right)$ is not linear, and can take many forms. In practice, the lattice BGK equation can be used to simulate fluid dynamics (ie. the Navier-Stokes equation), while equation (7) is only valid for dilute gases.

In the simulations, the relaxation time (τ) acts like a kinematic viscosity (ν) . When it is reduced, the distribution relaxes faster toward the equilibrium and the fluid evolves quicker, this is characteristic of high Reynolds number flows, i.e. low viscosity. Hence, it is sensible that the viscosity must vary like the relaxation time. It can be shown through a Chapman-Enskog expansion [3] that on a regular lattice with spacing Δx and constant time-step Δt, there is indeed a linear relation between them:

$$\nu = \frac{\Delta x^2}{3\Delta t}\left(\frac{\tau}{\Delta t} - \frac{1}{2}\right). \tag{9}$$

It is common practice to consider instead the relation between the dimensionless quantities, i.e. $\nu = (\tau - 1/2)/3$. Hence, the dimensionless relaxation time τ must always be higher than $1/2$ in order for the viscosity to stay positive, and numerical instabilities can arise as τ approaches $1/2$ (i.e. inviscid flows).

The above relation naturally comes out of the Chapman-Enskog expansion in which the Navier-Stokes equations (with an additional compressible term) are recovered from the lattice BGK equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0, \tag{10}$$

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot (\rho \vec{u} \vec{u}) = -\nabla p + \nu \left(\nabla^2 (\rho \vec{u}) + \nabla (\nabla \cdot (\rho \vec{u})) \right), \tag{11}$$

with an error proportional to $\mathcal{O}\left(Ma^3\right)$ in space [3] and proportional to $\mathcal{O}(Ma \cdot dt)$ in time [50], where $Ma = u/c_s$ is the Mach number of the system, $p = c_s^2 \rho$ is the pressure, $c_s = \Delta x/(\sqrt{3}\Delta t)$ is the speed of sound.

2.1.5 *Multiple Relaxation Times*

The *multiple-relaxation-time* (MRT) lattice Boltzmann equation (sometimes referred to as the *generalised* LBE) was introduced in 1992 by d'Humières [51] to overcome some of defects of the BGK model, and it has been shown to be superior in terms of accuracy and stability [20]. The principle of MRT is to perform the collision in the space of moments rather than the usual space of distribution functions, this introduces new relaxation times (one for each independent moment) that can be adjusted to tune both stability and accuracy.

The collision equation for the MRT model can be expressed as

$$C_{MRT}(\boldsymbol{f}) = \boldsymbol{M}^{-1} \cdot \boldsymbol{S} \cdot \left(\boldsymbol{m} - \boldsymbol{m}^{(eq)} \right) \tag{12}$$

where $\boldsymbol{m}$ and $\boldsymbol{m}^{(eq)}$ represent respectively the velocity moments of the distribution functions $\boldsymbol{f}$ and their equilibria, $\boldsymbol{M}$ is a $q \times q$ matrix which linearly transforms the distributions functions $\boldsymbol{f}$ to the velocity moments $\boldsymbol{m}$, $\boldsymbol{m} = \boldsymbol{M} \cdot \boldsymbol{f}$, and $\boldsymbol{S}$ is a $q \times q$ diagonal relaxation matrix, where q is the number of discrete lattice velocities (see section 2.2.1).

Although the MRT model was introduced around the same period as the BGK model, the latter has become the most popular lattice Boltzmann model in spite of its well-known deficiencies. Although the MRT overcomes some defect of the BGK, it comes at a price : on top of the higher complexity, the choice for the high order moments in not unique and depends on the lattice considered [20], and the values for the free relaxations rates needs to be optimised for each problem (an optimal choice suiting any simulation task is unknown, see section 8.5). Moreover, taking moments in a static frame of reference and relaxing them at different rate can introduce violations of Galilean invariance that are not present in the BGK model (see section 2.4.2).

In a simplification attempt to allow for a better control of the effect of the relaxation rates on the overall dynamics, a two-relaxation times (TRT) model was introduced by Ginzburg in 2009 [52, 53]. This model only allows the two most important relaxation rates in the LBE; it retains some of the advantages of the MRT model in terms of accuracy and stability, while maintaining the simplicity of implementation and hence the computational efficiency. The idea behind the TRT model is that since each velocity vector $\vec{e}_i$ has an opposite one $\vec{e}_{\bar{i}} = -\vec{e}_i$, any pair of population with opposite velocities $(\vec{e}_i, \vec{e}_{\bar{i}})$ can be decomposed into its symmetric (even) and anti-symmetric (odd) components, written respectively as f^+ and f^-:

$$f_i = f_i^+ + f_i^-, \quad f_{\bar{i}} = f_i^+ - f_i^-, \quad f_i^{\pm} = \frac{1}{2}\left(f_i \pm f_{\bar{i}}\right), \tag{13}$$

f_i^+ is always positive and can be seen as the average probability of finding particles moving along the axis $\vec{e}_i$ in any direction, while f_i^- can be either positive or negative and depicts the average direction of motion along that axis.

The TRT collision can then be expressed as:

$$C_{\text{TRT}}(f_i) = \lambda_e \left(f_i^+ - f_i^{(eq)+}\right) + \lambda_o \left(f_i^- - f_i^{(eq)-}\right) \tag{14}$$

where λ_e and λ_o are the two relaxation parameters respectively associated with the even and odd equilibrium distribution functions $f_i^{(eq)+}$ and $f_i^{(eq)-}$.

In a more recent paper, Karlin et al. [54] revisited the MRT model and introduced a new two-step relaxation operator that gives a compromise between the enhanced stability and accuracy of the MRT and the simplicity of the BGK scheme.

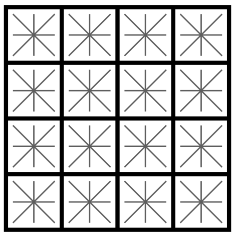
2.2 General framework of the LBM

As a numerical method, the framework of the LBM is closely linked to its numerical implementation. This section introduces the concept of *space and time discretisation* that is inherent to any CFD method but is achieved in a slightly different manner in LBM. Then the general LBM algorithm is formally presented, as a numerical framework to solve the lattice Boltzmann equation introduced in the previous section. The two key steps of the algorithm, i.e. the streaming and collision steps, are presented here in a general form that is not dependent on any of the specific LBM model introduced throughout the chapter. More details on the algorithm and boundary condition will be given in the next chapter.

2.2.1 *Space and Time Discretisation*

All CFD models rely on the discretisation of the considered quantities (i.e. velocity, pressure, etc.). While these quantities possess an infinite degree of freedom in nature, because they are spatially extended fields, computers can only store a finite amount of variables, and represent them with a finite accuracy. These fields must therefore be represented by their values on a finite set of points distributed in space, this is known as *space discretisation*.

The simplest and most often used one is a regular lattice with fixed grid spacing (Δx), related to the resolution (N) along the length (L) by $\Delta x = L/(N-1)$. The location ($\vec{x}_{ijk}$) of a simulation node on the lattice is a multiple of the lattice spacing Δx in each of the dimensions,


A regular grid typically used for LBM.

$$\vec{x}_{ijk} = i\Delta x\,\hat{e}_x + j\Delta x\,\hat{e}_y + k\Delta x\,\hat{e}_z, \tag{15}$$

where $(\hat{e}_x, \hat{e}_y, \hat{e}_z)$ is the unity base of the tree-dimensional space, and $i, j, k = 0 \cdots N-1$. A scalar field $\phi(\vec{x})$ is hence represented by N^3 values ϕ_{ijk} that are numerical approximation of $\phi(\vec{x}_{ijk})$.

The time, normally a continuous quantity in nature, is discretised in the same manner by a set of equi-distributed finite time steps $t_n = n\Delta t$. A time varying field $\phi(t)$ is replaced by a numerical approximation $\phi_n \approx \phi(t_n)$. Moreover, the cumbersome index notation will be avoided in the remaining of this thesis for the more readable notation with parentheses, i.e. the expression $\phi\left(\vec{x}_{ijk}, t_n\right)$ is used to represent the numerical approximation (not the exact value) of ϕ at the position $\vec{x}_{ijk}$ and time step t_n. Alternatively, the notation $\vec{x}$ might be used in place of $\vec{x}_{ijk}$ when the indices are of little importance and $\phi(i, j, k, t)$ might be used when the values of the indices needs to be specified. Finally, although the distribution functions $\mathbf{f}(\vec{x}, t)$ are always dependent of space and time, $(\vec{x}, t)$ might be neglected to simplify the expression.

The above discretisation and writing conventions are reasonably standard in the field of CFD. Where the LBM distinguish itself is in the discretisation of the velocities. In the Boltzmann equation (6), the velocities $\vec{u}$ can take an infinite number of directions and magnitudes, while in the LBM these velocities are restricted to a finite set of directions ($\mathbb{V}$) that includes the zero velocity $\vec{e}_0 = \vec{0}$ and the links between a node and its closest neighbours. To insure isotropy, this set must be symmetric and contain a sufficient amount of directions[1]. A lattice with q velocities in d dimensions is usually denoted as DdQq, see some examples in figure 4 and Appendix A.

In summary, the Boltzmann equation is simulated numerically by discretising the system in both space and time, and major simplifications are made:

1 A minimum amount of velocity directions for two and three dimensional simulation is typically 9 and 19 (respectively). Lattices with more directions will achieve a better isotropy.

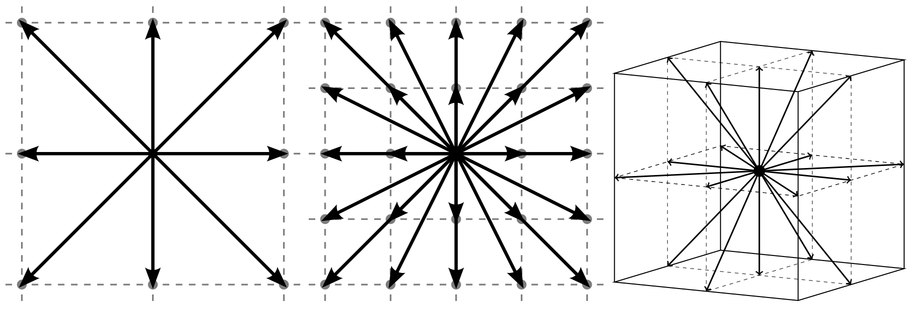

(a) D2Q9 (b) D2Q25 (c) D3Q19

Figure 4: Example of some lattice structure in two and three dimensions.

- the time evolves in finite time-steps Δt,

$$t \in \Delta t \mathbb{N} \tag{16}$$

- particle positions are confined to the nodes of a lattice,

$$\vec{x} \in \Delta x \mathbb{N}^d \tag{17}$$

- particle velocities are reduced to a finite set $\mathbb{V}$, made from the links between lattice sites,

$$\mathbb{V} = \{\vec{e}_i\}_{i \in \{0,\dots,q-1\}} \tag{18}$$

2.2.2 *Algorithm*

The attractiveness of the LBM comes from the simplicity of its algorithm. The velocity distribution functions are discretised in space, velocity and time as explained in the previous section. The index i is used to match each distribution function $f_i(\vec{x}, t)$ to its corresponding discrete velocity $\vec{e}_i$, the bold notation $\boldsymbol{f}$ represent a column vector regrouping the q distribution functions,

$$\boldsymbol{f}(\vec{x}, t) = \left(f_0(\vec{x}, t), \dots, f_{q-1}(\vec{x}, t)\right)^{\mathsf{T}}, \tag{19}$$

where T is the transpose operation. The discrete lattice Boltzmann equation can then be written as

$$\boldsymbol{f}(\vec{x} + \vec{e}_i \Delta t, t + \Delta t) - \boldsymbol{f}(\vec{x}, t) = \boldsymbol{\Omega}(\boldsymbol{f}(\vec{x}, t)). \tag{20}$$

In the above equation, the left hand side represents the streaming motion of the particles stored in f_i along the direction $\vec{e}_i$ between the time t and $t + \Delta t$, and the right hand side is the change in $\boldsymbol{f}$ due to the collision operator $\boldsymbol{\Omega}$. As a result, any lattice Boltzmann model solving for the LBE is implemented with the following steps:

1. Initialisation.
   The flow requires some initial state at the beginning of the simulation, it is common practice to initialise it in a rest state (zero velocity) but the choice of initial condition can influence the outcome of the simulation (see example in section 6.3.3).

2. Streaming.
   The velocity distribution functions are *streamed* along each lattice link as in figure 5. This operation allows the diffusion of the information from a node to its neighbour and it can be understood as the equivalent to the advection term in the Navier-Stokes equation. But unlike the advection in classical CFD, the streaming step in LBM is an exact operation, it does not involve any interpolation and does not introduce any numerical viscosity.

3. Collision.
   The details of this step are highly dependent on the model, but in general the macroscopic moments (e.g. velocity and density) are computed at each node from the local (streamed) distribution functions. These are then used to compute a local equilibrium and the distribution functions are relaxed towards it (either with BGK or MRT). The *locality* of the collision-step allows to update each node independently and result in a highly efficient parallel algorithm (see chapter 4).

4. Boundary.
   Clearly the streaming step as depicted in figure 5 is only valid for the bulk of the simulation domain, on the edges some distribution functions are leaving the simulation domain and some undefined ones are entering. It is the role of the boundary condition to properly define the distribution functions entering the domain so that the recovered macroscopic quantities can be specified. See chapter 3 for more details.

2.3 LBM FOR MULTI-PHYSICS APPLICATIONS

As mentioned in the introduction, the lattice Boltzmann method has been applied with success to many types of flow problems and engineering applications. The traditional BGK LBM is only valid for single phase isothermal incompressible flows but the collision operator and, in particular, the form of the equilibrium distribution function can be modified for different flow properties or even for different macroscopic equations.

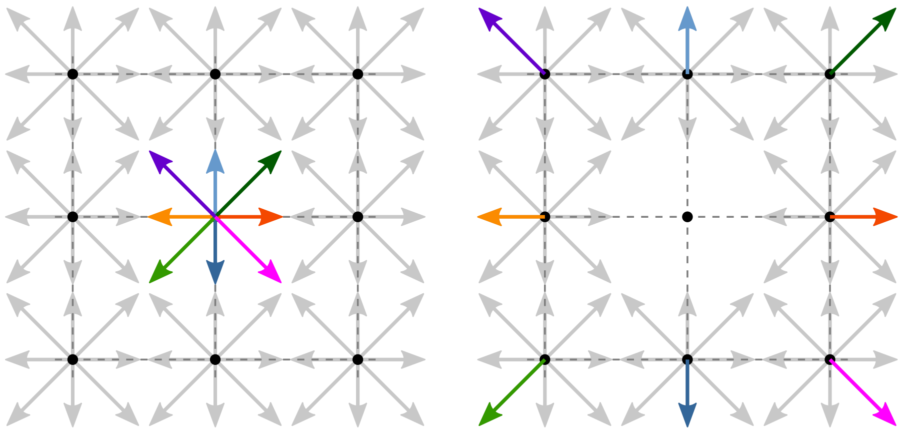


Figure 5: Location of the density distribution functions before and after the streaming step on a D2Q9 lattice. The velocities are drawn at half their actual length to avoid overlapping.

2.3.1 *Standard LBM Model*

While it is difficult to track down an official LBM model, what is referred to in this section as the *standard model* uses the equilibrium function first introduced by Qian et al. in 1992 [55] using the BGK relaxation scheme. It is probably the most used model in the literature and it is designed to simulate an isothermal incompressible single phase flow; although it introduces a small artificial compressibility term in the macroscopic Navier-Stokes equation (see comment in section 2.4.3). The next two sections discuss how it can be modified to remove the artificial compressibility or to simulate fully compressible fluids.

In this model, the macroscopic density and velocity are recovered from the set of velocity distribution functions at each node $\boldsymbol{f}(\vec{x},t)$. The density is simply the sum of all the distributions

$$\rho(\vec{x},t) = \sum_{i=0}^{q-1} f_i(\vec{x},t) \ , \tag{21}$$

where q is again the number of velocity directions. The momentum is the average of the microscopic velocities $\vec{e}_i$ weighted by their respective distribution functions f_i. Once the density is known, the velocity can be obtained from the momentum as

$$\vec{u}(\vec{x},t) = \frac{1}{\rho}\sum_{i=0}^{q-1} f_i(\vec{x},t)\,\vec{e}_i \, . \tag{22}$$

As explained previously, in the BGK approximation, the collision operator $\boldsymbol{\Omega}(\boldsymbol{f})$ is written as the relaxation of the distribution functions $\boldsymbol{f}(\vec{x},t)$ towards a local equilibrium $\boldsymbol{f}^{(eq)}(\vec{x},t)$ with the characteristic relaxation time τ. Hence, each velocity direction $i \in \{0, \ldots, q-1\}$ obeys the following discrete lattice Boltzmann equation

$$f_i\left(\vec{x}+\vec{e}_i, t+1\right) = f_i\left(\vec{x}, t\right) - \frac{f_i\left(\vec{x}, t\right) - f_i^{(eq)}\left(\vec{x}, t\right)}{\tau}, \tag{23}$$

where the left hand side is a simple streaming operation.

The local equilibrium $\boldsymbol{f}^{(eq)}\left(\vec{x}, t\right)$ is determined as low Mach number expansion of the Maxwell-Boltzmann distribution [56], up to the second order in the velocity $\vec{u}$ and takes the general form

$$f_i^{(eq)} = \rho\left(2\pi K_B T\right)^{-D/2} e^{-\frac{\left(\vec{e}_i-\vec{u}\right)^2}{2K_B T}} \tag{24}$$

$$\simeq \rho w_i \left(A + B\left(\vec{e}_i \cdot \vec{u}\right) + C\left(\vec{e}_i \cdot \vec{u}\right)^2 + D\left(\vec{u} \cdot \vec{u}\right)\right), \tag{25}$$

where A, B, C, D are constants to be determined and $\{w_i\}$ are lattice-dependent constants that depends on the magnitude of $\{\vec{e}_i\}$ (but not its direction). The above expression is only valid for small velocities, i.e. small Mach numbers u/c_s, where $c_s = 1/\sqrt{3}$ is the dimensionless speed of sound. The constants are found by imposing the constraints of conservation of mass and momentum during the collision process, i.e. the zeroth and first order moments of $f_i^{(eq)}$ must verify

$$\sum_{i=0}^{q-1} f_i^{(eq)} = \rho, \qquad \sum_{i=0}^{q-1} f_i^{(eq)}\vec{e}_i = \rho\vec{u}. \tag{26}$$

The higher order moments of $f_i^{(eq)}$ must be chosen so that the resulting continuum equations correctly describe the hydrodynamics of the model. This forces the second order moment to be

$$\sum_{i=0}^{q-1} f_i^{(eq)} e_{i\alpha} e_{i\beta} = P_{\alpha\beta} + \rho u_\alpha u_\beta, \tag{27}$$

where $P_{\alpha\beta}$ is the pressure tensor.

Using all of the above constraints, the constants can be computed and the resulting equilibrium is

$$f_i^{(eq)} = \rho w_i \left(1 + 3\left(\vec{e}_i \cdot \vec{u}\right) + \frac{9}{2}\left(\vec{e}_i \cdot \vec{u}\right)^2 - \frac{3}{2}\left(\vec{u} \cdot \vec{u}\right)\right), \tag{28}$$

the values of the weighting coefficients depends on the lattice structure and are given in Appendix A. For instance, for the D2Q9 lattice, the weights are $w_0 = 4/9$ for the rest velocity, $w_{1-4} = 1/9$ for the main axes and $w_{5-8} = 1/36$ for the diagonals.

2.3.2 *Incompressible Model*

The equilibrium proposed in the previous section recovers the Navier-Stokes equations with an additional compressibility term, as in equation (11). If the fluctuations in the fluid density are small, then the density can be considered constant i.e. $\rho \approx \rho_0$ and equation (11) becomes the standard incompressible Navier-Stokes equations,

$$\vec{\nabla} \cdot \vec{u} = 0\,,$$
$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \vec{\nabla} \vec{u} = -\frac{\vec{\nabla} p}{\rho} + \nu \vec{\nabla}^2 \vec{u}\,.$$

Hence, the Navier-Stokes equation can be recovered from the standard BGK LBM on the assumption that (i) the Mach number is small and (ii) the density has small spatial and temporal fluctuations. In some cases, the solution of the BGK LBM might depart from the direct solution of the incompressible Navier-Stokes equations due the effects of the compressibility.

Another model [57] proposed to eliminate the compressible effects by directly introducing the pressure into the equilibrium distribution functions (in place of the density) in order recover exactly the incompressible Navier-Stokes equations, with no artificial compressible term. The model, originally designed for a D2Q9 lattice, was later extended [58] for any D*d*Q*q* lattice (hence for three dimensions). The newly defined equilibrium takes the following form

$$f_i^{(eq)} = \lambda_i p + s_i\left(\vec{u}\right)\,, \tag{29}$$

where $s_i\left(\vec{u}\right)$ is the usual $s_i\left(\vec{u}\right) = w_i \left(1 + 3\left(\vec{e}_i \cdot \vec{u}\right) + \frac{9}{2}\left(\vec{e}_i \cdot \vec{u}\right)^2 - \frac{3}{2}\vec{u}^2\right)$, and the parameters $\{\lambda_i\}$ are determined by the constraints of conservation of mass, momentum and second order tensor, that leads to

$$\lambda_i = 3w_i, \qquad \lambda_0 = 3\left(w_0 - 1\right). \tag{30}$$

For instance the coefficients for a D2Q9 lattice are $\lambda_0 = -5/3$, $\lambda_{1-4} = 1/3$ and $\lambda_{5-8} = 1/12$.

The macroscopic variables for an incompressible fluid, i.e. the velocity $\vec{u}$ and the pressure p, are given by

$$\vec{u} = \sum_i f_i \vec{e}_i, \qquad p = -\lambda_0^{-1}\left(\sum_{i>0} f_i + s_0\left(\vec{u}\right)\right)\,. \tag{31}$$

The exact incompressible Navier-Stokes equations can be derived from the above model through a multi-scale expansion. The physical meaning of f_i in this model is different from that of the standard LBM, here it is rather a Lagrangian variable than a distribution function and it can take negative values. Furthermore, the pressure is no longer determined through the density by an equation of state any more, instead the pressure is directly defined from $\boldsymbol{f}$ and can also take negative values.

Another alternative form of equilibrium, is the so-called *pseudo-incompressible* model first introduced by He and Luo in 1997 [3]. In this model, it is assumed that the density is close to a constant $\rho = \rho_0 + \delta\rho$ that is substituted into the equilibrium as

$$f_i^{(eq)} = w_i \left(\rho + \rho_0 \left(3\,(\vec{e}_i \cdot \vec{u}) + \frac{9}{2}\,(\vec{e}_i \cdot \vec{u})^2 - \frac{3}{2}\vec{u}^2 \right) \right) , \qquad (32)$$

where the terms in $\delta\rho\vec{u}$ and $\delta\rho\vec{u}^2$ are neglected. The above equilibrium yields the exact incompressibility condition $\vec{\nabla} \cdot \vec{u} = 0$ for steady flows, but otherwise propagates sounds waves in almost the same way as the standard LBM, although at an incorrect speed [59]. Moreover, it has been shown to introduce some additional violations of the Galilean invariance [60].

2.3.3 *LBM for Compressible Flows*

The standard LBM is only valid for slightly compressible flows (i.e. quasi-incompressible) as the equilibrium distribution functions are obtained as a low Mach number expansion of the Maxwell-Boltzmann distribution. However, compressible flows problems often involve the study of shock waves which happen at high Mach number.

For a compressible flow, the macroscopic energy density (ρE) must be solved on top of the usual density, velocity and pressure, hence the equilibrium must satisfy

$$\sum_i \frac{1}{2} f_i^{(eq)} e_{i\alpha} e_{i\alpha} = \frac{1}{2}\left(\rho\vec{u}^2 + Dp\right) = \rho E\,. \qquad (33)$$

Several ways to construct new equilibrium distribution functions that satisfy the above equation have been proposed in the literature. For weakly compressible flows, i.e. still relatively small Mach number, it is possible to construct $f_i^{(eq)}$ from the same expansion as equation 25, but the coefficients are not constants, they are functions of the velocity and energy [61, 62]. While these models are straightforward extensions of the standard LBM, they require a large set of microscopic velocities, they are limited in the range of specific-heat ratio and Prandtl number as well as suffering for severe numerical instability, even with MRT [63, 64] or with two coupled lattices [65].

Some alternative models have been suggested for the simulation of compressible flows with the LBM [66, 67, 68, 69], including solving for the discrete velocity Boltzmann equation using a finite volume method [70] which has been applied to two- and three-dimensional flows around an aircraft [71, 72]. Overall, compressible models for the LBM are more complicated and further research efforts are needed to tackle the current challenging issues.

2.3.4 *Multiphase and Multicomponent Models*

A multiphase fluid system involves phase interfaces (e.g. liquid and gas) while a multicomponent system is mixture comprised of different fluid species (e.g. water and oil). Furthermore, a multicomponent fluid can be either miscible (e.g. tea and milk) or immiscible (e.g. water and air) and a multiphase system can contain one or more fluid components.

Many industrial processes involve multiphase and multicomponent fluid flows, examples are inkjet printing [73], atomisation and spray [74], digital rock physics for petroleum extraction [75, 76], microfluidics [77], etc. But because the macroscopic dynamics of these flows is usually very complicated, traditional CFD models can encounter difficulties with these systems. On the other hand, the LBM proposes to recover the macroscopic behaviour of multiphase flows as a natural consequences of microscopic interactions among fluid molecules. This feature is recognised as one of the main advantages of the LBM.

A variety of multiphase and multi-component models have been established for the LBM from different viewpoints, as presented in this section.

Colour Based Models

The first LBM model for multiphase flow was put forward by Gunstensen et al. in 1991 [78] as an improved version of the multiphase lattice gas automaton [79]. In this model, each phase is represented by a different set of distribution functions $\boldsymbol{f}_r$ and $\boldsymbol{f}_b$, distinguished by their respective colour (e.g. red and blue). The total distribution function for the mixture, $\boldsymbol{f} = \boldsymbol{f}_r + \boldsymbol{f}_b$ evolves with an additional collision operator $\boldsymbol{\Omega}^p(\boldsymbol{f})$ to model the perturbations that aroused by interfacial tensions,

$$\boldsymbol{f}(\vec{x} + \vec{e}_i \Delta t, t + \Delta t) - \boldsymbol{f}(\vec{x}, t) = \boldsymbol{\Omega}(\boldsymbol{f}) + \boldsymbol{\Omega}^p(\boldsymbol{f}) \, . \tag{34}$$

The inter-particle interactions are modelled by the local colour gradient ($\vec{G}$) associated with the density differences between the two phases (ψ),

$$\begin{aligned} \psi(\vec{x}, t) &= \rho_r(\vec{x}, t) + \rho_b(\vec{x}, t) \, , \\ \vec{G}(\vec{x}, t) &= \sum_i \psi(\vec{x} + \vec{e}_i, t)\, \vec{e}_i \, , \\ \boldsymbol{\Omega}^p &= A|\vec{G}| \cos(2\theta_i) \, , \end{aligned} \tag{35}$$

where θ_i is the angle between $\vec{e}_i$ and $\vec{G}$, A is a parameter controlling the surface tension. In the bulk region, the colour gradient is zero, and so is $\boldsymbol{\Omega}^p$, hence the surface tension only takes effect in the interfacial region, which is physically sound.

Once the total distribution is calculated according to equation (34), the red and blue distributions need to be recovered through a *recolouring* process [78] to enforce the direction of the colour flux ($\vec{H} = \rho_r \vec{u}_r - \rho_b \vec{u}_b$) to match that of the colour gradient $\vec{G}$. During this process, the fluid is driven to the bulk region with the same colour, inducing the phase separation.

Modified colour models have been proposed [80, 81, 82, 83], including its extension to miscible fluids [84], and these models have been successfully applied to flows in porous media [81] and spinodal decomposition [80]. However, the colour models are the earlier LBM models for multiphase flows, they are based on a heuristical approach rather than a physically correct description of the fluid.

Pseudo-Potential Models

In 1993, Shan and Chen [85] introduced a *pseudo-potential* to depict the interaction force between fluid particles, and was later built upon by several models [86, 87]. The pseudo-potential approach is the most widely used LBM multiphase model due to its simplicity and versatility [29], furthermore it can be simply adapted to a multi-component system through the use of multiple distribution functions [85, 88].

In this model, the force experienced by the particles of the component σ at the position $\vec{x}$ from the particles of $\bar{\sigma}$ at $\vec{x}'$ has the following form,

$$\vec{F}\left(\vec{x},\vec{x}'\right) = -G\left(\left|\vec{x}-\vec{x}'\right|\right)\psi_{\sigma}\left(\vec{x}\right)\psi_{\bar{\sigma}}\left(\vec{x}'\right)\left(\vec{x}'-\vec{x}\right)\,, \tag{36}$$

where G is a function controlling the interaction strength depending on the distance $|\vec{x}-\vec{x}'|$ (usually a Green function) and ψ is a function that only depends on the local density, which is implicitly a measure of of the average distance between two particles. This is contrary to continuum physics, where the detail of the interaction is given as a function of the distance between two interacting particles (i.e. van der Walls force). The function $\psi\left(\rho\left(\vec{x}\right)\right)$ is called the *effective mass* [89]. The components σ and $\bar{\sigma}$ can be different or the same. The total force acting on the particles at $\vec{x}$ is the sum of all the interaction forces with the neighbours,

$$\vec{F}\left(\vec{x}\right) = -\psi\left(\vec{x}\right)\sum G\left(\left|\vec{x}-\vec{x}'\right|\right)\psi\left(\vec{x}'\right)\left(\vec{x}'-\vec{x}\right)\,. \tag{37}$$

When only the closest neighbours on the lattice are considered, the force can be written as,

$$\vec{F}\left(\vec{x}\right) = -G\psi\left(\vec{x}\right)\sum_{i=1}^{q} w\left(|\vec{e}_i|^2\right)\psi\left(\vec{x}+\vec{e}_i\right)\vec{e}_i\,, \tag{38}$$

where G is a scalar denoting the interaction strength, positive and negative values lead to a repulsive and attractive force between particles, respectively. $w(|\vec{e}_i|^2)$ are some weights used for the calculation of the force and they are different from those in equation (28). The effective mass in the work of [85] has the following form,

$$\psi\left(\vec{x}\right) = \rho_0\left(1-\exp\left(-\frac{\rho\left(\vec{x}\right)}{\rho_0}\right)\right)\,, \tag{39}$$

where ρ_0 is a normalisation constant usually chosen as unity.

The interaction force needs to be included into the LBM, it can be achieved in different ways with various degrees of accuracy and stability [90] but in the original model of [85] it is achieved by modifying the velocity used in the calculation of the equilibrium distribution functions,

$$\vec{u}^{(eq)}(\vec{x}) = \vec{u}(\vec{x}) + \tau \left(\frac{\vec{F}(\vec{x})}{\rho(\vec{x})} + \vec{g} \right) , \tag{40}$$

where $\vec{g}$ is the acceleration due to gravity. In the case of a multi components system, the common velocity $\vec{u}$ in the above equation becomes a weighted average of the velocity of each phase

$$\vec{u} = \frac{\sum_\sigma \frac{\rho_\sigma \vec{u}_\sigma}{\tau_\sigma}}{\sum_\sigma \frac{\rho_\sigma}{\tau_\sigma}} . \tag{41}$$

This force modifies the equation of state from that of an ideal gas, i.e. $PV = nRT$, to

$$P(\vec{x}) = \rho(\vec{x}) c_s^2 + \frac{G c_s^2}{2} \psi(\vec{x})^2 . \tag{42}$$

It is possible to rearrange the expression for ψ in order to get a more realistic equation of state, and this can significantly enhance the attainable density ratio [91].

Free Energy Models

Another approach is the free energy model proposed by Swift et al. in 1996 [92] in which phase effects are directly introduced into the collision process by considering a generalized equilibrium distribution function that includes a non-ideal pressure tensor term.

The free energy of a one-component, two-phase fluid can be expressed by a Landau free energy functional [93],

$$\Psi = \int_V \left(\psi_b(\rho) + \frac{\kappa}{2} (\partial_\alpha \rho)^2 \right) dV - \mu_b \int_V \rho \, dV + \int_S \psi_s(\rho_s) \, dS , \tag{43}$$

where ψ_b describes the bulk free energy of the system, $\frac{\kappa}{2}(\partial_\alpha \rho)^2$ models the free energy associated with any interfaces in the system and the parameter κ is related to both the surface tension and interface width, the second integral correspond to a Lagrange multiplier that conserves the total mass of the system and the last integral describes the interaction between the fluid and the solid surface.

Based on the free energy Ψ, the pressure (p) and thermodynamic pressure tensor $(\boldsymbol{P})$, which considers the contribution of the interface, can be written as

$$p(\vec{x}) = \rho \frac{\delta \Psi}{\delta \rho} - \Psi = p_0 - \kappa \rho \vec{\nabla}^2 \rho - \frac{\kappa}{2} \left(\vec{\nabla} \rho \right)^2 , \tag{44}$$

$$\boldsymbol{P}_{\alpha\beta} = p \delta_{\alpha\beta} + \kappa \frac{\partial \rho}{\partial x_\alpha} \frac{\partial \rho}{\partial x_\beta} , \tag{45}$$

where $p_0 = \rho\psi'(\rho) - \psi(\rho)$ is the equation of state.

The free energy model uses the same relaxation scheme as the standard LBM, see equation (23), but the equilibrium $f_i^{(eq)}$ has an additional dependency on the density gradient and is constructed to enforce the following macroscopic constraints,

$$\sum_i f_i^{(eq)} = \rho\,, \qquad \sum_i f_i^{(eq)}\vec{e}_i = \rho\vec{u}\,,$$
$$\sum_i f_i^{(eq)} e_{i\alpha}e_{i\beta} = \boldsymbol{P}_{\alpha\beta} + \rho u_\alpha u_\beta + \nu\left(u_\alpha\partial_\beta\rho + u_\beta\partial_\alpha\rho + \delta_{\alpha\beta}u_\gamma\partial_\gamma\rho\right). \tag{46}$$

As for the standard LBM, the equilibrium is also a power series expansion in the velocity $\vec{u}$, and can be expressed as

$$f_i^{(eq)} = A_i + B_i\left(\vec{e}_i\cdot\vec{u}\right) + C_i\vec{u}^2 + D_i\left(\vec{e}_i\cdot\vec{u}\right)^2 + \boldsymbol{G} : \vec{e}_i\vec{e}_i\,, \tag{47}$$

and according to the constraints in equation 46, the coefficients can be obtained as

$$\begin{gathered}
A_i = w_i\left(p_0 - \kappa\rho\vec{\nabla}^2\rho - \frac{\kappa}{2}\left(\vec{\nabla}\rho\right)^2 + \nu\vec{u}\cdot\vec{\nabla}\rho\right), \\
B_i = w_i\rho, \qquad C_i = -\frac{w_i\rho}{2}, \qquad D_i = \frac{3w_i\rho}{2}, \\
G_{\alpha\alpha} = \frac{1}{2}\left(\kappa\left(\partial_\alpha\rho\right)^2 + 2\nu u_\alpha\partial_\alpha\rho\right), \\
G_{\alpha\beta} = \frac{1}{16}\left(\kappa\left(\partial_\alpha\rho\right)\left(\partial_\beta\rho\right) + \nu u_\alpha\partial_\beta\rho + \nu u_\beta\partial_\alpha\rho\right).
\end{gathered} \tag{48}$$

With the above equilibrium, the recovered hydrodynamic equations for the free energy model are

$$\frac{\partial\rho}{\partial t} + \vec{\nabla}\cdot\left(\rho\vec{u}\right) = 0\,, \tag{49}$$

$$\begin{aligned}
\frac{\partial\left(\rho\vec{u}\right)}{\partial t} + \vec{\nabla}\cdot\left(\rho\vec{u}\vec{u}\right) &= -\vec{\nabla}\cdot\boldsymbol{P} + \nu\vec{\nabla}^2\left(\rho\vec{u}\right) + \vec{\nabla}\left(\lambda\vec{\nabla}\cdot\left(\rho\vec{u}\right)\right) \\
&\quad - \left(\tau - \frac{1}{2}\right)\frac{\partial p_0}{\partial\rho}\vec{\nabla}\cdot\left(\vec{u}\vec{\nabla}\rho + \left(\vec{\nabla}\rho\right)\vec{u}\right),
\end{aligned} \tag{50}$$

where the last term represents the surface tension and

$$\nu = \frac{1}{4}\left(\tau - \frac{1}{2}\right), \qquad \lambda = \left(\tau - \frac{1}{2}\right)\left(\frac{1}{2} - \frac{\partial p_0}{\partial p}\right). \tag{51}$$

He-Chen-Zhang *Model*

A common issue with multiphase LBM model is that large density gradient in the vicinity of the interface can lead to numerical instability. To overcome this difficulty, He et al. [87] proposed a model that treats the interface as a separate quantity using an index function ϕ.

They also introduced a change of variable in the distribution functions so that these large gradients are cancelled by a small quantity in the transformed equations.

The evolution of ϕ is described by a first set of distribution functions, noted $\boldsymbol{f}$, with the following discrete lattice Boltzmann equation

$$f_i\left(\vec{x}+\vec{e}_i\Delta t, t+\Delta t\right)-f_i\left(\vec{x},t\right)= \\ -\frac{1}{\tau}\left(f_i-f_i^{(eq)}\right)-3\Gamma_i\left(\vec{u}\right)\left(1-\frac{1}{2\tau}\right)\left(\vec{e}_i-\vec{u}\right)\cdot\vec{\nabla}\psi\left(\phi\right), \quad (52)$$

where $f_i^{(eq)}=\phi\Gamma_i\left(\vec{u}\right)=\phi w_i\left(1+3\vec{e}_i\cdot\vec{u}+\frac{9}{2}\left(\vec{e}_i\cdot\vec{u}\right)^2-\frac{3}{2}\vec{u}^2\right)$ and the index function is defined as $\phi=\sum_i f_i$.

The external forces and surface tension do not have an effect on the mass conservation, thus they are neglected in equation . Only the intermolecular force term, $\vec{\nabla}\psi\left(\phi\right)$, must be retained because it is essential in maintaining a sharp interface, which is expressed with the Carnahan-Starling equation of state [94] as

$$\psi\left(\phi\right)=p-\frac{\phi}{3}=\phi^2\left(\frac{\frac{2}{3}\left(2-\phi\right)}{\left(1-\phi\right)^3}-4\right). \quad (53)$$

The density ρ and viscosity ν are determined by interpolating the bulk values in the liquid phase $\left(_l\right)$ and gas phase $\left(_g\right)$

$$\rho\left(\phi\right)=\rho_g+\frac{\phi-\phi_g}{\phi_l-\phi_g}\left(\rho_l-\rho_g\right), \nu\left(\phi\right)=\nu_g+\frac{\phi-\phi_g}{\phi_l-\phi_g}\left(\nu_l-\nu_g\right). \quad (54)$$

The relaxation time τ is related to ϕ via the viscosity

$$\tau\left(\phi\right)=\frac{1}{2}+3\nu\left(\phi\right). \quad (55)$$

A second set of distribution functions, noted $\boldsymbol{g}$, describes the evolution of the velocity and pressure,

$$g_i\left(\vec{x}+\vec{e}_i\Delta t, t+\Delta t\right)-g_i\left(\vec{x},t\right)=-\frac{1}{\tau}\left(g_i-g_i^{(eq)}\right)+ \\ \left(1-\frac{1}{2\tau}\right)\left(\vec{e}_i-\vec{u}\right)\cdot\left(\Gamma_i\left(\vec{u}\right)\left(\vec{F}_s+\vec{g}\right)-\left(\Gamma_i\left(\vec{u}\right)-\Gamma_i(\vec{0})\right)\vec{\nabla}\psi\left(\rho\right)\right), \quad (56)$$

where $g_i^{(eq)}=w_i\left(p+\rho\left(3\vec{e}_i\cdot\vec{u}+\frac{9}{2}\left(\vec{e}_i\cdot\vec{u}\right)^2-\frac{3}{2}\vec{u}^2\right)\right)$ and the macroscopic pressure and momentum are recovered as

$$p=\sum_i g_i-\frac{1}{2}\vec{u}\cdot\vec{\nabla}\psi\left(\rho\right), \qquad \rho\vec{u}=3\sum_i g_i\vec{e}_i+\frac{1}{2}\left(\vec{F}_s+\vec{g}\right), \quad (57)$$

and $\psi\left(\rho\right)$ is the non-ideal part of the equation of state and $\vec{F}_s$ is the surface tension force

$$\psi\left(\rho\right)=p-\frac{\rho}{3}, \qquad \vec{F}_s=\kappa\rho\vec{\nabla}\left(\vec{\nabla}^2\rho\right). \quad (58)$$

The large density gradients, $\vec{\nabla}\psi(\rho)$, appearing in equation (56), are multiplied by the small term $\left(\Gamma_i(\vec{u}) - \Gamma_i(\vec{0})\right)$, which help to reduce numerical errors and improve the stability.

Free-Surface Models

In the simulation of fluid systems with high density ratio (e.g. water and air) pseudo-potential and free energy models suffer from instability issues. In the free-surface approach, the low density phase (e.g. the air) is neglected and the collision process is carried out only on the nodes occupied (either fully or partially) by the water, the location of the interface is updated solely based on the motion of the water and can be done either with a volume of fluid technique [95, 96] or through the level set method [97, 98] . This type of model is popular in classical CFD but contrary to the previously presented models, the dynamics of the interface is not a result of the microscopic interactions, instead the surface tension effects must be introduced ad hoc by modifying the collision based on the interface curvature [97]. As only the liquid phase is simulated, some of the distribution functions at the interface needs to be reconstructed and fluid nodes need to be created or destroyed as the interface moves [99].

Further Reading

Because of its computational efficiency and conceptual simplicity, the LBM has been widely employed for the simulations of a variety of multiphase flows [100, 93, 73, 74, 76, 77]. However, the LBM suffers from some limitations which restrict its applications, these include the relative large spurious current [101] that increase with the density ratio and are linked to the isotropy of the force [102, 103], the thermodynamic inconsistencies, the limitation to low density and kinematic viscosity ratio and the dependence of surface tension and density ratio on the viscosity (see [29] and references therein).

2.3.5 *Thermal Models*

Currently, thermal LBM models fall into one of three categories: (i) the multi-speed models, (ii) the double distribution functions models and (iii) hybrid models.

Multi-Speed models

The multi-speed models are straight-forward extensions of the classical isothermal LBM BGK model, where the temperature T (or the internal energy e) is computed as a second order moment of the distribution functions $\boldsymbol{f}$, as [62]

$$\rho e = \frac{1}{2} \sum_{i} (\vec{e}_i - \vec{u})^2 f_i \,, \tag{59}$$

and the temperature is related to the internal energy by $e = (3/2)\, k/mT$ where k is the Boltzmann factor and m the mass [31].

Usually, in order to accurately obtain the macroscopic evolution equation for the temperature, a multi-speed model requires a larger set of microscopic velocities than what is used in the corresponding isothermal model (typically 52 in 2D [104]) and requires the equilibrium to be expanded up to the third or fourth order [105]. Hence, the method is not very computationally efficient.

In the original model [62], the Prandtl number is fixed to unity, this was later fixed [106] but with an incorrect viscosity in the viscous dissipation of the energy. Moreover, the method suffers from severe numerical instabilities for large temperature variations [104], which limits its domain of applications to small Rayleigh numbers.

Double Distribution Functions Models

These models are based on the idea that if the viscous heat dissipation and compression work done by the pressure are negligible, then the temperature field is passively advected by the fluid and obeys a simple advection-diffusion equation. In which case, the temperature equation can be solved on another lattice using an independent set of temperature distribution functions.

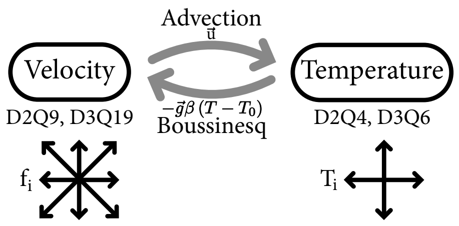


Figure 6: Structure of couple thermal LBM model using two independent set of distribution functions. The first set is used to recover the macroscopic density and velocity, while the second set recovers the macroscopic temperature, but only requires a reduced number of directions.

In the original model [107], the temperature was treated as a purely passive scalar but since then, a variety of models have been developed that treat the effect of the temperature on the velocity field in different ways; for instance [108] treats the temperature as a component of a fluid mixture and [109] proposed a model to simulate thermal flows of non-ideal gases. In the approach of [30] and chosen in this work, it is assumed that the temperature variations have little effect on the fluid motion except in the buoyancy term, where the fluid density is expected to be a linear function of the temperature,

$$\rho(\vec{x}) = \rho_0 \left(1 - \beta \left(T(\vec{x}) - T_0\right)\right) \tag{60}$$

where ρ_0 and T_0 are respectively the average fluid density and temperature and β is the coefficient of thermal expansion. This assumption, known as the Boussinesq approximation, is only valid for small Mach number flows.

Under such circumstances, the governing equations for the velocity and the temperature are

$$\vec{\nabla} \cdot \vec{u} = 0\,, \tag{61}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \vec{\nabla} \vec{u} = -\vec{\nabla} P + \nu \vec{\nabla}^2 \vec{u} + \vec{g} \beta \left(T - T_0\right)\,, \tag{62}$$

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \vec{\nabla} T = \alpha \vec{\nabla}^2 T\,, \tag{63}$$

where $P = p/\rho_0$ is the reduced dynamic pressure, $\vec{g}$ is the acceleration due to gravity and α is the thermal diffusivity.

The first two equations can be solved with the standard LBM model with an appropriate forcing term (see next paragraph), and the third equation is solved on a second lattice with a simplified equilibrium,

$$T_i\left(\vec{x} + \vec{e}_i \Delta t, t + \Delta t\right) - T_i\left(\vec{x}, t\right) = -\frac{T_i\left(\vec{x}, t\right) - T_i^{(eq)}\left(\vec{x}, t\right)}{\tau_T} \tag{64}$$

where T_i is the temperature distribution function associated with the microscopic velocity $\vec{e}_i$, $T_i^{(eq)}$ is the corresponding equilibrium and τ_T is a dimensionless relaxation time, related to the thermal diffusivity by

$$\alpha = \frac{\left(2\tau_T - 1\right)}{4} \frac{\Delta x^2}{\Delta t}\,. \tag{65}$$

The macroscopic temperature is recovered as the sum of the temperature distribution functions,

$$T = \sum_i T_i\,. \tag{66}$$

The lattice used to solve the temperature advection-diffusion equation requires relatively less direction than the one used for the velocity. Typically D2Q5 and D3Q7 are sufficient. Furthermore, the zero velocity can be dispensed with. The general form for the temperature distribution function on a lattice with or without the zero velocity, i.e. DdQ(2q+1) or DdQ(2q), is given as [110]:

$$T_i^{(eq)} = \frac{T}{b}\left(1 + \frac{b}{2} \vec{e}_i \cdot \vec{u}\right) \tag{67}$$

where $b = 2d$ or $b = 2d + 1$.

The velocity and temperatures lattices are coupled by the addition of the Boussinesq forcing term $\vec{F}_B = -\vec{g}\beta\,(T - T_0)$ on the right hand side of the discrete lattice Boltzmann equation (23),

$$f_i\,(\vec{x} + \vec{e}_i\Delta t, t + \Delta t) = f_i\,(\vec{x}, t) - \frac{f_i\,(\vec{x}, t) - f_i^{(eq)}\,(\vec{x}, t)}{\tau} + F_i\Delta t\,. \tag{68}$$

A simple approximation for the forcing term F_i is to apply the following forcing term to the two distribution functions that are aligned with the direction of the gravity [30]:

$$F_i = \frac{\vec{e}_i \cdot \vec{F}_B}{2} = \frac{\pm\beta\,(T - T_0)}{2}\,. \tag{69}$$

It is also possible to use other discretisations of the forcing term to improve the accuracy, [111] compares various expressions and proposes the following expression in order to reduce the lattice effects on the recovered macroscopic equations,

$$F_i = \left(1 - \frac{1}{2\tau}\right) w_i\,(\vec{e}_i - \vec{u} + (\vec{e}_i - \vec{u}) \cdot \vec{e}_i) \cdot \vec{F}_B\,. \tag{70}$$

where the fluid velocity is redefined as

$$\vec{u} = \frac{1}{\rho}\left(\sum_i f_i\vec{e}_i + \frac{\Delta t}{2}\vec{F}_B\right)\,. \tag{71}$$

It should noted that the above expression is only valid for a specific form of the equilibrium and for a specific collision operator (i.e. BGK), moreover numerical experiments have shown that the gain in accuracy over the simpler expression in equation (69) is minimal (see section 6.5.4).

Hybrid Methods

In the previous it was shown that the double distribution models solve separately for the velocity and temperature equations on two different lattice. It is also possible to combine the LBM for the resolution of the velocity equation with another numerical method (usually finite difference) for the resolution of the simpler advection-diffusion equation for the temperature. This type of models are known as *hybrid models* [112, 113, 114, 115]. Unlike the other thermal models, an hybrid model does not add a new set of distribution functions for the temperature, instead it adds a new scalar field, hence the memory requirements are significantly reduced and hybrid models can achieve really high performance on GPU [115].

Again, assuming that the compression work and viscous heat dissipation are negligible, the temperature equation can be expressed as

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \vec{\nabla} T = \alpha \vec{\nabla}^2 T\,, \tag{72}$$

and this equation is discretised with the following scheme

$$T\left(\vec{x}, t+\Delta t\right) - T\left(\vec{x}, t\right) = \alpha \Delta^* T - \vec{u} \cdot \vec{\nabla}^* T\,, \tag{73}$$

where $\vec{\nabla}^*$ and $\Delta^* \equiv \left(\vec{\nabla}^*\right)^2$ are respectively the discrete gradient and Laplace operators. Their expression depends on the lattice used for the velocity field, for example, in the D2Q9 model

$$\begin{aligned}
\nabla^*_x T &= T_{i+1,j} - T_{i-1,j} + \frac{1}{4}\left(T_{i+1,j+1} - T_{i-1,j+1} + T_{i+1,j-1} - T_{i-1,j-1}\right)\,,\\
\nabla^*_y T &= T_{i,j+1} - T_{i,j-1} + \frac{1}{4}\left(T_{i+1,j+1} - T_{i+1,j-1} + T_{i-1,j+1} - T_{i-1,j-1}\right)\,,
\end{aligned} \tag{74}$$

$$\begin{aligned}
\Delta^* T = 2&\left(T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}\right)\\
&- \frac{1}{2}\left(T_{i+1,j+1} + T_{i-1,j+1} + T_{i-1,j-1} + T_{i+1,j-1}\right) - 6T_{ij}\,.
\end{aligned} \tag{75}$$

2.3.6 *Fluid-Structure Interaction*

An object moving in a fluid transfers some of its momentum to the surrounding fluid, and reciprocally the movements of the flow creates a force on an immersed object that is impacting its momentum. Hence, the study of a moving object immersed in a fluid is a complex two-way coupling problem. The object can be either a rigid solid or it can have a deformable structure. In any case, the tracking of the fluid-solid interface as the objects moves and deforms is a challenging task.

In conventional CFD methods, the computation of a flow around a moving deformable object often involves several tedious grid generations over time, making the solution process complicated. To ease this process, it is desirable to develop an approach that decouples the solution of the main governing equations and the implementation of the boundary conditions. The immersed boundary method (IBM) is such an approach, it was originally proposed for the simulation of blood flow in the heart [116]. In this method the flow field uses a fixed Eulerian mesh that does not need to coincide with the Lagrangian points used to represent the solid boundary and the effect of the boundary is represent by an additional forcing term on the flow, which is solved on the whole domain including the exterior and interior of the object. Since the flow is solved without considering the presence of immersed objects, it is no longer necessary to match the location of the object boundary with the grid and a simple Cartesian mesh is sufficient.

The IBM approach has been successfully adapted to the LBM, for the simulation of the sedimentation of rigid particles in a viscous fuid [117, 118, 119] and for the simulation of cellular blood flow with deformable cells [120]. Details on the implementation of the IBM within a LBM scheme are available in section 3.2.8.

The dynamics of the objects (i.e. collisions and deformations) cannot be solved by the LBM alone and require an external solver (for instance, using the finite element method [119]). This solver requires information on the solids characteristics as well as the force exerted by the fluid onto the object which is extracted from the LBM flow field. This force can be either evaluated by the stress-integration method on the surface of the body [121] (as in classical CFD) or by the momentum exchange method [122, 123]. The stress-integration method is computationally laborious for two-dimensional flows and in general difficult to implement for three-dimensional flows, while the momentum-exchange method is reliable, accurate, and easy to implement for both two-dimensional and three-dimensional flows [124].

Force evaluation based on stress integration

The force exerted by the fluid onto an object can be evaluated by integrating the total stresses on the boundary of that object $\partial\Omega$,

$$\vec{F} = \int_{\partial\Omega} \left(-p\boldsymbol{I} + \rho\nu\left(\vec{\nabla}\vec{u} + \left(\vec{\nabla}\vec{u}\right)^{T}\right)\right)\cdot\vec{n}\,dA, \tag{76}$$

where $\vec{n}$ is the unit normal vector out of the boundary $\partial\Omega$. The pressure p can be easily evaluated using the equation of state $p = c_s^2\rho$, while the deviatoric stress $\tau_{\alpha\beta} = \rho\nu\left(\partial_\alpha u_\beta + \partial_\beta u_\alpha\right)$ can be evaluated using the non-equilibirum part of the distribution functions $f_i^{(neq)} = f_i - f_i^{(eq)}$,

$$\tau_{\alpha\beta} = \left(1 - \frac{1}{2\tau}\right)\sum_i f_i^{(neq)}\left(e_{i,\alpha}e_{i,\beta} - \frac{1}{D}\vec{e}_i\cdot\vec{e}_i\delta_{\alpha\beta}\right), \tag{77}$$

where $e_{i,\alpha}$ and $e_{i,\beta}$ are respectively the α^{th} and β^{th} Cartesian components of the discrete velocity $\vec{e}_i$.

Both the values of the pressure and each of the six components of the symmetric deviatoric stress tensor need to be obtained at the actual surface location using a second-order extrapolation scheme based on the values of p and $\tau_{\alpha\beta}$ at the neighbouring fluid nodes. This make the method complicated and its implementation error-prone.

Force evaluation based on the momentum-exchange

In the momentum exchange method, the force exerted on the solid on a given boundary node $\vec{x}_b$ is calculated from the force at each link between two opposite directions $\vec{e}_\alpha = -\vec{e}_{\bar{\alpha}}$

$$\vec{F}_\alpha\left(\vec{x}_b\right) = \left(f_\alpha\left(\vec{x}_b\right) - f_{\bar{\alpha}}\left(\vec{x}_b + \vec{e}_\alpha\right)\right)\vec{e}_\alpha, \tag{78}$$

and the total hydrodynamic force is found by simply summing over all boundary links,

$$\vec{F} = \sum_{\text{all } \vec{x}_b} \sum_{\alpha \in \text{links}} \vec{F}_\alpha (\vec{x}_b) \, . \tag{79}$$

The momentum exchange method is very easy to implement computationally, moreover it has been shown to yield a similar accuracy to the stress integration technique and even becomes superior when the resolution is limited [124]. It is therefore the recommended force evaluation method for its simplicity, accuracy and robustness, either for straight or curved boundaries [124].

2.4 ALTERNATIVE MODELS

While the BGK LBM can qualify as the standard LBM model due to its popularity, its simple collision operator means its numerical stability degrades for high Reynolds number flows, i.e. low viscosity or as the relaxation τ approaches $1/2$. This can be improved with the use of multiple relaxation time [125, 59] (MRT, see section 2.1.5) but choosing the right values for the additional relaxation times can be sometimes cumbersome.

As a result, there has been significant efforts to develop alternative models to the BGK LBM, usually to improve the stability for high Reynolds numbers flows and this section summarises some of them. It should be noted that any of these alternative models can in theory be adapted for multi-physics applications (thermal effects, multiphase, etc.) as described in the previous section.

When the Reynolds number is really high, the flow becomes turbulent, and in such cases it is sensible to make use of a turbulence model, as will be discussed in the next section.

2.4.1 *Entropic LBM*

The idea of using Boltzmann H-theorem [126] for lattice Boltzmann systems was first introduced by Chen and Teixeira in 2000 [127] to reduce the instabilities observed in multi-speed thermal LBM models. It was later proposed to adjust the local relaxation time to satisfy the H-theorem [18], essentially making the method unconditionally stable provided that some requirements are fulfilled [128]. A space varying relaxation time is similar to the technique employed by a sub-grid turbulence model in LBM (see section 2.5.1) and the entropic LBM can be interpreted as a kind of sub-grid model [129, 130].

A discrete H-function $H(f) = \sum_i f_i \ln (f_i / w_i)$ is introduced and the equilibrium distribution function is modified as an extrema of $H(f)$ under constraints of mass and momentum conservation:

$$f_i^{(eq)} = \rho w_i \prod_{j=1}^{D} \left(2 - \sqrt{1 + 3u_j^2}\right) \left(\frac{2u_j + \sqrt{1 + 3u_j^2}}{1 - u_j}\right)^{e_{i,j}} \tag{80}$$

where j is the index of the spatial directions. The relaxation process is modified to include a dynamically adjustable parameter α,

$$f_i\left(\vec{x} + \vec{e}_i \Delta t, t + \Delta t\right) - f_i\left(\vec{x}, t\right) = \alpha\beta\left(f_i^{(eq)} - f_i\right), \tag{81}$$

and α is evaluated at each node and each time-step as the solution of the H function monotonicity constraint:

$$H\left(f + \alpha\left(f^{(eq)} - f\right)\right) = H(f). \tag{82}$$

The above equation does not have an analytical solution, hence a root finding algorithm must be used at each node, introducing a significant computational overhead compared to the BGK. This procedure ensures that the entropy is not increasing over time. In a simulation, the over-relaxation parameter α is automatically fixed to the value $\alpha = 2$ whenever the simulation is resolved and is modified only when the simulation is under-resolved, in a mechanism similar to a sub-grid model, although only informed by the Second Law of Thermodynamics (H-theorem). These stabilisation events may be rare in time and very localised in space, but they give the method its unconditional stability [128].

The computational overhead introduced by the root finding process depends on the physics of the problem (as the number of iterations depends on the turbulence) and can introduce load balancing issues (as some nodes might converge faster than others). There is a number of possible optimisation strategies over the standard Newton-Raphson algorithm [131], see discussion in [132], including approximating H based on a Taylor series expansion [133] which effectively removes the need for an iterative process. This last technique is particularly convenient on GPU, as it avoids computational divergence between the threads, but the algorithm looses some of its stability. The equilibrium in equation (80) can also be used without a root finding algorithm (i.e. using $\alpha = 2$ everywhere) and the resulting method can simulate fluid flows, although its stability degrades to that of BGK LBM [59].

The entropic LBM has set off strong debates within the LBM community [134, 135]. Nevertheless, the method has been shown to produce accurate result at high Reynolds number, even for under-resolved flows [136, 132, 137]. Additionally, the entropic scheme can be combined with MRT [138] for enhanced accuracy for large Reynolds number, and can be adapted for multi-physics problem, including multiphase flows [139] and thermal flows [140].

2.4.2 *Cascaded LBM*

The cascaded LBM is a recent model developed by Geier in 2006 [19, 141] which implements the MRT in a central moment space in a attempt to restore the Galilean invariance of the LBM. The BGK LBM itself is not fully invariant due to a cubic velocity dependency of the viscosity [142], which limits its domain of validity; and the original MRT method transforms the distributions functions into a set of raw moments that are relaxed at different rates, introducing additional violations of the Galilean invariance that are not present with BGK[2]. Setting all relaxation rates to the same value solves the problem but negates the superiority of MRT over SRT. The insufficient Galilean invariance of the standard BGK and MRT can be identified as the source of the instabilities that appear for small viscosities [19].

The cascaded LBM uses an additional transformation to convert the raw moments into central moments that are shifted by the local fluid velocity, hence restoring Galilean invariance. As a result, it has a better stability and isotropy than MRT, allowing to reach higher Reynolds number. Moreover the method appears to be unconditionally stable, i.e. it achieves stable simulation down to the limit of zero viscosity. However its complexity is significantly greater than that of MRT, hence the detailed equations are not reproduced here, instead the reader is referred to the following papers for more details on the implementation [143, 144, 145].

In three dimensions, the D3Q27 lattice is used, and the microscopic velocities are $\vec{e}_{(i,j,k)} = (i, j, k)$ where $i, j, k \in \{-1, 0, 1\}$ and the components of $\vec{e}_{(i,j,k)}$ are written as $\left(e_{(i)}, e_{(j)}, e_{(k)}\right)$. The raw moments are computed as

$$\rho M_{pqr} = \sum_{i,j,k} f_{(i,j,k)} e_{(i)}^{p} e_{(j)}^{q} e_{(k)}^{r}, \tag{83}$$

where $p, q, r \in \{0, 1, 2\}$. For instance with this notation $M_{000} = 1$ and M_{100}, M_{010} and M_{001} are respectively the x, y and z components of the macroscopic velocity $\vec{u} = (u_x, u_y, u_z)$. Using the same notations, the central moments are defined as

$$\rho \widetilde{M}_{pqr} = \sum_{i,j,k} f_{(i,j,k)} \left(e_{(i)} - u_x\right)^{p} \left(e_{(j)} - u_y\right)^{q} \left(e_{(k)} - u_z\right)^{r}. \tag{84}$$

As the central moments can be expressed from the raw moments (and vice versa) a possible algorithm for the cascaded LBM is as follows, i.e. like MRT with an additional transformation to and from central moments.

1. Compute the raw moments M_{pqr} from the distribution functions $f_{(i,j,k)}$.

2 In a recent paper [60], Dellar proposes a MRT model with velocity-dependent collision rates to restore the Galilean invariance.

2. Compute the central moments $\widetilde{M}_{pqr}$ from the raw moments.

3. Compute equilibrium central moments $\widetilde{M}^{(eq)}_{pqr}$ and relaxes using different relaxation times.

4. Compute the updated raw moments M^*_{pqr} from the updated central moments $\widetilde{M}^*_{pqr}$.

5. Compute the updated distribution functions $f^*_{(i,j,k)}$ from the updated raw moments.

6. Stream the new distribution functions to their neighbours.

The complexity of the cascaded LBM and the need for a large number of velocities in three-dimensions (i.e. D3Q27) means that it has a limited popularity. Nevertheless, it has been used successfully for turbulent simulations at high Reynolds number (up to 10^5) [145, 146], but the lack of a sub-grid turbulence model means it does require large resolutions to accurately resolve the flow, as in DNS.

While the cascaded LBM fixes the Galilean invariance of MRT, it does not fix all of its problems. The relaxation of different moments with different rates leads to a *hyperviscosity*, i.e. a numerical viscosity, that severely limits the accuracy of the method [147, 148]. This artifact is not well known as it exists only in three dimensions and depends on moments that are neglected on some popular lattices even in three dimensions. An ad hoc solution to this problem was proposed in the form of the factorised central moment LBM [147], later formalised in the context of the cumulant LBM [145]. Instead of solving the problems of the MRT by hand as they arise, [145] postulate constraints for the observable quantities to evolve on independent time scales. While assuming different relaxation rates for different moments violate these constraints, cumulants [149], as an alternative to the moments, fulfil the constraints naturally.

In can be noted that the cascaded LBM can also be seen as adopting a generalised equilibrium in the frame at rest [150].

2.4.3 *Link-Wise Artificial Compressibility Method*

The link-wise artificial compressibility method (LW-ACM) is a recent formulation of the artificial compressibility method for solving the incompressible Navier-Stokes equation. The artificial compressibility method was first introduced by A.J. Chorin in 1967 [151], the principle of the method lies in the introduction of an artificial compressibility term in the incompressible Navier-Stokes equation that disappears from the final results in the limit of the vanishing Mach number. It is in that regard similar to the LBM (see [7] for a discussion on the similarities and differences). In a recent paper, Asinari et. al. [21] propose a reformulation of the method by a finite set of

GPU model	maximum resolution	Performance
GTX Titan	576^3 (6 GB)	2500 MLups
Tesla K40	736^3 (12 GB)	2200 MLups

Table 1: Performances of the LW-ACM D3Q19 isothermal model, measured on the Tesla K40 GPU and extracted from [153] for the GTX Titan.

discrete links on a regular Cartesian mesh, in analogy with the LBM. The method also shows similarities with the lattice kinetic scheme [152].

The LW-ACM exploits both the advantages of the LBM (in terms of computational efficiency and simple boundary conditions) and of classical CFD (for instance, the consistency can be analysed using a standard Taylor expansion, like for finite difference schemes). The method only deals with macroscopic variables (i.e. velocity and pressure) and the updating rule makes it possible to recover all the necessary information from these variables. As a result, the memory demand is significantly smaller than LBM and the method can be optimised to achieve a better computational throughput, with a numerical stability and accuracy similar to that of MRT.

The small memory requirements allows to simulate large resolutions with good performances. Table 1, summarises the performance for a three-dimensional (D3Q19) LW-ACM program running on GPU, measured in million of nodes updated per second (MLUPS) and the maximum resolution fitting in memory on the Tesla K40 used for the thesis and on a GTX Titan as reported in [153]. These results are significantly higher than that of the BGK LBM on the same lattice, see section 5.1.2. The method has been very recently extended to the simulation of thermal flows and performs at 1300 MLups for a maximum size of 536^3 on the GTX Titan [154].

2.4.4 *Further Reading*

It is difficult to give a comprehensive list including every LBM models. It is the author's belief that the entropic model, cascaded model (and its generalisation in the context of cumulants) and the link-wise ACM are the most promising alternatives to the standard BGK/MRT LBM, which explains why these were detailed in the previous subsections. But there is a plethora of other models that would deserve further investigations.

Perhaps the simplest modification of the BGK LBM aiming at improving its stability is the regularised model, introduced by Latt in 2006 [155, 156]. The basic idea of this approach is to replace the non-equilibrium part of the distribution functions $f_i^{(neq)} = f_i - f_i^{(eq)}$

with a new first order regularised value $f^{(reg)}$ derived from the non-equilibrium stress tensor $\boldsymbol{\Pi}^{(neq)}_{\alpha\beta} = \sum_i f_i^{(neq)} \left(e_{i\alpha}e_{i\beta} - \delta_{\alpha\beta}c_s^2\right)$ as

$$f^{(reg)} = \frac{w_i}{2c_s^4}\boldsymbol{Q}_i : \boldsymbol{\Pi}^{(neq)} \tag{85}$$

where $\boldsymbol{Q}_i = \vec{e}_i\vec{e}_i - c_s^2\boldsymbol{I}$. This procedure enhances the stability and accuracy of the BGK LBM to a level close to that of a MRT model, but without the cumbersome moment transformation.

A recurring issue of LBM is the creation of spurious oscillation (usually in a chequerboard pattern) near areas of steep velocity gradients, Godunov theory [157] shows that this issue affect every high-order scheme and that the capturing of a discontinuity without oscillation requires that the spacial accuracy of the scheme is reduced to first order. Brownlee et al. [158, 159, 160] proposed to adapt a flux limiter scheme [161] to the LBM. Flux limiter scheme were invented to combine high resolution schemes in areas with smooth fields and first-order schemes in areas with sharp gradients. Limiters can be constructed for the LBM by changing the non-equilibrium part of the distribution, i.e. by forcing the distribution functions to stay close to their local equilibrium, without changing the macroscopic variables. This helps enforcing the positivity of the distribution functions and hence the stability. While the limiters only need local information to the node, the neighbour nodes can be included as well to implement a non-local filter such as a *median filter* to reduce speckle noise [162] or a selective viscosity filter [163]. The optimal filtering strategy is problem specific difficult and cannot be given in a general way.

2.5 TURBULENCE MODELLING IN LBM

Although the alternative LBM models presented in the previous section can achieve stable simulation for higher Reynolds number than the BGK LBM, the use of a subgrid turbulence model becomes necessary when the simulation is under resolved, i.e. the grid resolution is large compared to the Kolmogorov scale. As mentioned in the introduction (see section 1.2.4) the variety of scales and the complexity of turbulent flows makes makes the modelling of turbulence an extremely challenging topic in CFD. Generally speaking, these models can be classified into two categories: (i) large eddy simulation (LES) and (ii) Reynolds averaged Navier-Stokes (RANS) based turbulence models, each resolving the flow down to a different scale (see figure 1).

2.5.1 *Large Eddy Simulation*

The viewpoint of a Large Eddy Simulation (LES) is that large eddies are usually more energetic and contribute more to the transport than the smaller ones, so it is sufficient to only solve accurately for the large scale of the flow and the effect of the unresolved small scales on the fluid motion is modelled using a sub-grid model. In practice, simple turbulence models, like the Smagorinsky model [164], suppose that the energy dissipation by the small scale is isotropic and can be represented using a local eddy viscosity ν_t which is added to the fluid kinematic viscosity $\nu = \nu_0 + \nu_t$. In effect, this additional viscosity increases the numerical stability of a simulation by damping short-wavelength oscillations. LES simulations are still expensive, but not as much as DNS and can give higher accuracy than RANS, especially when the transient behaviour of the flow is of importance.

The Smagorinsky turbulence model, originally introduced in 1963 to compute the Reynolds stress term in the filtered Navier-Stokes equations [164], has been very popular in LBM since 1994 [165]. The simplicity of the calculation of the strain rate tensor in LBM is the main reason for its success. In LBM, the local momentum stress-tensor $\bar{S}_{\alpha\beta}$ can be calculated from the second-order moment of the non equilibrium distribution functions,

$$\bar{S}_{\alpha\beta} = \frac{1}{2}\left(\frac{\partial \bar{u}_\alpha}{\partial x_\beta} + \frac{\partial \bar{u}_\beta}{\partial x_\alpha}\right) = \sum_{i=1}^{q} \vec{e}_{i\alpha}\vec{e}_{i\beta}\left(f_i - f_i^{(eq)}\right). \tag{86}$$

And the eddy viscosity can be computed as [165]

$$\nu_t = \frac{1}{6}\left(\sqrt{\nu^2 + 18 C_s^2 \Delta^2 \sqrt{\bar{S}_{\alpha\beta}\bar{S}_{\alpha\beta}}}\right), \tag{87}$$

where Δ is the filter width (i.e. Δx in LBM, often chosen as unity) and $C_s > 0$ is the Smagorinsky constant. Small values of C_s mean that the sub-grid turbulence adds very little viscosity to the fluid, while large values of C_s mean that sub-grid eddies have a strong impact on the diffusion of the fluid and the grid scale. Its actual value can depend on the geometry of the system but it is typically in the order of $C_s \approx 0.1$, which is smaller than the value of $C_s = 0.17$ used in the Navier-Stokes LES [166]. Moreover, numerical experiments in a 2D lid-driven cavity (see section 6.2.2) have shown that even smaller values (i.e. $C_s = 0.03$) yield better results, although deteriorating the stability.

The computation of the stress-tensor and corresponding eddy viscosity in LBM only requires information local to the node, hence it is very computationally efficient (see section 5.1.5). In effect, the Smagorinsky turbulence model is implemented in LBM by allowing the relaxation time to vary locally in space, as $\tau = 1/2 + 3(\nu_0 + \nu_t) = \tau_0 + 3\nu_t$.

2.5.2 *RANS Based Models*

The Reynolds averaged Navier-Stokes (RANS) approach to turbulence is to only simulate the mean flow averaged in time and to model the fluctuations using additional transport equations. It is the most computationally efficient way to simulate turbulent flows for engineering applications, but it should be regarded as an engineering approximation. Moreover, a single model cannot address all the complexity of turbulence, and a large variety of RANS models exits: algebraic models [167, 168, 169], one equation models [170, 171], and two equation models [172, 173, 174, 175]. These models can be adapted to the LBM, although not in their time-averaged form, as the LBM is intrinsically time dependant.

Amongst the various RANS models, the $k-\epsilon$ model [172, 173] is one of the most popular in engineering. It introduces two additional transport equations to model the turbulent kinetic energy (k) and the turbulent energy dissipation (ϵ). The first attempt at a RANS simulation with the LBM dates as early as 1995 [26], the method uses two additional sets of distribution functions to represent each of the turbulent properties k and ϵ, and their collision dynamics is defined as to recover the turbulence equations in their macroscopic behaviour. Another solution is to use the LBM to solve the velocity field in conjunction with a finite difference scheme to solve $k-\epsilon$, as in [176].

Other RANS models have been translated to LBM, such as the $k-\omega$ model [177] and the single equation Spalart-Allmaras model [178, 177, 179].

2.5.3 *Further Reading*

In its standard form, the Smagorinsky models assumes that the turbulence is isotropic but that is not the case for the flow near the walls and other boundaries. Thus, the model needs to be improved to be able to successfully simulate the near-wall flow. One possibility is to introduce a damping function that depends on the distance to the wall in order to account for the damping of scales near the walls [180, 181, 176]. Another possibility is to use a dynamic Smagorinsky model [180] which removes the need for an empirical value of the LES constant and allows it to vary both in space and time by computing it using two different filters.

Although the majority of the sub-grid turbulence models used in the LBM are built around the Smagorinsky model, some recent work by Sagaut [25, 182, 183] reuses the idea of the approximate deconvolution model of Stloz et al. originally introduced in 1999 for the Navier-Stokes LES [184]. This approach applies an approximate deconvolution operator directly on the filtered distribution functions, it is hence a microscopic description of the turbulence (rather that using

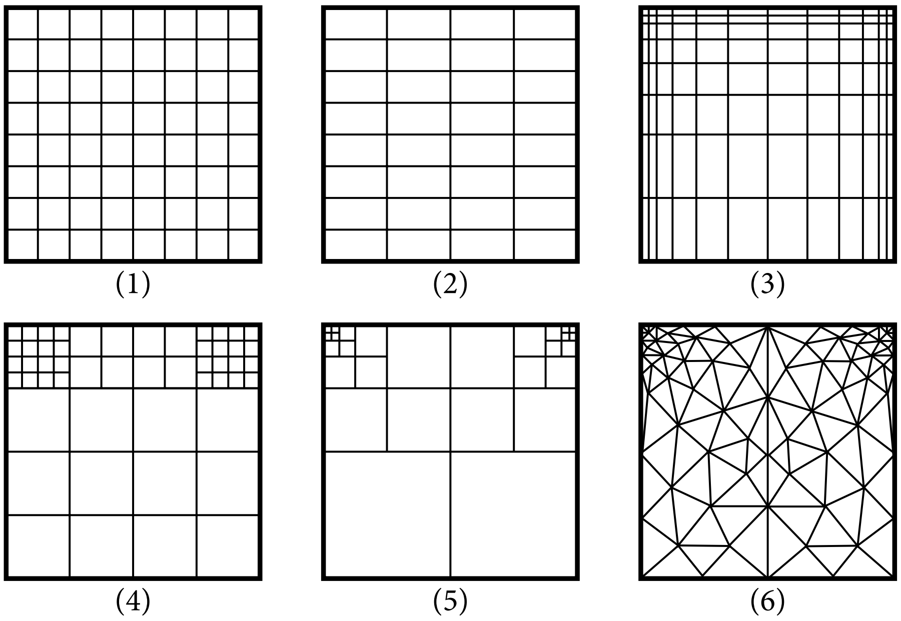


Figure 7: Illustration of different grid types.
(1) Regular grid. (2) Rectangular grid. (3) Non-uniform grid.
(4) Multi-block. (5) Quad-tree. (6) Unstructured grid.

the macroscopic stress-tensor), and has been implemented as part of the LaBS simulation tool[3] for engineering applications in automotive and aerospace industries.

The Smagorinsky model has been applied to large range of problems, both academic [166, 165, 185, 180] and industrial [178, 186, 146, 187], showing that the LBM LES achieves a similar degree of accuracy to Navier-Stokes LES [166] and can be combined with mesh refinement techniques for large Reynolds number flows on complex geometries [146, 187].

Other LES turbulence models have been incorporated into the LBM framework, such the wall-adapting large eddy-viscosity model [188] and the vorticity-streamfunction for two-dimensional flows [189].

It should be noted that a turbulence model can also be used in combination with the the improved collision operator presented in the previous section for enhanced stability and accuracy. For instance, the Smagorinsky model can be combined with the MRT [180, 181, 190, 185, 191].

2.6 LBM WITH NON-UNIFORM GRIDS

The standard LBM is restricted to square lattice grid in two spatial dimensions and cubic lattice grids in three dimensions. This is an heritage of the lattice gas automaton and allows the streaming step to be an exact operation that does require any interpolation, and hence

3 http://www.labs-project.org

does not introduce any additional numerical viscosity. The MRT can potentially integrate an aspect ratio $a = \Delta x/\Delta y$ into the transformation matrix into to allow the LBM on rectangular grids [192, 193, 194].

But the flexibility of these meshes is limited, and flow around irregular geometries require a fine mesh, i.e. high resolution, to be accurately resolved. Instead, it is possible to locally refine the mesh around the interface using a grid refinement [45, 195] or multi-block [196, 197] method. In these methods, two grids with different lattice spacing are combined, the distribution functions are updated on the fine grid first and then on the coarser grid. The fine and coarse grids usually share some common nodes, on which the distribution functions are exchanged by interpolation. Moreover, multiple levels of grid refinement can be introduced usually though the use of a quadtree or a octree for two- or three-dimensional grids [198]. This allows for a better accuracy of the solution near the boundaries and can still achieve a good computational throughput by maintaining a coarse grid for the bulk of the fluid domain. The grid refinement can also be adapted over time to follow the flow structures, this technique is known as *dynamic mesh refinement* [199, 200].

The LBM can function on non-uniform grids, i.e. the lattice spacing Δx varies with space, but the distribution functions must be interpolated after each streaming step [201]. This technique, known as the *interpolation supplemented LBM* allows the use of body-fitted grids [202, 121]. Moreover, once the fact that the LBM is a special finite difference scheme of the discrete Boltzmann equation is recognised, the computational mesh can be fully decoupled from the the underlying lattice, potentially allowing any arbitrary unstructured mesh using finite volume or finite element [203, 204].

However, these techniques introduce a significant complexity, hence a computational overhead, and the different interpolations can affect both the effective viscosity and the anisotropy of the solution [205]. It is unclear if the improvement on the accuracy is worth the added complexity and computational cost, as a fine regular GPU grid might still be faster than a locally refined CPU mesh, even with several recursive levels of refinements. Furthermore, several techniques like the immersed boundary method (see section 2.3.6) remove some of the needs for a refined mesh by using sub-grid interpolations to take account of the exact location of the mesh within the grid.

2.7 SUMMARY

This chapter gave a pseudo-exhaustive introduction to the LBM and the large variety of existing models. Some sections, like the study of the various thermal and turbulence models, was directly motivated by the application of the LBM to indoor air flows while some other sections, like multiphase flows and fluid-structure interaction, was mostly motivated by other applications that will be briefly discussed in chapter 8. Nevertheless, multiphase problems can be encountered in indoor air flow applications as well, for instance in the study of humidity [206] or for novel cooling techniques using spray evaporative cooling [207]. Fluid-structure interactions are relevant as well for the study of the effects on particles dispersion of an opening door or a person walking [208]. And the capability of the LBM in these areas will help future research. Meanwhile, the sections on alternative collision models and non-uniform grids techniques should be seen as viable alternative methods that may be investigated further in some future work.

3 LBM ALGORITHMS

The Navier-Stokes is a partial differential equation with time and space derivatives, and as such, it possesses a unique solution when adequate initial and boundary values for a given problem are specified. The problems of fluid dynamics are therefore *initial and boundary value problems*.

When the flow is assumed to be steady-state, i.e. not varying over time, the time derivatives in the equation can be removed and the steady-state Navier-Stokes becomes simply a boundary value problem. However, the LBM equation is intrinsically time dependant and requires some initial value. Different initial values can potentially lead to different flow results (see for instance the effect of the initial temperature in section 6.3.3). For transient Navier-Stokes, it is common to specify the initial value for a problem by imposing the velocity and pressure field at the initial time $t = 0$. The same principle applies to the LBM, but for the distribution functions; it is common practice to initialise them using the equilibrium distribution function, computed with the specified velocity and density.

While specifying an initial value is usually straightforward, boundary values also require to be specified to uniquely define a problem, and several approaches exist. The velocity profile along the domain boundaries can be specified by values, i.e. Dirichlet boundary condition, or recovered from the velocity gradient in the direction normal to the boundary, i.e. Neumann boundary condition, or the value of pressure on the boundary can be specified. Although there is still a wish to impose these boundary conditions for a LBM simulation, boundaries in LBM work at different level: the unknown distributions functions, i.e. those coming from outside of the simulation domain, needs to be specified (by value) so that the recovered macroscopic quantities matches the desired macroscopic boundary condition (i.e. Dirichlet, Neumann or pressure). But the problem of defining the unknown distribution functions for a specific macroscopic value is usually under determined, i.e. there are fewer equations than unknown. Hence there can be multiple implementations of the same boundary condition with different trade-off between accuracy and stability [209].

This chapter discusses the general implementation of the LBM on a computer, and should be applicable to any architecture. Some of the most useful boundary conditions are described for a single phase isothermal flow but should be amenable to other types of flows.

The goal of any LBM program is to numerically solve the discrete lattice Boltzmann equation,

$$\mathbf{f}\left(\vec{x}+\vec{e}_i\Delta t, t+\Delta t\right)=\mathbf{f}\left(\vec{x},t\right)+\Omega\left(\mathbf{f}\left(\vec{x},t\right)\right). \tag{88}$$

As explained in chapter 2, this equation can be solved on different types of lattices (DdQq), the collision operator can take different forms (BGK, MRT, etc.) and the equilibrium can have multiple definitions depending on the physics (single phase, multiphase, etc.). However, whatever the details of the scheme, there is always first an initialisation step to set the initial values of the distribution functions, followed by three steps:

1. the streaming step, that moves the distribution functions to their neighbour nodes,
2. the collision step, that implements the collision operator, usually by relaxing towards an equilibrium state,
3. the boundary step, that defines the unknown distribution functions at the boudaries.

The sequence of these three steps correspond to one time-step of the LBM simulation, they are hence repeated in a loop until the final time is reached. The simplicity of the LBM algorithm is one of its most appealing features.

3.1.1 *Initialisation Step*

As mentioned previously, the initialisation step is performed once at the beginning of the simulation and its goal is to define a starting value for the distribution functions. It is common practice to use the equilibrium distribution functions, evaluated with the desired macroscopic quantities as the initial values. As the equilibrium always conserves these quantities, the effective macroscopic values should be exactly those specified.

$$f_i\left(\vec{x},0\right)=f_i^{(eq)}\left(\rho\left(\vec{x}\right),\vec{u}\left(\vec{x}\right)\right) \tag{89}$$

For the standard LBM, a simple choice is often to start with a uniform flow with $\vec{u}=\vec{0}$ and $\rho=1$. But the initial velocity and pressure fields can also be made to vary in space, see for instance the study of the rolling shear layer in section 8.5.

3.1.2 *Streaming Step*

This step is the only one whose implementation is unchanged across every LBM models (apart from those on non-uniform grids). It is thus a fundamental step of the LBM and it is relatively simple, as it only require to copy the distribution functions to a different location and does not involve any computations.

The streaming step basically implements the left hand side of equation (88), on each lattice node $\vec{x}$, each distribution function $f_i(\vec{x})$ is streamed to an adjacent node in the direction of the microscopic velocity $\vec{e}_i$, i.e. at location $\vec{x}+\vec{e}_i$. Some care has to be taken as to avoid overwriting an existing distribution function during the process. While it is possible to use a single array $\boldsymbol{f}(\vec{x})$ to limit memory usage, the process can be cumbersome, especially in a parallel environment, and it is simpler and more efficient to just use a second array $\boldsymbol{f}^{\text{temp}}(\vec{x})$ to store the streamed distribution functions, despite the memory overhead [210, 211].

The streaming step can be formally written as

$$f_i^{\text{temp}}(\vec{x}+\vec{e}_i, t) = f_i(\vec{x}, t) \; , \tag{90}$$

and the process for a D2Q9 lattice is illustrated on the figure 5. The distributions functions are streamed from the centre node to the adjacent nodes. It is algorithmically equivalent to stream from the adjacent nodes to the centre node, i.e. $f_i^{\text{temp}}(\vec{x}) = f_i(\vec{x}-\vec{e}_i)$ and a discussion on the differences between the two schemes will be given in section 4.4.3.

The streaming step needs to be modified for the nodes on the boundary of the simulation domain because some distribution are exiting the domain and some unknown ones are entering it. It is often convenient to suppose that the domain is periodic in all directions, as the periodic boundary condition is very easy to implement (see section 3.2.1). It allows the streaming to keep the same form throughout all of the domain and the correct boundary conditions can be added as a post process by overwriting the periodically streamed distribution functions.

3.1.3 *Collision Step*

While the streaming step does not perform any computation and is purely a memory copy between neighbour nodes, the collision step contains most of the computations but they only usually only require information local to each node, and it is hence a perfect target for efficient parallel programming.

Its actual implementation is highly dependent on the LBM model considered, this section describes the implementation of the collision for the BGK and MRT single phase models.

BGK Collision Model

Using the BGK collision model, the distribution are simply relaxed toward a local equilibrium using a single relaxation time τ, it can be decomposed into three stages:

1. **computation of the macroscopic quantities** from the streamed distribution functions $\boldsymbol{f}^{\text{temp}}$. On each node, the local distribution functions are from the temporary array and used to compute the macroscopic quantities of the model (density ρ, velocity $\vec{u}$, etc...) using the equations (21) and (22),

2. **computation of the equilibrium distribution functions** $\boldsymbol{f}^{(\text{eq})}$ from the previously computed macroscopic quantities. This stage depends on the physical model, i.e. the form of the equilibrium. Some models might require some non-local quantities, like the gradient of the density for multiphase models, in which case the density has to be be stored to an array in stage 1 so that the gradient can be computed in stage 2,

3. **computation of the new distribution functions**, i.e. $\boldsymbol{f}\left(t+\Delta t\right)$ by relaxing the distribution functions in the temporary array $\boldsymbol{f}^{\text{temp}}$ toward the equilibrium $\boldsymbol{f}^{(\text{eq})}$ and save the result back to the original array $\boldsymbol{f}$.

In summary the BGK collision scheme can be implemented as

$$f_i\left(\vec{x}, t+\Delta t\right) = f_i^{\text{temp}}\left(\vec{x}, t\right) - \frac{1}{\tau}\left(f_i^{\text{temp}}\left(\vec{x}, t\right) - f_i^{(\text{eq})}\left(\vec{x}, t\right)\right) . \qquad (91)$$

MRT Collision Model

The MRT collision is also composed of four stages during which the distributions functions are transformed back and forth to the moments space where the relaxation is performed, the streaming step is left unchanged.

1. **computation of the macroscopic moments** from the streamed distribution functions $\boldsymbol{f}^{\text{temp}}$. It is essentially the same as stage 1 of BGK, but with additional moments. The moments are computed at each node through a matrix operation $\boldsymbol{m} = \boldsymbol{M} \cdot \boldsymbol{f}^{\text{temp}}$, where $\boldsymbol{M}$ is a $q \times q$ transformation matrix,

2. **computation of the equilibrium moments** $\boldsymbol{m}^{(\text{eq})}$, usually from the density and velocity computed in stage 1,

3. **relaxation** of the moments $\boldsymbol{m}$ towards their equilibrium $\boldsymbol{m}^{(\text{eq})}$ using different relaxation times $\omega_i = 1/\tau_i$, i.e. $m_i^* = m_i - \omega_i\left(m_i - m_i^{(\text{eq})}\right)$,

4. **reconstruction of the distributions functions $\boldsymbol{f}$** by using the inverse transformation matrix $\boldsymbol{M}^{-1}$ on the relaxed moments $\boldsymbol{m}^*$, i.e. $\boldsymbol{f}(\vec{x}, t+\Delta t) = \boldsymbol{M}^{-1} \cdot \boldsymbol{m}^*$.

In summary, the MRT collision scheme can be implemented as

$$\boldsymbol{f}(\vec{x}, t+\Delta t) = \boldsymbol{M}^{-1} \cdot \left(\boldsymbol{m}(\vec{x}, t) - \boldsymbol{\omega} \cdot \left(\boldsymbol{m}(\vec{x}, t) - \boldsymbol{m}^{(eq)}(\vec{x}, t)\right)\right), \quad (92)$$

where $\boldsymbol{m}(\vec{x}, t) = \boldsymbol{M} \cdot \boldsymbol{f}^{temp}(\vec{x}, t)$ and $\boldsymbol{\omega}$ is a diagonal matrix whose elements are $\{\omega_0, \ldots, \omega_q\}$.

It is important to remind that the choice of the transformation matrix $\boldsymbol{M}$ is not unique. Firstly, its size depends on the lattice considered (i.e. there areq moments, where q is the number of microscopic directions). Moreover, appart from the straighforward moments (density, velocity, pressure tensor), the higher order moments are often non-physical and can be chosen freely, provided that they are all moments are independent (i.e. that the matrix $\boldsymbol{M}$ is invertible). Therefore, no explicit expression for the transformation matrix is given in the thesis, instead the interest reader is referred to the following references [125, 20], where a sample implementation can be found for the D2Q9, D2Q15 and D3Q19 lattices.

The implementation of the MRT in matrix form as described above is straightforward to implement but it is computationally inefficient. The matrices are sparse, i.e. they contain many zero elements, and hence matrix multiplications involve many unnecessary operations. An optimised implementation should avoid using matrix multiplications altogether. One possibility is to expand these multiplications to remove the zero elements and avoid additional memory accesses, this is cumbersome to do by hand but the process can be automated (see section 4.7). Another possibility is to re-express the distribution functions directly from the moments, i.e. algebraically and without a transformation matrix [15].

3.1.4 *Boundary Step*

In a nutshell, the goal of the boundary step is to define the otherwise undefined distribution functions at the boundaries after the streaming step. There are multiple strategies that will be discussed in the next section.

Once the three steps (streaming,collision,boundary) have been applied, the simulation is considered to have advanced in time by Δt. These steps are repeated N times until the total simulation time $t = N\Delta t$ has been reached. The macroscopic density and velocity fields can be saved at regular intervals to a file, or in the case of a steady-state solution only once at the end when the simulation has converged.

The boundary conditions are crucial in CFD because they define the flow problem. Moreover, they sometimes dictate the accuracy and stability of a simulation, as their effect can be seen in the bulk phase, even far away from the boundary regions. As such, they deserve full section dedicated to them.

The difficulty of boundary implementation in LBM comes from the fact that there is no unique way of defining the distribution functions at the boundary to recover the desired macroscopic conditions. Indeed, while the macroscopic density and velocity is easily calculated from the particle populations, the reverse procedure is more troublesome. Since the beginning of LBM in the early nineties many different boundary implementations have been proposed [212, 209, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225] and there is no consensus as to which boundary condition is the best one. Instead, none of the method is really superior to the others, the choice of the best boundary condition depends on the flow problem, and there is usually a trade-off between accuracy and stability in show in multiple review papers [209, 226]. This chapter discusses the implementation of some of the most used boundary condition for the LBM. (And does not propose yet another boundary implementation, as it is customary for every Ph.D. thesis on LBM!)

3.2.1 *Periodic*

This is probably the simplest boundary condition, the distribution functions going out from one side of the domain are simply streamed back into the domain on the opposite side, as if they were attached together. For example along the x-direction, it can be implemented either with an *if* statements as

$$\begin{aligned} x_{\text{right}} &= \begin{cases} x+1 & \text{if } (x < N-1) \\ 0 & \text{else} \end{cases}, \\ x_{\text{left}} &= \begin{cases} x-1 & \text{if } (x > 0) \\ N-1 & \text{else} \end{cases}, \end{aligned} \tag{93}$$

where $x \in \{0, \dots, N-1\}$ is the node location and x_{right} and x_{left} are the location of the right and left neighbours, respectively, or with a *mod*ulus operation as

$$\begin{aligned} x_{\text{right}} &\equiv x+1 (\text{mod } N), \\ x_{\text{left}} &\equiv x-1 (\text{mod } N). \end{aligned} \tag{94}$$

Figure 8 illustrates this process for a D2Q9 lattice. If all the boundaries of a domain are set to periodic, then the domain topology can

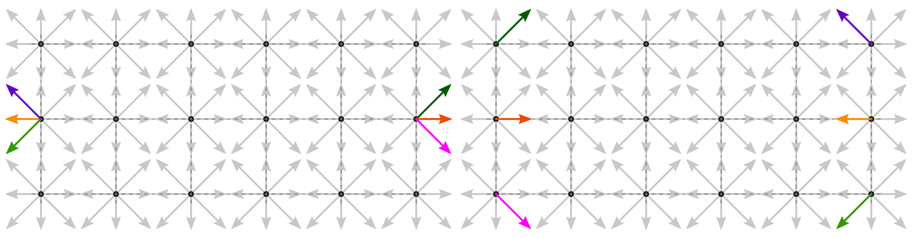
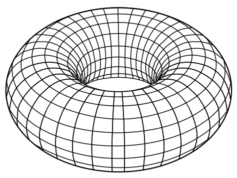

(a) before streaming (b) after streaming

Figure 8: Illustration of the periodic boundary condition along the x-axis for a D2Q9 lattice.

be assimilated to a *torus* (or an *hyper-torus* for three-dimensional domains). Such configuration can be used to approximate an infinite domain and to limit the effect of the boundaries on the fluid behaviour. It is a classical configuration for the study of isotropic turbulence in CFD.


torus geometry

3.2.2 *Force Equilibrium*

This simple boundary condition works by imposing the equilibrium distribution functions on either just the unknown or all the distribution functions of a boundary nodes, and can be use to set the velocity and density at the boundary.

$$f_i = f_i^{(eq)}\left(\rho, \vec{u}\right) \tag{95}$$

This is in general not a physically accurate boundary, because it does not conserve mass and it should be avoided in production code. However, it is very simple to implement and particularly robust, so it can be very useful to quickly set up a new boundary condition in a new code or for a new problem before switching for a more accurate (but often less stable) boundary scheme.

3.2.3 *Bounce-Back*

In this simple boundary condition, the distribution functions leaving the domain are reverted to replace the unknown ones,

$$f_i\left(\vec{x}\right) = f_{\bar{\imath}}\left(\vec{x}\right) , \tag{96}$$

where $\vec{e}_i$ and $\vec{e}_{\bar{\imath}}$ are pointing in opposite directions, i.e. $\vec{e}_i = -\vec{e}_{\bar{\imath}}$.

This effectively implements a *no-slip* boundary condition on a fixed wall, as no momentum is neither received or lost by the node. The relative simplicity of this boundary, compared to its equivalent in traditional CFD, played a major role in the popularity of the LBM.

This boundary has two variant implements, so called *full-way* and *half-way* bounce-back. This appellation is misleading as the physical location of the wall is always one half grid spacing beyond the last fluid node, in both variants. The main difference is that the half-way bounce-back treats boundary nodes as fluid nodes, hence the collision step is applied on those nodes, while they are considered as solid nodes in the full-way bounce-back.

Full-way bounce-back

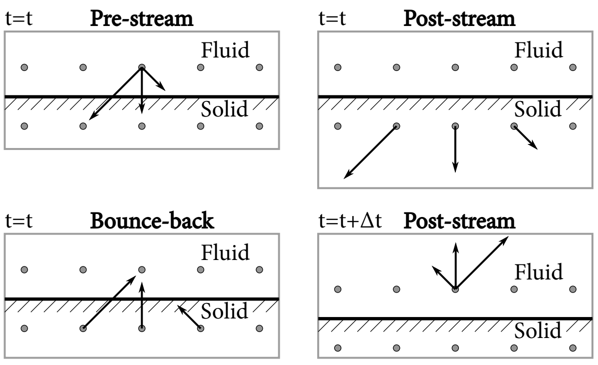


Figure 9: Illustration of the full-way bounce-back boundary condition on a D2Q9 lattice, for a no-slip wall located at the bottom of the simulation domain. The variable vector length represents different density values.

The full-way variant of the bounce-back method was historically the first one and dates back to the age of the lattice gas automaton [227]. It is relatively simpler because all of the distribution functions are reverted regardless of the wall normal and the boundary nodes are considered to be part of the solid, so the collision step is not applied on them, which is why this method is sometimes called *dry* bounce-back. These nodes live outside of the simulation domain and act as *ghost-nodes* purely to temporary store the bounced-back distributions. The physical location of the boundary is half-way in between the ghost-nodes and the fluid nodes, as depicted on figure 9. This boundary condition is perfect to describe the geometry of a porous media, where the pores can sometimes be one node thick and the normal to the pore is not easily defined [22, 81, 23, 228, 38].

Half-way bounce-back

The half-way variant of the bounce-back boundary condition is similar in its principle to the full-way variant, but the boundary nodes are considered to be part of the fluid domain (i.e. they are *wet*) and they are subject to collision step as any other fluid nodes. In this variant, the reflection occurs during the same time-step but only on the distribution functions that are exiting the domain. The half-way bounce-back is purely local and does not require a ghost node, but the wall-normal to the wall needs to be known in order to choose

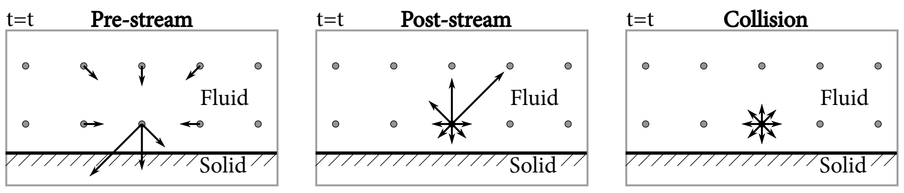


Figure 10: Illustration of the half-way bounce-back condition on a D2Q9 lattice, for a bottom solid wall. The three distribution functions normal to the wall, which are bounced-back, have different lengths to simplify the visualisation.

which distribution to flip. The wall is physical located at half a grid spacing outside of the fluid domain, as depicted on figure 10.

The half-way bounce-back has been shown to be more accurate than the full way scheme, as it does not suffer from the lag of one-time step. In particular, [215] showed that the half-way achieves second order convergence under space refinement while the full-way bounce-back is only first order. However this seems to be contradictory to the second order numerical convergence obtained for both models on a Poiseuille flow, see section 6.1.

Although the bounce-back method perform great in most situations, it might introduce a small slip velocity at the wall [215] and the exact location might be offset from the half-way grid [59], this can be solved with the MRT collision operator [59].

3.2.4 *Free Slip*

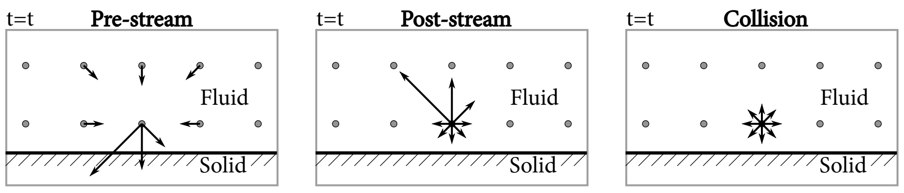


Figure 11: Illustration of the free-slip boundary condition on a D2Q9 lattice at the bottom wall.

The free-slip boundary condition, also called *symmetry* boundary condition, is that of a fixed wall where the fluid is allow to freely slip on it, i.e. the velocity at the boundary is defined by the following macroscopic equations,

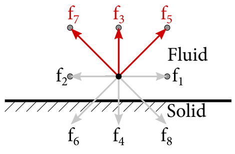
$$\vec{u} \cdot \vec{n} = 0 \tag{97}$$

$$\frac{\partial \vec{u}}{\partial n} = \vec{0} \tag{98}$$

where $\vec{n}$ is the normal direction to the boundary and n the corresponding axis (either x, y or z). Hence there is no flow across the boundary and the gradient of the velocity is zero at the boundary (as if the velocity was symmetric).

This boundary is efficiently implemented in LBM by reflecting the distribution functions along the boundary, as depicted in figure 11. This type of boundary is required for two dimensional axisymmetric flow models [229, 230, 231] or can simply be used to allow the flow to slip at the wall. As for the bounce-back scheme, the free-slip boundary can have a *wet* and *dry* implementation, and both require the wall normal. The free-slip can also be combined with bounce-back to achieve a partial slip boundary condition [232].

3.2.5 Zou-He



Unknown distributions at the bottom boundary are represented in red.

As the name suggests, this boundary condition was proposed by Zou and He [225], its principle is to modify the unknown distributions at the boundary, i.e. those coming from outside of the domain, so that the recovered density (or velocity) is the desired one. This issue is that there are usually more unknown distributions than moments, hence more unknowns than equations. This result in an under resolved system and additional equations are required to close the system. In their method, Zou-He create more equations by applying a bounce-back on the non-equilibrium part of the distributions. The proposed scheme can be directly applied to the D2Q9 and D3Q15 lattices to set either the velocity or density at the boundary, but requires significant changes for larger lattices which makes it difficult to implement for a D3Q19 lattice [233].

The following demonstrates how to apply the Zou-He boundary on a bottom wall with a fixed velocity for a D2Q9 lattice. There are three unknowns distributions, f_3, f_5 and f_7, following the numbering convention of section A.1 and the density at the boundary (ρ) is also an undetermined so there is a total of 4 unknowns. There are 3 equations coming from the definition of the density and velocity, in the following equations unknowns are coloured in red,

$$\begin{aligned} \rho &= f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 \,, && (99) \\ \rho u_x &= f_1 - f_2 + f_8 - f_6 + f_5 - f_7 \,, && (100) \\ \rho u_y &= f_3 + f_5 + f_7 - f_4 - f_6 - f_8 \,. && (101) \end{aligned}$$

The density can be obtained right away by combining equations (99) and (101),

$$\rho = \frac{1}{1 - u_y} \left(f_0 + f_1 + f_2 + 2 \left(f_4 + f_6 + f_8 \right) \right) \tag{102}$$

An additional fourth equation is introduced by considering the bounce-back rule on the non-equilibirum part of the particle distribution functions normal to the boundary,

$$f_3 - f_3^{(eq)} = f_4 - f_4^{(eq)} \tag{103}$$

where $f_3^{(eq)}$ and $f_4^{(eq)}$ can be computed from the velocity and density.

The system of equation is now close and the three unknown distribution functions can be obtained as

$$f_3 = f_4 + \frac{2}{3}\rho u_y \,, \tag{104}$$

$$f_5 = f_6 + \frac{\rho}{2}\left(u_x + \frac{1}{3}u_y\right) + \frac{1}{2}\left(f_2 - f_1\right) , \tag{105}$$

$$f_7 = f_8 + \frac{\rho}{2}\left(-u_x + \frac{1}{3}u_y\right) - \frac{1}{2}\left(f_2 - f_1\right) . \tag{106}$$

The boundary nodes in this method are wet, i.e. they are also fluid nodes, thus the collision step must be applied on them. The strength of this method is its numerical accuracy but its weakness is its deficient stability for high Reynolds number.

3.2.6 Ho-Cheng-Lin

While the Zou-He boundary condition is convenient for the D2Q9 lattice (and usable for the D3Q15), its implementation for the popular D3Q19 lattice is significantly more cumbersome [233]. The boundary proposed by Ho-Chen-Lin [217] decomposes the unknown distribution functions as a combination of their equilibrium and a corrector,

$$f_i = f_i^{(eq)} + w_i \vec{Q} \cdot \vec{e}_i \tag{107}$$

where $\vec{Q} = (Q_x, Q_y, Q_z)$ is to be determined. This effectively reduces the number of unknowns to 3.

Ho-Chen-Lin boundary can be implemented in four steps:

1. For a pressure boundary, i.e. $\rho = \rho_0$, the first step is to compute the velocity normal to the boundary $(u_\perp)$. If the distribution functions are written in the form $f_{(i,j,k)} = (i, j, k)$ where $i, j, k \in \{-1, 0, 1\}$ then the equation for the density and velocity are

$$\rho = \rho_0 = \sum_{j,k} f_{(-1,j,k)} + \sum_{j,k} f_{(0,j,k)} + \sum_{j,k} f_{(1,j,k)} \,, \tag{108}$$

$$\rho u_\perp = \sum_{j,k} f_{(1,j,k)} - \sum_{j,k} f_{(-1,j,k)} \,, \tag{109}$$

where the unknowns are $u_\perp$ and $\left\{f_{(1,j,k)}\right\}$ $j, k \in \{-1, 0, 1\}$, and the two equations can be combined to solve $u_\perp$ as

$$u_\perp = 1 - \frac{1}{\rho_0}\left(\sum_{j,k} f_{(0,j,k)} + 2\sum_{j,k} f_{(-1,j,k)}\right) . \tag{110}$$

For an inlet boundary, i.e. $u_\perp = u_0$, ρ can be solved instead as

$$\rho = \frac{1}{1 - u_\perp}\left(\sum_{j,k} f_{(0,j,k)} + 2\sum_{j,k} f_{(-1,j,k)}\right) . \tag{111}$$

2. Once the density ρ and velocity $\vec{u}$ are known, the equilibrium distribution functions $f_i^{(eq)}$ can be calculated.

3. By replacing the unknown distribution functions in the 3 momentum equations by the expression (107), the system is reduced to 3 unknowns Q_x, Q_y and Q_z which can be solved analytically.

4. Once $\vec{Q}$ is known, the unknown distribution can be evaluated with equation (107).

3.2.7 *Interpolated Bounce-Back*

This boundary scheme is based on the bounce-back scheme (i.e. the distribution functions are reflected at the wall to their incoming directions, see section 3.2.3. In the bounce-back scheme the boundary is always located half-way between two nodes (even in the so-called *full-way* bounce-back). The interpolated bounce-back (IBB) uses first or second order interpolations with the neighbour nodes to reconstruct the value of the distribution function at a location that does not coincide with a grid node. The method is mostly designed for fixed boundary that are not aligned with the grid and can be extended to moving boundaries via an additional term [212, 221].

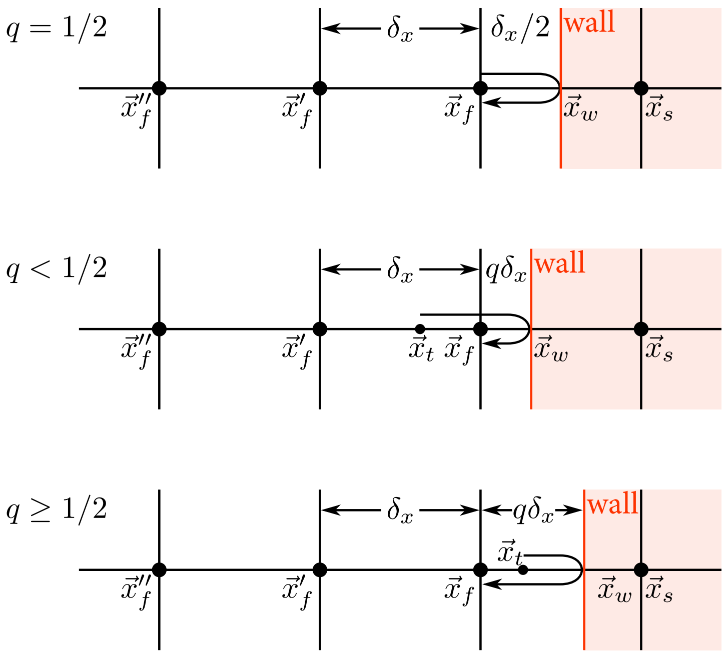


Figure 12: Illustration of the interpolated bounce-back boundary condition in one dimension for a wall located between two grid sites. $\vec{x}_t$ is the original location of the distribution function that ended at $\vec{x}_f$ after doing a full bounce-back on the wall.

In the IBB, the location of the wall can be at any place place between two grid points. The bounced-back distribution function at the last fluid node is recomputed based on the location of a virtual fluid distribution function having bounce-back on the wall, see $\vec{x}_t$ on the figure 12, using some interpolation between the distribution functions of that node and of its neighbour(s).

For each lattice link crossing the boundary, the distribution functions coming from inside the wall ($f_{i'}(\vec{x}_f)$) is undefined after the streaming step and needs to be defined using an interpolation of the opposite distribution at the wall $f_i(\vec{x}_f)$ and its closest neighbours $f_i\left(\vec{x}'_f\right)$ and $f_i\left(\vec{x}''_f\right)$, where $\vec{e}_{i'} = -\vec{e}_i$. The formula used for its definition depends on the distance between the wall and the last fluid node q (see figure 12)

$$f_{i'}(\vec{x}_f, t+1) = \begin{cases} 2q f_i(\vec{x}_f) + (1-2q) f_i\left(\vec{x}'_f\right) + 2w_i\rho\vec{e}_i \cdot \vec{u}_w, & q < 1/2 \\ \frac{1}{2q} f_i(\vec{x}_f) + \frac{2q-1}{2q} f_{i'}(\vec{x}_f) + \frac{w_i\rho}{q}\vec{e}_i \cdot \vec{u}_w, & q \geqslant 1/2 \end{cases} \tag{112}$$

The final term is introduced to account of an optional non-zero wall velocity $\vec{u}_w$. The above formula is based on a first order interpolation and only uses the first neighbour $\vec{x}'_f$. A second order interpolation using the second closest neighbour $\vec{x}''_f$ can be used instead, and is shown to yield higher accuracy [212].

$$f_{i'}(\vec{x}_f, t+1) = \begin{cases} \begin{array}{l} q(1+2q) f_i(\vec{x}_f) + \left(1-4q^2\right) f_i\left(\vec{x}'_f\right) \\ \quad -q(1-2q) f_i\left(\vec{x}''_f\right) + 2w_i\rho\vec{e}_i \cdot \vec{u}_w \end{array} & q < 1/2 \\ \\ \begin{array}{l} \dfrac{1}{q(2q+1)} f_i(\vec{x}_f) + \dfrac{(2q-1)}{q} f_{i'}(\vec{x}_f) \\ \quad -\dfrac{(2q-1)}{(2q+1)} f_{i'}\left(\vec{x}'_f\right) + \dfrac{2w_i\rho}{q(2q+1)}\vec{e}_i \cdot \vec{u}_w \end{array} & q \geqslant 1/2 \end{cases} \tag{113}$$

Although the algorithm is presented for a one dimensional configuration, it can still be applied to 2D and 3D simulation, as long as the exact location of the intersection between the wall and the lattice can be computed. For irregular shapes, this intersection might be difficult to obtain.

Although the original papers showed that the interpolated bounce-back improved over the standard bounce-back implementation for a Poiseuille flow [221] and for a moving cylinder [212], it is not better in all circumstances, Obrech [234] showed no significant differences in the obtained Strouhal number for the flow past an inclined flat plate.

As the object is moving, solid nodes becomes fluid nodes, the distribution functions of these nodes needs to be reconstructed to describe a valid fluid node. The method to compute the values of the distribution function is not unique. Lallemand et al. [212] proposes to update solid nodes as if they were fluid nodes moving at the speed of the solid or to use yet another extrapolation formula.

It should be noted that the use of the neighbour node for the interpolation breaks the locality of the LBM. Accessing neighbour distribution functions might slow down the computations and the second order interpolation might be difficult to perform in a porous media, where solid nodes might be separated by less than three fluid nodes. Nevertheless, the IBB can still be implemented efficiently on the GPU [235].

3.2.8 *Immersed Boundary Method*

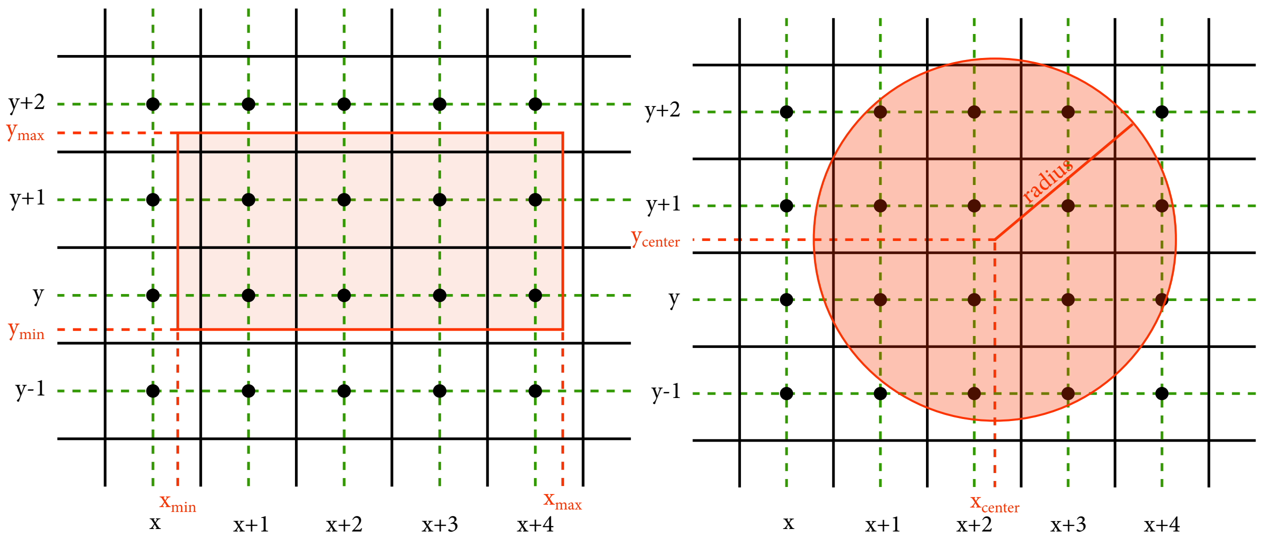


Figure 13: Example of the cell coverage ratio for a two dimensional box and for a disk, it is computed per cell as the ratio of the red area to the area of the cell (equal to one).

As introduced in section 2.3.6, the immersed boundary method (IBM) was originally designed to handle fluid-structure interaction of a deformable structure for classical CFD methods [116]. An equivalent implementation exists for the LBM (see [119] and references therein) that can be for moving and deformable boundaries or simply boundaries that are not aligned with the simulation grid (as the interpolated bounce-back). The momentum transfer is achieved through a secondary collision operator which strength depends on the *cell coverage ratio*, i.e. the ratio of the volume of solid within a cell to the volume of the cell itself. The method has several advantages over the interpolated bounce-back:

- all the operations are local to the node, no need to interpolate between neighbour nodes,
- the normal at the wall is not required (all the distribution functions of a boundary node are affected by the scheme),

- easier implementation.

The algorithm makes use of the cell coverage ratio $B_n(x, y, z)$ that is the amount of solid within the node at location (x, y, z). $B_n = 0$ corresponds to a pure fluid node and $B_n = 1$ corresponds to a pure solid node. As an example, the following formulae shows how to compute $B_n(x, y)$ for a two-dimensional simulation where the solid is a box defined by its two extreme corners (x_{min},y_{min}) and (x_{max},y_{max}), as shown on figure 13. A node at location (i, j) is considered to be inside the box (or on the boundary) if $(i + 1/2 \geqslant x_{min})$ and $(j + 1/2 \geqslant y_{min})$ and $(i - 1/2 \leqslant x_{max})$ and $(j - 1/2 \leqslant y_{max})$, in which case the cell coverage ratio is the product of the coverage along each axis. For instance, $B_n(x, y) = \left(x + \frac{1}{2} - x_{min}\right) \cdot \left(y + \frac{1}{2} - y_{min}\right)$. As for IBB, the evaluation of B_n for complex object geometry is not always simple and can become the most challenging component of the algorithm, figure 13 shows that even an object as simple as a disk can have a complex coverage. In such cases, some strategies can be adopted such as cell decomposition or polygonal approximation [119].

The algorithm for the IBM is as follows:

1. Compute the cell coverage ratio $B_n(x, y)$
2. Stream the distribution functions as usual
3. Compute the density ρ, velocity $\vec{u}$ and corresponding equilibrium $f_i^{(eq)}(\rho, \vec{u})$ as usual.
4. Compute a second equilibrium $f_i^{(eq)}(\rho, \vec{u}_p)$ based on the object velocity $\vec{u}_p$.
5. Compute an additional collision operator, Ω_i^s, for each pair of opposite distribution functions $\vec{e}_i$ and $\vec{e}_{i'} = -\vec{e}_i$.
$$\Omega_i^s = \left(f_{i'} - f_{i'}^{(eq)}(\rho, \vec{u}_p)\right) - \left(f_i - f_i^{(eq)}(\rho, \vec{u}_p)\right) \tag{114}$$
6. Relax toward the equilibrium using the usual collision operator, i.e., $\left(f_i - f_i^{(eq)}(\rho, \vec{u})\right)/\tau$ for BGK and the secondary collision operator Ω_i^s weighted with B_n
$$f_i^{(new)} = f_i - (1 - B_n)\left(\frac{\left(f_i - f_i^{(eq)}(\rho, \vec{u})\right)}{\tau}\right) + B_n\Omega_i^s \tag{115}$$
7. If the fluid is submitted to external force (like gravity), then this force is multiplied by $(1 - B_n)$ before being added.

The inside of the solid is simulated as a fluid which can be the cause of instabilities; a possible solution is force the inside solid nodes to the equilibrium, i.e. $f_i^{(new)} = f_i^{(eq)}(\rho, \vec{u}_p)$

3.2.9 *Further Reading*

Boundary conditions conditions in LBM is a difficult topic, the accuracy and stability of a specific implementation can vary with the Reynolds number of the flow [209]. Moreover, the simplicity of the implementation and its applicability to multiple lattice types as well as edges and corner nodes must be considered. Surprisingly, the force equilibrium scheme (3.2.2) provides a simple and stable way to implement boundaries while achieving a reasonably good accuracy. Hence, it offers a quick way to set up a working simulation before trading for a more accurate boundary scheme.

While the boundaries described in this chapter fulfil the requirements for the applications studied in chapters 6, 7 and 8, a few other techniques are worth investigating. Such as, the regularised boundary condition that uses the value of macroscopic moments at the wall [156], the non-reflecting boundary scheme to destroy the pressure waves that can otherwise pollute the results [213], non-local methods that interpolates the relevant quantities from the neighbour nodes [222].

In the case of a LES simulation, the boundary condition should include a modified model for the sub-grid stresses at the wall, see [176, 188].

4 OPTIMISED IMPLEMENTATION ON GPU

The previous chapter gave a basic overview of the implementation of the lattice Boltzmann method. This chapter discusses in more detail the implementation of the method on graphics processing unit (GPU) and the optimisation techniques developed during this work in order to achieve the highest computational throughput.

Firstly, the reader is introduced to a short reminder on the history of parallel computing, as well as a brief introduction to the main principles of programming on GPU. The following sections dives into the specifics of the implementation and optimisation of a LBM kernel for a GPU, before discussing other related subjects such as real-time visualisation, multi-GPU programming and automated code generation.

The results on the resulting computational efficiency after optimisation of the program are presented in the next chapter.

4.1 A BRIEF HISTORY OF GPU

From the first calculating devices, like the abacus in the antiquity to the widespread use of electronic computing devices in today's society, the world of computers has been through important changes.

The term "computer" used to refer to "human computer" as in a person who computes, often as a job, and this usages carried on up to the second world war, where human computers played an important role in the war effort.

The first mechanical calculators, like Pascal's calculator, were invented in the 17th century. And the first electronic programmable computers, using vacuum tubes emerged in the 1940s. After the invention of transistors, followed by that of integrated circuits and finally that of microprocessors, the computer landscape has evolved tremendously throughout the last 60 years. Nowadays, computers can be found everywhere, in various shapes and sizes, from smartphones to supercomputers.

While most computers are based around a single central processing unit (CPU) that performs operations in a sequential order, there has been a shift over the recent years towards multi-core processors that divide the work amongst several cores (in parallel). This is mostly caused by the limits in core frequency, as the frequency of a core increases it is able to perform more operations per second, but its power usage and thermal envelope increases. As a result, today's frequency (in the GHz, i.e. 10^9 operations per second) have reached a limit. A GPU is a recent type of processing unit that focuses on massively parallel computations.

The first GPUs appeared in the late nineties to accelerate simple graphical operations, these were not programmable in the sense that they were using a fixed pipeline designed to compute vertex projection and triangle rasterisation in a specific, pre-defined way, hard-coded on the hardware. GPUs became programmable in 2001, with the introduction of *shaders*, short programs that allows to change how vertexes, triangles and textures are handled, followed in 2004 by their formalisation within the OpenGL programming language. It is around then that the first attempts to implement the LBM on GPU was made with the contribution of Li et al. in 2003 [236]. At the time, the implementation was convoluted, as the distribution functions had to be mapped to different textures, where each texture could store 4 distributions as a red, green, blue and alpha (transparency) components. Nevertheless, they successfully implemented both a D2Q9 and D3Q19 BGK model, achieving a speed-up of up to 56 compared to a CPU of the time. Their work was latter extended[237] to a cluster of 32 GPUs (NVIDIA GeForce FX 5800 Ultra) for the simulation of the air flow around a city at a resolution of $480 \times 400 \times 80$ (which would be attainable nowadays on a single GPU).

The initial release of CUDA in 2007 popularised GPU computing by exposing a programming model close to the C programming language, i.e., more convenient for engineers than a shader language. Tölke et al.[44, 238] reported the first 2D and 3D CUDA implementation of LBM, with gains of up to two orders of magnitude over the CPU. Since then, GPU have significantly evolved in terms of performance and flexibility, and programming models have changed as well (CUDA is in version 7 at the time of writing). Nowadays, GPUs are used in many field of research and engineering, such as (but not limited to) computational fluid dynamics[239], deep learning neural networks[240], finance[241], scientific visualisation[242].

A major drawback of CUDA is that it is proprietary : only NVIDIA cards can run CUDA programs. Other frameworks aim at providing a more generic approach: OpenCL[1] (introduced in 2009 and supported by a concurrent company AMD) is similar to CUDA but can run on both NVIDIA GPU, AMD GPU, and CPU, while HMPP[2], OpenACC[3] and ArrayFire[4] open GPU acceleration capabilities through the use of preprocessor directives. The lack of universal standard in today's GPU ecosystem can be confusing. OpenCL is a good contender for that role, but as of today it is still dragging behind CUDA, with a limited number of functionality, reduced performance, and unstable drivers.

GPUs have the additional advantage of having a good ratio of performance per watt, as a result, 9 out of the 10 highest rated supercomputers in the world features GPUs; according to the Green500[5], which ranks the most energy efficient supercomputers.

1 https://www.khronos.org/opencl/
2 http://exxactcorp.com/index.php/software/prod_list/2
3 http://openacc.org/
4 http://arrayfire.com/
5 http://www.green500.org (last updated in November 2014)

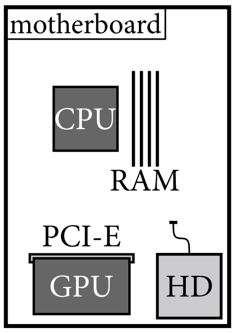
Before discussing the details of the implementation of the LBM on GPU in the next section, this section gives an understanding of how GPU works and how GPU programming differs from CPU programming.

There are three main ideas behind GPU programming, and they will be detailed in the following subsections: the GPU is physically distinct from the CPU, the GPU is a "massively parallel machine" and follows a *single instruction multiple data* (SIMD) programming paradigm.

4.2.1 *GPU programming methodology*

The GPU programming methodology is closely linked to the computer architecture. In a typical GPU-enabled computer, there is:

- a motherboard, to which each component connects to,
- a CPU, directly connected to the motherboard through a CPU socket,
- several bars of RAM memory, connected with DIMM sockets, for fast short-term storage,
- a hard drive, connected to a SATA port, for slow long-term storage,
- and a GPU, connected to the motherboard via a PCI-Express port.



Although today's GPU has more features than it used to, it still cannot function independently from the CPU. It is often regarded as a co-processor to the CPU, i.e. as another processor to relieve the CPU from some of the demanding computations. Historically, the CPU has been the one driving the system, controlling the RAM, the hard-drive and other components. It is still the case nowadays, the GPU is being controlled by the CPU with a job submission process. Thus, running a program on the GPU is typically done in three steps, as illustrated in Figure 14:

1. send the data needed for the computations to the GPU,
2. run the GPU program (called a *kernel*),
3. send the results of the computations back to the CPU.

The GPU is physically far away from the CPU, and sending data through the PCI-Express port is very slow. To overcome this limitation, the GPU is equipped with its own memory which can achieve

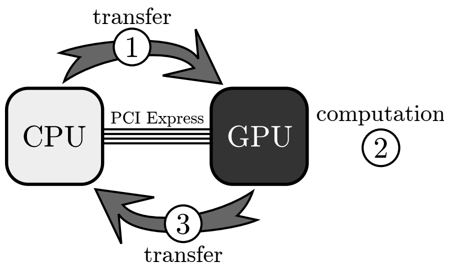


Figure 14: Using the GPU is done in three steps.

high bandwidth throughput, even higher than that of the CPU RAM. Usually, the CPU pre-processes the data, performs a memory copy from the RAM to the GPU on-board memory, schedules the GPU to run some computations on the data and then performs another memory operation to copy the data back in order to post-process the results. If special care is not taken, the memory transfers can end up taking more time than the computations themselves.

4.2.2 *Differences between CPU and GPU*

Historically, higher performances were achieved by increasing the frequency at which processors were running. But once the limit of processor frequency was reached, the only way to create faster processors was to built some that contain multiple units of computation, called cores, and sharing the workload between these cores. Nowadays, both CPUs and GPUs are multi-core processors, but the differences in their architectures result in a different operating and programming model.

Because the CPU is the central unit of the computer, it needs to be able to handle tasks of many different types, e.g. sorting files, scheduling processes, handling sockets... etc. This requires the CPU to have a large number of fast control units, that allows for quickly changing from one task to the next, and to have large amount of memory cache to reduce the unavoidable random access latencies. On the other hand, the GPU is a more specialised hardware, it can only handle one task at a time. However, modern GPUs host thousands of cores while modern CPU only have a few. So although the GPU can only tackle one task at a time, it can do it very efficiently, given that the task can be processed in parallel among all the cores. The GPU can solve some problems several order of magnitude faster than the CPU, but it needs an efficient parallel algorithm and a large amount of data to work on. This is the main distinction between *many-core* and *multi-core* architectures.

The difference between CPU and GPU parallelism becomes evident when looking at domain decomposition, for a parallel fluid simulation algorithm for instance.

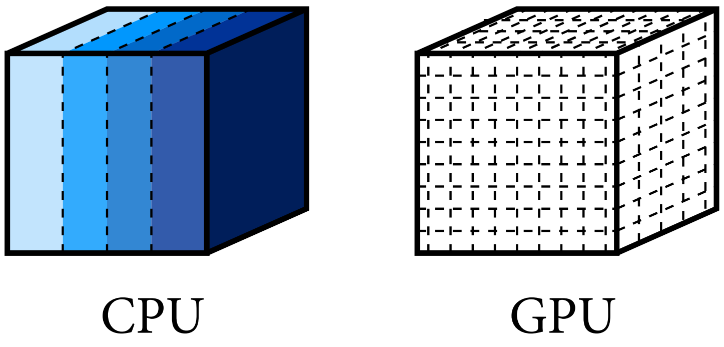


Figure 15: Illustration of domain decomposition on a CPU and on a GPU.

- On a *multi-core processor*, like a recent CPU, there are a few cores that can work on different tasks or be used cooperatively towards the same task, by sharing the work equitably between them. For example on a four-cores CPU, the program can split the simulation domain into four sub-domains of equal sizes (Figure 15). Each core would then compute a quarter of the domain. If the total number of nodes in the domain is $N_x \times N_y \times N_z$ where N_x, N_y and N_z are the number of nodes in each direction, then each core would handle $N_x \times N_y \times N_z/4$ nodes. It can be non-trivial to share the work equitably amongst the cores, especially for complex geometries, this is known as a *load balancing* issue.

- On a *many-core processor*, like a recent GPU, there are thousands of cores and each core can handle thousands of threads. And the GPU has a thread scheduling mechanism that allows to run a kernel with as many thread as necessary, even if this number is higher than what the hardware can handle at one time. Thus there is no need to divide the fluid domain into sub-domains on the GPU. Instead, each node of the simulation can be associated with a different thread (Figure 15). This is a completely different level of parallelism to what is possible on a single CPU.

In GPU computing, a thread is the smallest level of parallelism, each core handles many threads. It allows to efficiently hide the instructions and memory latencies. Threads can be seen as "virtual cores".

4.2.3 *SIMD programming philosophy*

There is another major difference between CPU and GPU which is linked with the number of control units. GPU threads and CPU cores are not designed to handle the same kind of work. As mentioned previously, CPU cores can perform different tasks from each other very efficiently, and on the other hand the GPU threads need to be all doing the same task. Although having different works amongst GPU threads is possible, it comes at price on the performance. Moreover, GPU threads are slower than CPU cores, because they are running at a smaller clock frequency and they have a different architecture. They require comparatively more clock cycles than the CPU cores to do the same operations, but on the other hand, millions of threads can be run at the same on a GPU.

In other words, the GPU follows a SIMD programming model, which stands for *Single Instruction Multiple Data*, each GPU thread executes the same kernel function, on the corresponding section of the data, i.e. the associated node, and all the threads are run in parallel. The CPU does not follow the SIMD programming model, instead it follows a MIMD model (Multiple Instruction Multiple Data) where different instructions can be executed by the cores on different parts of the data.

In conclusion,

- CPU cores can be doing very different works from each others, the amount of work done by one core shouldn't affect the speed of another core. For instance, one core could be decoding a video while the second core is busy sending file through the network. This is possible because each core can work independently from the others,
- GPU threads, however, work collectively as a group and if one of the threads is doing something different, the others have to wait for it to finish. This is why it is crucial that all the threads on the GPU follow the same instructions and access the data in the same way[6].

To use a simile, the GPU can be seen as a the head of a rake *and the data as dead leaves on the ground: if one of the branches hits a pebble, the branch will create more resistance and the whole rake will slow down.*

The GPU SIMD model comes with a special thread organisation, that needs to be defined by the programmer. The thread are organised in two levels on the GPU, see Figure 16:

1. First, the threads are grouped in three-dimensional blocks. Each block is executed on a streaming multiprocessor[7], and all threads of a same block share the resources of the streaming multiprocessor where they are running.
2. Then, the blocks are organised on a three-dimensional grid. This grid represents all the streaming multiprocessors running in parallel. The grid can be bigger than the actual number of streaming multiprocessors present on the GPU, in which case the next block will be run as soon as a free space is available on the grid.

4.2.4 *Code sample*

To summarise, the following listing shows how a typical GPU program would look like in pseudo-code. Its actual implementation in the CUDA programming language can be found in Appendix D.

6 This is know as coalesced memory access, see section 4.4.2.

7 A streaming multiprocessor is a group of GPU cores and the associated shared resources, such as register and shared memory. More details in Appendix D.

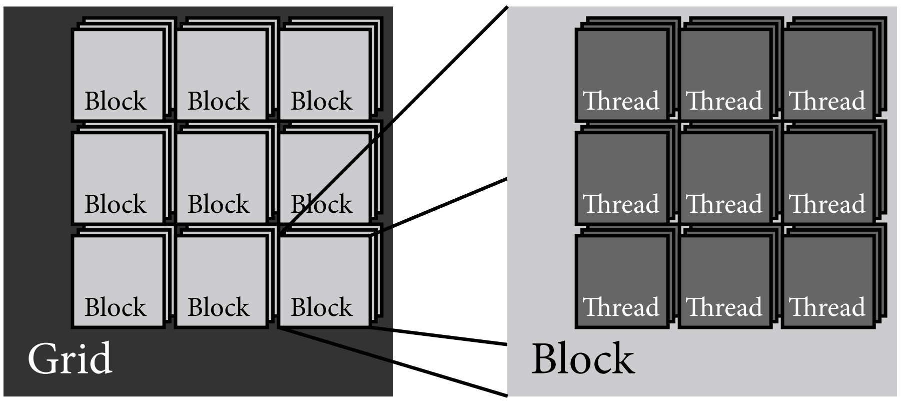


Figure 16: Threads organisation.

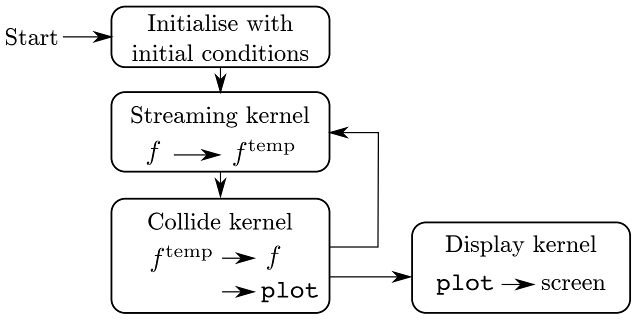


Figure 17: Core structure of the LBM program.

Listing 1: Structure of a GPU program.

```
allocate memory on the CPU and on the GPU
initialise the data on the CPU
transfer the data from the CPU to the GPU
define the thread layout (grid size and block size)
execute the kernel on the GPU
transfer the data back from the GPU to the CPU
free memory on the CPU and the GPU
```

4.3 IMPLEMENTATION OF THE LBM ON GPU

This section introduces the numerical implementation for a real-time 3D flow solver using the LBM on a GPU. The source code developed during the thesis has been written using the CUDA C language to be run on various NVIDIA GPUs. Nevertheless, all the programming techniques presented in this chapter, including the optimisation strategies presented in the next section, should be amenable to other GPU brands (i.e. AMD) and other SIMD programming languages (i.e. OpenCL).

The core structure of the LBM program running on GPU is represented in Fig. 17. At the start of the program, all the required memory is allocated on the GPU and the CPU (optionally), this includes the two set of arrays (19 arrays for a D3Q19 model) for the distribution functions $\mathbf{f}$ and the temporary distribution functions $\mathbf{f}^{tmp}$ and an array for the results to output. If used, the distribution functions stored on the CPU are initialised to their equilibrium values, then they are transferred to the GPU. Otherwise, the distribution functions are initialised directly on the GPU by a specific kernel. Once the initialisation is finished, the program can enter the main loop, which consists in the following steps: stream, collide and output the results. The streaming-step of the LBM is handled by a "streaming kernel" that reads the distribution functions stored in $\mathbf{f}$ and stores the streamed distribution functions in $\mathbf{f}^{tmp}$. The collision step of the LBM, i.e. computing the macroscopic quantities, the equilibrium distribution functions and relaxing the distribution functions towards their local equilibrium, is done by the "collide kernel" that reads the streamed distribution functions from $\mathbf{f}$ and write the collided one back to $\mathbf{f}^{tmp}$. The collide kernel also saves the quantity to display (e.g. the density or the magnitude of the velocity) into an array called `plot`. This `plot` array can later be transformed and displayed on the screen by the display kernel, called at regular intervals. In a traditional CFD software, the content of the plot array would be stored on the hard drive in order to be studied later using a specialised post processing tool. This is obviously still possible with the LBM solver, but if the simulation can be done in real-time, it is more convenient to visualise the results as they are being computed and to do the post-processing in real-time. The visualisation tool can display the results in various forms: slice moving through the domain, volume rendering, velocity profile, etc. more details are given in Section 4.5.

Note that Fig. 17 only shows the bare bones of the system and that other parts would need to be included to account for real-time user interaction or sensors input.

4.4 OPTIMISATION OF THE LBM ON GPU

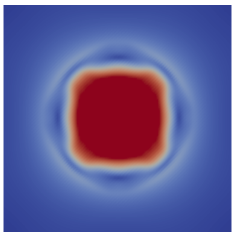
This section studies how the basic LBM program presented in the previous section can be optimised to run at the highest computational throughput on a GPU.

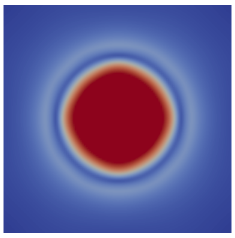
The lattice Boltzmann algorithm requires relatively more memory transactions than computing instructions, as each node operates on a large number of distribution functions and requires only a few computations in order to update them. In other words, accessing data, i.e. the distribution functions, in memory takes longer than the computations on these data. As a result, the LBM is mainly limited by the memory bandwidth of the device it is running on. There are different strategies to increase the memory throughput of LBM:

- using smaller nodes.
  Several node types can be used for three dimensional LBM:

D3Q11, D3Q15, D3Q19, D3Q27... Smaller nodes, like D3Q11, uses comparatively less distribution functions than bigger nodes which has the advantage of reducing the number of required memory accesses to update a node. On the other hand, smaller nodes span less directions in space which can lead to anisotropy in the simulation results. The D3Q19 node was chosen for most simulations, as an optimum ratio of accuracy over memory cost, but the type of node can be customised, see Section 4.7 for more details.

Comparison of the velocity fields on a cross section at the front of a jet for the D3Q15 and D3Q19 lattices. The D3Q15 shows spurious high frequency fluctuations and a clear anisotropy.


D3Q15


D3Q19

- using an improved collision operator.
  Although this does not strictly accelerate the method, it is possible to take advantage of the wasted clock cycles while waiting for memory to perform more operations toward an improved collision model with multiple relaxation time or a turbulence model. See the discussion in section 5.1.5.

- minimise memory access.
  Reading and writing data to the GPU global memory is the limiting factor of LBM, so an optimised implementation should attempt to remove all unnecessary more memory access.

- increase data coalescence.
  The memory access pattern can have a big impact on the GPU performance. With a well designed pattern, an optimised implementation can achieve significant speed-up over another implementation.

4.4.1 *Minimise memory access*

An easy and efficient way of reduce the number of memory access to global memory in a LBM program is by combining both stream and collide kernels into a single kernel, as described in Fig. 18

Indeed, when using two kernels, the program performs two read-access and two write-access to global memory (per distribution function per node and per time step). The distribution functions are read from $\mathbf{f}$, streamed to their neighbours and stored into $\mathbf{f}^{\mathrm{tmp}}$ by the streaming kernel. Then the collide kernel reads distribution functions from $\mathbf{f}^{\mathrm{tmp}}$, computes the equilibrium, relaxes and saves the new distribution functions back into $\mathbf{f}$. However, by joining the two kernels into one big kernel the number of memory access can be halved: the stream&collide kernel does not store the streamed distribution functions into $\mathbf{f}^{\mathrm{tmp}}$, instead it keeps them in registers that are directly used for the computations of the collision step, only the final streamed and collided distribution functions are stored into $\mathbf{f}^{\mathrm{tmp}}$. In the following time-step the stream&collide kernel can either read from $\mathbf{f}^{\mathrm{tmp}}$ and write to $\mathbf{f}$ or $\mathbf{f}$ and $\mathbf{f}^{\mathrm{tmp}}$ can be swapped, which is a

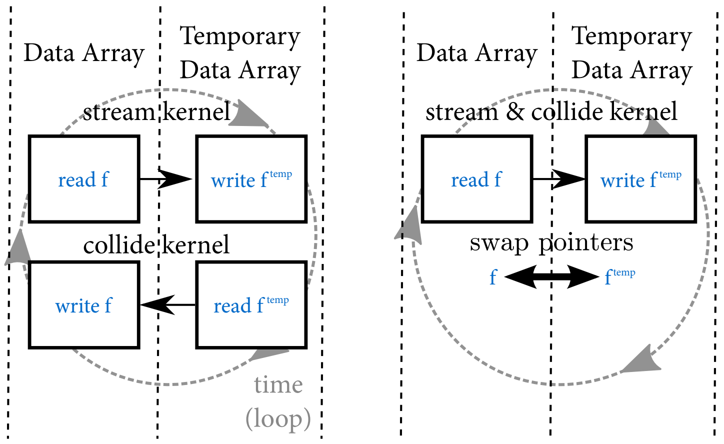


Figure 18: Blending the streaming and collision kernels into one reduces the number of memory accesses by a factor of two.

very fast operation that only requires to exchange memory pointers no memory copy is needed.

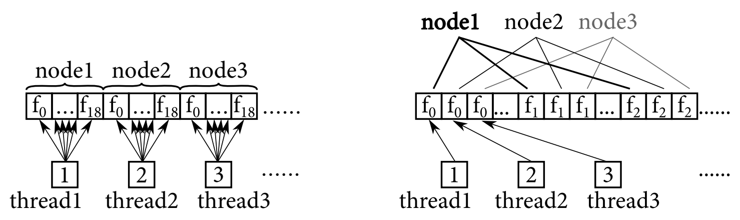
While merging the stream and collide kernels into one reduces by a factor of two the usage of the global memory bandwidth, the technique also comes with two main drawbacks:

1. increased kernel size.
   The length of the stream&collide kernel in terms of the number of lines of source code is the sum of the number of lines of each kernel. This can be an issue for readability and maintainability as lengthy functions are harder to comprehend and maintain. This is partially solved by the use of code generation techniques, as described in Section 4.7.

2. increased kernel complexity.
   Because the stream&collide kernel has to do more computations, it also require more resources and each thread running the kernel requires more registers. But the total number of register available per block is limited and if a kernel requires too many registers, then it will not be possible to fully populate the block with threads which could result in slower performance. This problem will be discussed in more details in the section 5.2.1.

4.4.2 *Increase data coalescence*

Another efficient way to increase the memory throughput of a GPU program is to ensure memory accesses are *coalesced*. A memory access pattern is called *coalesced* when all the threads in a block access consecutive memory locations. In this case, the accesses are combined into one single request by the hardware. Finding a well structured memory access can be challenging for any architecture, either CPU or


node1 node2 node3
f_0 ... f_{18} f_0 ... f_{18} f_0 ... f_{18} ......
1 2 3 ......
thread1 thread2 thread3
node1 node2 node3
f_0 f_0 f_0 ... f_1 f_1 f_1 ... f_2 f_2 f_2 ......
1 2 3 ......
thread1 thread2 thread3

(a) Array of Structures (b) Structure of Arrays

Figure 19: Two different memory access patterns for two different architectures.

GPU, but the CPU makes the impact of unstructured memory access (called *uncoalesced*) minimal by the use of large memory cache and more sophisticated control units. Nevertheless, CPU and GPU both benefits from a well structured memory access, although they prefer different ones: the CPU favours memory arranged in an a so-called Array of Structures (AoS) while the GPU favours memory aligned in a Structure of Array (SoA) pattern instead.

The following listing describe how an Array of Structure can be defined in the CUDA-C language, the 19 distribution functions are grouped together in a structure called `Node`, then all the nodes are aligned in a one-dimensional array. The layout of the distribution functions in memory and the resulting uncoalesced memory access are depicted on Fig 19a.

Listing 2: Array of Structures

```
//regroup 19 float variables together
struct Node { float f0, f1, f2, ..., f18; }
//create an array of Nx*Ny*Nz nodes
Node Lattice[Nx*Ny*Nz];
//example of how to access distribution functions
//  node (i,j,k), fifth distribution
float value = Lattice[i+j*Nx+k*Nx*Ny].f5
```

To update a node, each thread need to access all the distribution functions of the corresponding node (and some of the neighbouring nodes as well). Using the AoS pattern, each thread accesses a local area of the memory. This is perfect for a CPU implementation because the CPU updates nodes one by one and the distribution functions for each node will fit nicely in the cache memory, allowing to update the nodes efficiently. But such pattern would be very inefficient for the GPU: the accesses are not coalesced, as consecutive threads are not accessing consecutive memory locations. For example, the accesses to the distributions f_0 of consecutive nodes by consecutive threads are separated in memory space by $18 \cdot 4 = 72$ bytes (the 18 distributions f_1 to f_{18}) and cannot be grouped into a single large memory access.

They would need to be serialized into many small accesses, which would significantly slow down the execution of the kernel.

On the other hand, the Structure of Arrays pattern, as described in the Listing 3, allows for coalesced access to the memory. In this pattern, all the distribution functions corresponding to the same microscopic directions $\vec{e_i}$ are grouped together in an array, and the whole lattice is described using 19 of these arrays.[8]

Listing 3: Structure of Arrays

```
//regroup 19 float arrays of size Nx*Ny*Nz together
struct Lattice
{
  float f0[Nx*Ny*Nz];
  float f1[Nx*Ny*Nz];
  ...
  float f18[Nx*Ny*Nz];
}
//example of how to access distribution functions
//  node (i,j,k), fifth distribution
float value = Lattice.f5[i+j*Nx+k*Nx*Ny];
```

As shown in Fig 19b, the Structure of Array memory pattern does not group the set of 19 distribution functions defining a node together, they are instead spread throughout the GPU memory. This pattern would be particularly inefficient on the CPU as the 19 distributions would not fit in the cache memory, but it fits the GPU nicely, as consecutive threads access consecutive memory addresses. Therefore, the memory accesses are coalesced and can be grouped into a single large memory request (one per distribution function).

To summarise, the optimal memory access pattern for a CPU is an AoS, while the GPU need a SoA pattern. This difference in patterns is one the reasons why it is difficult to develop cross architecture programs, where the same source code base can be use for both the CPU and GPU execution. Using a code generator (4.7), the access pattern can be customised for the target architecture.

So achieving data coalescence on the GPU requires to use a SoA pattern for accessing the memory, but the thread layout as well need to be carefully chosen to adequately reflect the memory layout. The safest way to insure proper data coalescence is to use the same layout for the threads and for the memory. The 19 three-dimensional arrays containing the distribution functions are physically stored in

8 With old GPUs of compute capability 1.x, the function parameters are passed to the device via shared memory and are limited to 256 bytes. The SoA presented on Listing 3 requires $2 \cdot 19 \cdot 8 = 304$ bytes which would overflow the parameters limit when compiling for architecture 1.x. Instead the 19 arrays have to be regrouped within a single array, and accessed via index computation. This limit was removed for the architecture 2.x and higher that uses 4KB of constant memory to store the parameters.

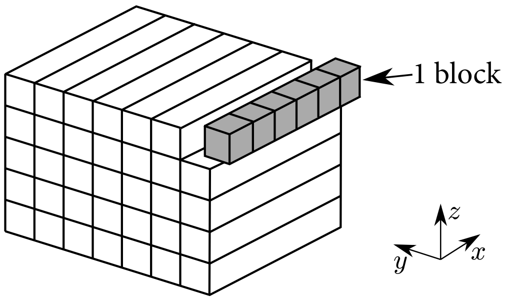


Figure 20: Thee-dimensional thread layout, using 1D blocks and a 2D grid..

memory as 19 one-dimensional arrays. The memory alignment follows the standard alignment in C programming: first aligned in the x-direction, then in the y-direction and finally in the z-direction, although this could be customised easily. The following code snippet shows how to access the element (x, y, z) of the three-dimensional array "`array`" using this alignment.

Listing 4: Three-dimensional memory index computation.

```
//compute the 1D index corresponding to (x,y,z)
int index = x + y * Nx + z * Nx*Ny;
//access the element (x,y,z) in the array
array[index] = ...
```

The threads are organised in a similar way to the memory, as shown in 20: threads are aligned along a one-dimensional block of size N_x and the blocks of threads are aligned on a two-dimensional grid of size (N_y, N_z). Threads could also be organised on 2D blocks of size (N_x, N_y) aligned on a 1D grid of size N_z, but the number of threads in a block is limited[9] so this is only possible for small simulations.

The following code snippet summarises how to define the grid and block size, how to compute the 3D position of the thread and the corresponding index to access the memory using the CUDA-C programming language.

9 There is a maximum of 1024 threads per block on the Tesla C2070.

Listing 5: Thread Organisation in CUDA

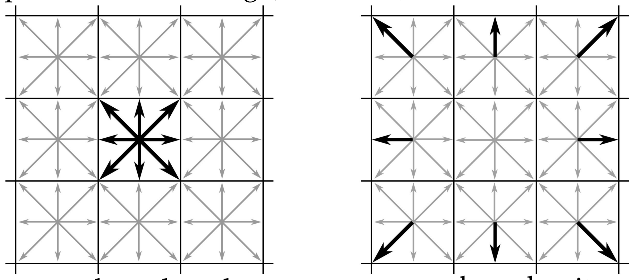
```
__global__ void ExampleKernel(real* array, int Nx, int Ny,
    int Nz)
{
  // compute the 3D position of the thread
  int x = threadIdx.x;
  int y = blockIdx.x;
  int z = blockIdx.y;
  // compute the corresponding 1D index
  int index = x + y * Nx + z * Nx*Ny;
  //access the element (x,y,z) in the array
  array[index] = ...
  ...
}

int main(void)
{
  ...
  // define grid and block sizes
  dim3 block_size(Nx, 1, 1);
  dim3 grid_size(Ny, Nz, 1);
  // launch the kernel
  ExampleKernel <<<grid_size, block_size>>> (array, Nx, Ny,
       Nz);
  ...
}
```

4.4.3 *The streaming issue*

Although using coalesced memory enables an important gain in memory bandwidth, with a corresponding increase in the efficiency of the program, the nature of the LBM algorithm makes it impossible to ever be fully coalesced. Indeed, the streaming-step requires one node to access the distribution functions of the neighbouring nodes, hence breaking the coalesced access pattern. Therefore, the streaming-step is the most critical step of the whole LBM algorithm and will usually represent most of the execution time of the stream&collide kernel.

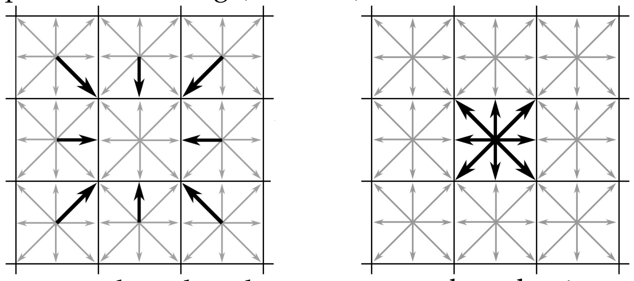
However, the inefficiency of the uncoalesced memory accesses of the streaming-step can be reduced by using a slightly different version of the streaming step. On the GPU, uncoalesced reads memory access are usually faster than uncoalesced write[243], so an efficient LBM implementation can take advantages of this by replacing the uncoalesced writes by uncoalesced reads during the streaming step. Indeed the streaming step can be implemented in two equivalent ways:

- push-out streaming (SLOWER)



  The distribution functions are pushed from the centre node to the adjacent nodes. This pattern involves local reads (i.e. coalesced) of the distributions and non-local writes (i.e. uncoalesced).

- pull-in streaming (FASTER)



  The distribution functions are pulled from the neighbouring nodes to the centre node. This pattern involves non-local (i.e. uncoalesced) reads and local (i.e. coalesced) writes.

Because uncoalesced reads are faster than uncoalesced writes the first streaming method 'push-out' is slower than the second method 'pull-in', while being equivalent. Numerical experiments have show implementing a pull-in streaming results in a speed-up of 10% in the total execution time on the Tesla K40, see section 5.1.6. The speed-up depends of the GPU architecture though, and older GPUs, with compute capability 1.x should suffer the most from the uncoalesced memory access, as they do not have cached global memory to hide them.

4.4.4 *Branch Divergence*

Another unavoidable source of performance lost on the GPU is the *branch divergence* (also called *thread divergence*). The use of flow controls (such as an *if condition* for example) can be used inside a CUDA kernel to request the threads to execute different instructions, but doing so goes against the SIMD principle (i.e., each thread executes the same instructions on a different section of the data, see subsection 4.2.3) and causes the execution paths of the threads to diverge, with a significantly slower computational efficiency. So flow controls

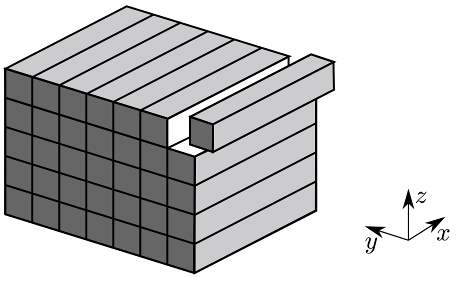


Figure 21: Effect of boundaries on branch divergence. Light gray boundaries do not introduce thread divergence, thus they are fast, but dark gray boundaries cause divergence and are slow.

slow down the program and should be avoided whenever possible, but they are always required to implement boundary conditions.

Listing 6: Boundary implementation and flow control.

```
if ( thread is on the boundary )
then ( special boundary code )
else ( generic bulk code )
```

In practice, the impact of thread divergence on the performance of the LBM on GPU depends on several factors, such as

- the GPU architecture; newer GPU are more efficient at flow control (50% slow down for the Tesla C2070 vs 30% for the newer Tesla K40, see section 5.1.7),
- the differences between the branches; in some cases, it is possible to modify the code to increase the similarities between the branches and this can aid reduce the performance loss,
- the location of the divergence; the divergence only appears within the same group of threads and not between multiple groups, hence it can sometimes be avoided by reorganising the thread layout.

In fact, blocks are further divided by the GPU at run-time into groups of 32 threads (called a warp) and the divergence only occurs at that level.

With the thread organisation described previously, i.e. block of threads aligned with the x-axis, boundary conditions aligned with that axis do not cause divergence as all the threads within the same block follow the same execution path. For example, in the case of the boundary for the face $y = 0$, all the threads within the same block have the same y-coordinate, thus they either execute the special boundary or the generic bulk code, but never both. In summary, boundary along the x-axis are efficient but those perpendicular are slow, this is illustrated by the figure 21.

4.4.5 *Other optimisations*

The reasonably simple optimisation techniques presented previously are sufficient to achieve very high performance on modern hardware. For instance, the BGK LBM D3Q19 isothermal code achieves 96% of the effective memory bandwidth using these optimisations (see section 5.1.4). On older hardware however, the cost of misaligned memory access used to be more important, and some additional optimisation techniques have been proposed[244, 243, 245, 44], such as using the shared memory to perform the streaming-step in the uncoalesced directions. Nowadays though, the improvements on the control units and the automated caching of data removes the need for such optimisations and simplifies the programming. Actually, keeping the code source simple by avoiding unnecessary complexity is often a good practice to improve performance.

The KISS principle, for "keep it simple stupid", is a famous design principle in computer science.

Bailey et. al [211] proposed a modified pattern for the streaming step that removes the need for a temporary array (see section 4.3), hence reducing the memory need by a factor of two. In their implementation, the "A-A" pattern requires two kernels, one kernel is executed for odd-iterations and performs a combined stream-collide-stream on a single array, the second kernel executes on even-iterations and only performs the collision step. This technique allows to simulate larger domain for a given memory size, but it does come at the price of higher programming complexity and slightly longer execution time. Moreover, the largest resolution allowing for real-time capability (large resolutions cannot be simulated in real-time, see section 5.3) can fit in the device memory.

In most optimised LBM GPU implementation, the CPU is left with no work to do apart from scheduling kernel execution. It is possible to consider using these wasted CPU cycles to compute a small part of the simulation domain, in a sort of hybrid CPU-GPU computing. This is exactly what Ye et al. [246] attempted, but the massive difference in CPU and GPU performance makes the added performance improvement marginal at best. The 30% improvement presented in their paper was only possible because the presented GPU implementation was really slow and unoptimised.

Some additional tricks and tweaks to get the most out of today's GPU are discussed in section 5.2.

It is worth mentioning that NVIDIA provides GPU programmers with useful tools that can analyse the program and generate reports on kernel execution time, memory usage and more to advise on where to concentrate the optimisation efforts. The available tools are `nvprof` that generate profiling data that can be imported into the NVIDIA visual profiler (see section 5.2.1 for an example) and `cuda-memcheck` and `cuda-gdb` that allow respectively to debug

one time-step of LBM	1.4 ms
copy results from GPU to CPU	6.8 ms
write results to the disk	1700 ms
display results with OpenGL	0.1 ms

Table 2: Time scales involved in computing and storing a 3D LBM simulation at 128^3.

memory leaks and kernel execution. All these tools are regrouped within an Integrated Development Environment (IDE) called NSight[10].

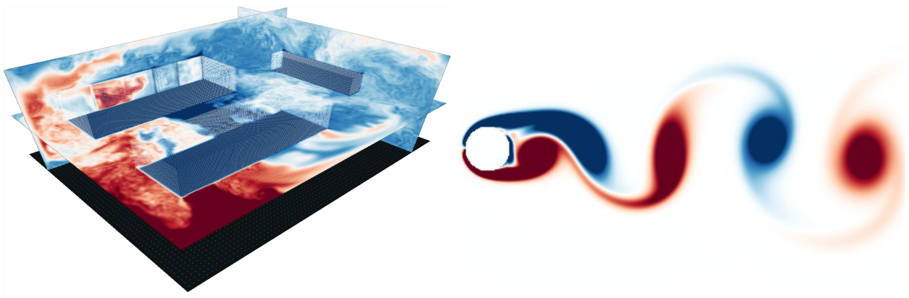
4.5 REAL-TIME INTERACTIVE VISUALISATION

In the context of real-time fluid simulation, as introduced in section 1.3, it is both more convenient and efficient to visualise the results as they are computed rather than storing them to a file for later post processing. This has many advantages such as interactivity, responsiveness, computational steering... but it is also a requirement to maintain a real-time capability for large simulations.

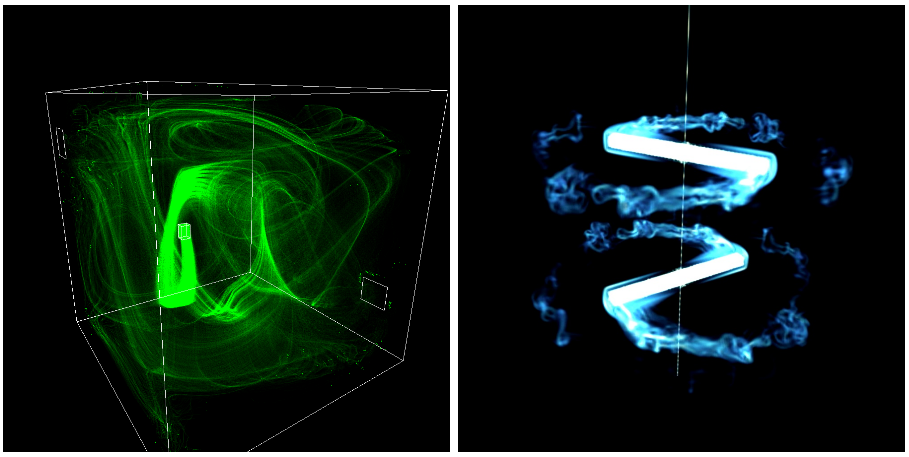
Indeed, CFD is notorious to generate a large amount of data, and storing then processing these data can be a challenge, this is closely related to field of research known as *big data*. This is again an issue of memory bandwidth, as storing the data on a disk is a slow process, which can easily take longer than the computations. To give an example of the differences in order of magnitudes between computing and storing, the table 2 compares some characteristic times based on the optimised D3Q19 LBM program described in this section and the measured performance (see chapter 5) for a cubic resolution of 128^3. At this resolution, approximately 700 time-steps are computed per second which generate 24 GB of data that would take 20 minutes to store to the disk. It is clear that storing every time-step is not an option, which is why some time-steps are usually “skipped” and the simulation is saved at regular interval instead. Even so, saving only every 1200 time-steps the program would be spending as much time on the computations as on the the writing. Of course, it is possible to omit as many time-steps as wanted, but then some important physical phenomena might happen in between two saves and be missed.

To remedy this problem, it is possible to integrate an efficient data visualisation with the computations as the data are already on the GPU (and the GPU is, after all, designed for rendering images). Hence, for this work, a simple OpenGL window is integrated with the running simulation that allows for the real-time visualisation of the flow features as well as some interactivity with mouse and keyboard inputs, all at a limited cost on the performance.

10 http://www.nvidia.com/nsight


(a) Slice extraction of the temperature field in a data centre.

(b) Vorticity field around a cylinder.


(c) Streamlines visualisation in a room using particle advection.

(d) Volume rendering of the vorticity field around moving blades.

Figure 22: Different types of real-time fluid visualisations.

Usually, the instantaneous density, velocity or temperature field is stored into the 3D `plot` array (see 4.3), that is later on post processed to be displayed on the screen with OpenGL. But in many situations, like for turbulent flows, the averaged fields over time is also of interest, in which case these fields are stored in a separate buffer on the GPU and accumulated at every time-step. This does add some overhead to the program (one additional memory access per field) which can affect the performance.

Once the relevant physical quantities have been stored in the GPU memory, several visualisation techniques are available to provide the user with useful information on the flow structures. The figure 22 presents some of these techniques applied on actual real-time simulations.

4.6 MULTI-GPU PROGRAMMING

In the cases where a single GPU does not provide sufficient performances for real-time simulation (i.e., a simulation requiring a very large number of nodes), it becomes necessary to consider *multi-GPU* *programming*, that is, distributing the computations across several GPU running in parallel.

Multi-GPU development is essentially identical to programming for multiple CPUs, the communications require a message passing interface (MPI) library, and special care has to be taken to minimise their impact on the performance, as the inter-GPU connections has a significantly reduced bandwidth. While generic strategies for multi-CPU optimisation, such as limiting interface surfaces and overlapping communications and computations, still apply to the GPU, two NVIDIA specific functionalities allow to limit CPU overhead.

1. For two (or more) GPU on the same motherboard, the *CUDA Peer-to-Peer[247]* allows one GPU to directly access the memory of another GPU through the PCI-Express without having to go through CPU memory at all. The implementation is very convenient, after enabling the Peer-to-Peer access on each GPU, a device pointer from another GPU memory can be passed as an argument to the kernel, allowing read and write access to a distant GPU memory.

2. For a cluster of GPU (i.e. several GPU on different motherboards connected via a network), *NVIDIA GPUDirect[247]* simplifies MPI implementation to limit the amount of CPU copies. With a CUDA-aware MPI, a buffer in GPU memory can be copied into a another GPU directly (with `MPI_Send(...)` and `MPI_Recv(...)`). Although MPI does make a copy to the CPU pinned memory in the background, it avoid unnecessary copies between various CPU memories and improves performance.

These two functions were first introduced with CUDA version 4.0 and require a GPU of the Fermi architecture (or newer). The first method is significantly simpler to implement, and it was chosen for this work, as the two GPU available were in the same computer. It has been implemented for both two- and three-dimensional LBM (see section 5.1.3), achieving 96% of the performance of a perfect scaling (i.e., 1.96 times the performance of a single GPU).

Various multi-GPU implementation of the LBM can have been proposed in the litterature, a few using peer-to-peer [248] and most using MPI [249, 250, 251, 252, 253]. Domain partitioning is sometimes chosen one-dimensional [249, 250], for simplicity reasons, while a 2D and 3D splitting [251, 252] would reduce the number of nodes at the interfaces, it would also introduce edges and corners interfaces that are difficult to handle.

Using CUDA streams, it is possible to overlap memory transfer with GPU kernel computation, hence combined with an asynchronous MPI the overhead of the communications can be significantly reduced.

Gray et. al. [253] showed a 26% improvement in the computational speed for a D3Q19 LBM code running on 32 GPUs.

Running the LBM program on multi-GPU can make the visualisation more difficult. While in 2D, each GPU simply has to compute a section of the final image, in 3D each GPU needs to render the visualisation to a separate buffer, then all the buffers need to be combined and the pixels sorted based on their depth; this technique is known as sort-last parallel rendering with depth-compositing [254].

4.7 GPU CODE GENERATION

It is difficult to combine a generic program, applicable to many different models and problems, with the numerical efficiency of a specialised program, designed to solve a specific problem. This is in essence why a low-level programming language, close to the computer's architecture (like C for instance), always outperforms a high-level language, with higher level of abstractions designed to simplify its usage (like Java for instance). On one hand, the use of a high-level language allows for a simpler implementation, a better maintainability of the code over time, and an easier export of the program to different computer architectures. On the other hand, a low-level language, closely related to a specific architecture (e.g. CUDA for NVIDIA GPU) allows for a tighter relation between the source code and the machine instructions and the compilations results in better performance.

In the context of the LBM on GPU, it is difficult to have a generic source code, that can handle multiple models (incompressible, compressible, multiphase, thermal, generic boundaries...), and maintain a high-level of performance, similar to that of a specialised kernel (for instance, specialised for incompressible single-phase isothermal flow in a periodic domain). For this reason, the source code developed during this thesis is made of several quasi-independent folders, each implementing a specific model, and more complex model can be created by combining source codes from multiple folders. This has the advantage of keeping the base models simple and efficient but does add some (programming) overhead when two models have to be merged.

Another possible approach is to first define a model in high-level language, then to apply code generation techniques to generate an optimised CUDA code, that could be made as efficient as a handwritten one. This is the approach followed by the open-source LBM solver Sailfish[255].

The need for code generation is best illustrated by the (simple) example of *loop unrolling*. It is common practice for code optimisation to unroll loops whenever possible. In LBM such a loop can be found in the streaming-step, the listing 7 shows the loop over the 19 directions, where the distributions are stored in a four-dimensional array (d), (x,y,z) is the node location, f[19] stores the distribution local to the node after streaming and ex[19], ey[19], ez[19] store the coordinates of each of the 19 directions. The same loop unrolled is reported on listing 8, this method is faster because the coordinates are substituted by addition and subtraction and registers are used in place of the array f (hence removing the access to the arrays ex, ey, ez and f). In theory, a smart compiler should be able to unroll the loops by itself and to substitute the arrays by their values if the arrays are properly defined as constant. But in practice, the manually unrolled loop is usually faster, and in the worst case it has the same speed.

Listing 7: Streaming-step loop for a D3Q19 lattice.

```
for (int i=0; i<19; i++)
  f[i] = d[i][x+ex[i]][y+ey[i]][z+ez[i]];
```

Listing 8: Streaming-step loop (unrolled) for a D3Q19 lattice.

```
float f0  = d[ 0][x  ][y  ][z  ];
float f1  = d[ 1][x-1][y  ][z  ];
float f2  = d[ 2][x+1][y  ][z  ];
float f3  = d[ 3][x  ][y-1][z  ];
float f4  = d[ 4][x  ][y+1][z  ];
float f5  = d[ 5][x  ][y  ][z-1];
float f6  = d[ 6][x  ][y  ][z+1];
float f7  = d[ 7][x-1][y-1][z  ];
float f8  = d[ 8][x+1][y+1][z  ];
float f9  = d[ 9][x-1][y+1][z  ];
float f10 = d[10][x+1][y-1][z  ];
float f11 = d[11][x-1][y  ][z-1];
float f12 = d[12][x+1][y  ][z+1];
float f13 = d[13][x-1][y  ][z+1];
float f14 = d[14][x+1][y  ][z-1];
float f15 = d[15][x  ][y-1][z-1];
float f16 = d[16][x  ][y+1][z+1];
float f17 = d[17][x  ][y-1][z+1];
float f18 = d[18][x  ][y+1][z-1];
```

Clearly, the process of unrolling the loops can be automated. Also, the above hand-made unrolled version restricts the lattice links to a specific set of directions. For this purpose, a code generator was designed in the simple high-level language LUA.

An LBM code generator for GPU must satisfy the following criterias:

1. Simplicity of the syntax. The code for the source generation should be shorted and simpler than the generated code, for instance the final streaming-step in LUA looks like this:

```
for i,ei in ipairs(node.directions) do
  fi = f(i,X+ei)
end
```

2. Algebra capability. This is not part of LUA, but a simple computer algebra system (CAS) was implemented so LUA can treat strings (like the density `rho` or the distributions `f0`, `f1`, `f2`, ...) as variables and performs mathematical operations on them. Also constant expressions can be pre-computed during the generation process and simplified.

3. Vector calculus. On top of the mathematical algebra, LUA was extended to operate on vector and arrays. For instance, this allows the node velocity to be defined as

   ```
   u = SUM( node.distributions * node.directions )
   ```

4. Abstraction of the target architecture. Defining a new LBM model within the code generator does not require any knowledge of CUDA, model definition is kept close to their mathematical descrition. This simplifies the addition of new models, but also by abstracting threads and memory layout allows to tweak the target language, potentially for OpenCL or CPU.

Simple models, such as the BGK LBM, can be mathematically defined independently of the lattice structure (i.e. node type), thus it is possible to generate the code for any node type (e.g. D2Q9, D3Q19, D3Q27...) simply based on the link directions and associated weights. However, other models like MRT have different formulations depending on the node-type (because the set of moments are different). Nevertheless, the code generation is still very valuable for MRT because it allows to expand the matrix multiplication during the collision in order to simplify the expressions and avoid lenghty memory access, but also allows to run safety checks on the matrix to insure orthogonality and reversibility. Listing 9 shows the lua code and generated CUDA code equivalent to the matrix multiplication used to calculate moments from [125] in D2Q9, i.e. $|\rho\rangle = M\,|f\rangle$.

Listing 9: Moment computation for a D2Q9 MRT model, in LUA and the CUDA output.

```
// LUA code
moments = model.M * node:df_matrix()

// Generated CUDA code
float rho = f0+f1+f2+f3+f4+f5+f6+f7+f8;
float E   = -4*f0-f1-f2-f3-f4+2*f5+2*f6+2*f7+2*f8;
float E2  = 4*f0-2*f1-2*f2-2*f3-2*f4+f5+f6+f7+f8;
float Jx  = f1-f3+f5-f6-f7+f8;
float Qx  = -2*f1+2*f3+f5-f6-f7+f8;
float Jy  = f2-f4+f5+f6-f7-f8;
```

```
float Qy  = -2*f2+2*f4+f5+f6-f7-f8;
float Pxx = f1-f2+f3-f4;
float Pxy = f5-f6+f7-f8;
```

The implemented code generator is by no mean a finished product, important features such boundary conditions and template-based generation are not integrated. Although, boundary conditions can be generated seperately and then added manually to the kernel code, see section 7.3.

The code generator has many other additional advantages, one of them is the ability to generate LaTeX formula from the LUA source code that can be compiled into a pdf file for equation verification or documentation.

4.8 SUMMARY

This chapter introduced the concept of GPU with its history from simple graphics accelerators to the powerful computational units of today. The difference of its inner working and programming compared to the commonly used CPU were clarified and a particular emphasis was given to the development of optimisation techniques for the LBM on GPU. These optimisations are centred around the maximisation of the available memory bandwidth, that will be shown to be the main limiting factor for the performance on GPU in the next chapter, in which the achieved performances will be measured and discussed.

Higher level concepts were also briefly introduced, such as the need for an interactive real-time visualisation, the techniques available for multi-GPU programming and finally, the reasons and practices for code generation.

5 COMPUTATIONAL PERFORMANCE

This chapter focuses on testing and analysing the numerical performance of the LBM program on GPU and the effect of the optimisations presented in the previous chapter. Some simple "tricks" are introduced that can significantly affect the performance of the program without requiring much programming efforts. These are measured on a variety of machines and for a wide range of models. The limits in which the current performances can be sufficient for real-time CFD are discussed. The next chapter will then extend the study to the numerical accuracy of the method by comparing simulation results for well established flow problems against analytical solutions or validated CFD results.

5.1 PERFORMANCE STUDY

5.1.1 *On measuring performances*

Measuring the numerical performance of the LBM on GPU is somewhat a difficult task. Performances are mostly dependant on the complexity of the LBM model and on the capability of the GPU used for the computations, but they also depend on other parameters that are harder to control. The operating system, as well as the video driver, can have an effect on how efficiently the CPU dispatch work to the GPU, thus affecting the performance. The presence of geometry has an influence threads execution path (see subsection 5.1.7), and some simple settings can a great impact on the efficiency of the GPU5.2. Even temperatures can alter the performances.

Unless specified otherwise, all the performances divulged in this section are measured on a desktop computer comprised of two Tesla K40[1] and one Quadro 600 running CentOS 6.6 and using the CUDA toolkit version 7.0 with the NVIDIA driver version 346.59 and compiled with the GNU compiler g++ version 4.7.2, table 3 gives a list of all its components.

Motherboard	ASUS P6T7 WS
CPU	Intel Xeon W3670
RAM	6×4 GB
GPU1	Quadro 600, 1 GB
GPU2	K40c, 12 GB
GPU3	K40c, 12 GB

Table 3: Description of the test system.

To study the performance, the same benchmark test is used throughout this chapter. This test is based on an optimised CUDA kernel for the three-dimensional simulation of an isothermal single phase fluid using a D3Q19 node, the fluid is initially set at

1 The two Tesla K40 GPUs were donated by NVIDIA, through the *Hardware Grant Program*.

rest and the domain sides are using periodic boundary conditions. This is an idealised problem, but it provides a good reference for performance study.

Performances are measured in term of the number of (lattice-)nodes updated per second, usually in millions, i.e. MLups. Computing the number of MLups is straightforward,

$$\mathrm{MLups} = r \times N_x \times N_y \times N_z/\Delta t \tag{116}$$

where r is the number of recursive calls to the LBM kernel, (N_x, N_y, N_z) is the size of the lattice in number of nodes and Δt is the time in seconds taken for the computations. By default, kernel launch is asynchronous, i.e. the CPU orders some work to the GPU and does not wait for it to complete its tasks, so when measuring computation times, is important to call the function `cudaDeviceSynchronize()` which forces the CPU to wait for the GPU to finish.

The number of MLups is dependent on the optimisation level, the physical model and the lattice type. For instance, a LBM model using a D3Q15 lattice (with only 15 velocity directions) should achieve a higher MLups than the same model on a D3Q19 lattice (with 19 directions).

5.1.2 *Single GPU performances*

The performance of the LBM benchmark are measured on a variety of GPU (some are designed for gaming, i.e. GTX, and some are designed for scientific computing, i.e. Tesla), and summarised on figure 23. Interestingly, the GTX cards achieve better or similar performance as the Tesla cards, while being less expensive. This is due to the differences in memory bandwidth, it is often higher on gaming card because of higher memory clock rates, and the LBM algorithm is very memory intensive.

So far, the memory bandwidth has been the limiting factor for the performance. This could change with the future generation of NVIDIA's GPU, called Pascal, featuring 3D staked memory for increased capacity and bandwidth.

- First on the list, the Brix is handy mini-pc ($6 \times 13 \times 11$cm) sold by the company Gybabite (£700) featuring a downgraded GTX 760. While the performance of that machine are about 4 times slower than the Tesla K40, it is highly portable system that can be taken for live demonstration in conferences.
- The Tesla C2070 (first released in 2010 at £4000) is based on an older Fermi architecture, it about half as fast as the K40 in single precision but is not very good at double precision.
- The Tesla K40 (first released end 2013 at £4000) is based on the following generation of GPU, i.e. the Kepler architecture. This is the main GPU used for the thesis and it is the one featured on most of the performance studies in this chapter. It performs approximately twice as fast in single precision than in double precision.

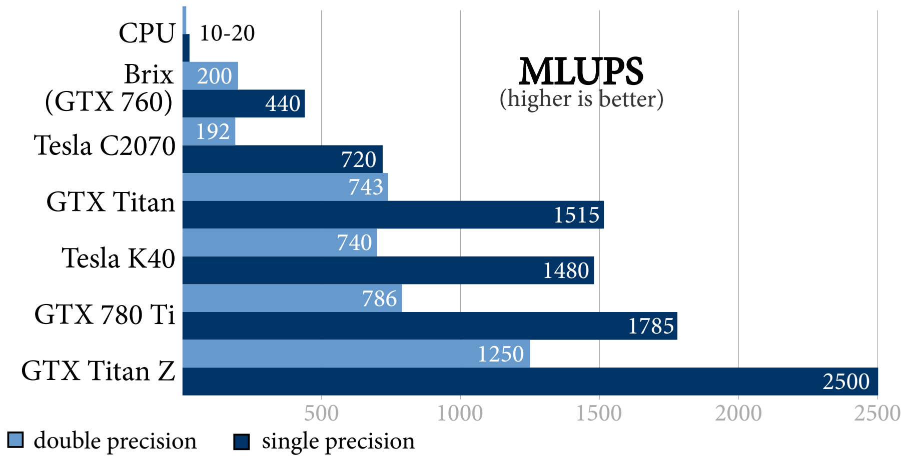


Figure 23: Performance of the D3Q19 BGK LBM benchmark program running on different architectures of GPU, in single or double precision for floating point operations.

- The GTX Titan (released in 2013 at £1000) has very similar performance to the Tesla K40, both in single and double precision, while being less expensive, the more recent GTX Titan X (March 2015, £1000) could be faster in single precision but was not tested.
- The GTX 780 Ti (released end 2013 at £700) outperforms the Tesla K40 reaching almost 1800 MLups, it is the best ratio of performance/price on this list. On the other hand, it is not optimised for double precision and it is more prone to overheating which results in fluctuating performances.
- The GTX Titan Z (released in 2014 at £3000) combines two GK110 processors on the same board, it is basically a dual GPU card. To utilise it to its full capacity, the benchmark was adapted to a multi-GPU code using peer to peer memory access (see next section). The two on-board GPUs are actually connected by a PCI-Express interface which limits its potential performance. Moreover, the MLups significantly degrades over time as it heats up, starting at 2850 MLups in single precision and stabilising around 2500 MLups, as reported on the figure. Based on the price and performance it is more interesting to invest in two GTX Titans.

Although not on the list, the newly released (June 2015, £600) GTX 980Ti should provide even higher performance than the GTX 780Ti.

Meanwhile CPU performances are of a different order of magnitude. A note on the CPU results, they do not provide a fair comparison, as the GPU CUDA code has been highly optimised during the Ph.D. while the CPU was not the focus. Here, the CUDA kernel is converted into a C++ code by transforming the thread computation into for loops and the outside most loop is made parallel over the 12 threads of the Intel Xeon W3670 using the following directive.

```
#pragma omp parallel for
```

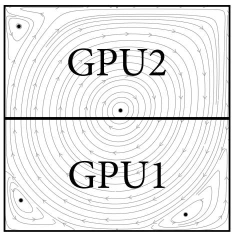
Furthermore, the kernel is split into two separate functions for the streaming and the collision, allowing for the collision function to be vectorised. The program is compiled with the Intel compiler using the following command to enable the highest level of optimisation, check on the vectorisation of the loop and enable the parallelisation with OpenMP,

```
icc main.cpp -O3 -vec-report2 -fopenmp
```

To measure the number of MLUPs, the same D3Q19 LBM model with single precision calculations and with the same 256^3 lattice size as for the GPU benchmarking is used. The CPU program achieves a performance of 6 MLups, i.e. 250 times slower than the GPU program on a Tesla K40. The streaming function, due to its non-local dependencies, cannot be vectorised by the compiler. Modern CPUs possess vector units that functions in a very similar way to GPU threads (performing operations on a array in parallel) and their usage can significantly improve the performance. As a test, the call to the streaming function can be removed from the program and it shows that the collision function alone runs at 24 MLups (although it would not give the correct physical results anymore). It is possible that the CPU performance could be optimised further by improving the vectorisation of the code and making a more efficient use of the cached memory. Still, even a highly optimised CPU code cannot compete against the GPU, McIntosh-Smith et. al. [256] reported a performance of 100 MLups on a Intel Xeon E5-2687 CPU for the same D3Q19 LBM model.

5.1.3 *Multi-GPU performances*

Although the developed multi-GPU LBM program is not fully optimised, this section reports the measured performances for a 3D simulation running in parallel across two Tesla K40 GPUs, utilising the CUDA peer to peer memory access described in section 4.6.



Domain decomposition for the 3D lid-driven cavity.

Rather than simulating an empty periodic domain, this multi-GPU benchmark simulates the three-dimensional lid-driven cavity (a problem described in section 6.2.3) on a 256^3 resolution. On a single Tesla K40, the lid-driven cavity problem performs slightly better (1535 MLups) than the periodic domain used for benchmarking because the presence of the walls avoid some of the uncoalesced memory access. With a perfect scaling and based on the single GPU performance, the highest performance possible on two GPU is double that of single GPU, i.e. 3070 MLups.

For the multi-GPU simulation, the domain is decomposed into two sub-domains, the bottom half is simulated by GPU1 and the top half by GPU2. At the interface, each GPU needs to access the distribution functions stored in the other GPU memory (via the PCI-Express port) during the streaming step. Because the memory bandwidth of the PCI-Express is significantly smaller than that of the on-board memory (8 GB/s versus 288 GB/s) it is expected to slow down the overall execution of the program compared to a perfect scaling.

PCI-Express Speed

v1.x	*4 GB/s*
v2.x	*8 GB/s*
v3.0	*15.75 GB/s*
v4.0	*31.51 GB/s*

After measurement, it is found that the program achieves a performance of 2876 MLups, which is only 3.6% slower than that of a perfect scaling. It is possible that this performance could be improved by using asynchronous memory transfers in order to overlap computations and memory access. Also, the upcoming NVIDIA NVLink technology will be a new high-speed GPU interconnect that will allow for faster transfers between GPUs.

5.1.4 *Maximum performance*

The performance of the optimised LBM GPU program presented in the two previous subsections seem satisfactory as it allows for real-time simulation (see section 5.3) with a good accuracy (see chapter 6). But is the program really optimal? Or could further optimisation improve its performance? And if so, to which extent? To answer these questions, this section looks at the maximum performance obtainable on a tesla K40 GPU based on the effective memory throughput.

As presented in the previous chapter and confirmed by the tests in subsection 5.2.1, the efficiency of the LBM implementation on a GPU is currently limited by its memory bandwidth and not by its computational power (i.e. floating point operations per second, or FLOPS) which is often the quantity advertised. Hence, an easy way to estimate the maximum performance of the program for a given GPU is to divide the bandwidth for global memory access by the number of bytes accessed per kernel call.

According to NVIDIA's documentation[2], the theoretical memory bandwidth (with the highest clocks and ECC off) for a Tesla K40 is $B_{th} = 288$ GB/s. This value is obtained by multiplying the number of memory interfaces (also known as buses) by the frequency at which they transfer data (i.e. the memory clock rate).

$$B_{th} = 2 \times 384b/8 \times 3GHz = 288GB/s \tag{117}$$

On the Tesla K40, the memory bus 384 bits wide, i.e. 48 Bytes, while the global memory is clocked at 3 GHz. The GPU memory works in a double data rate (DDR) fashion, where two data are accessed per cycle, thus the factor 2.

In practice however, the effective memory bandwidth is smaller than the theoretical bandwidth (which is the one advertised). Therefore, to establish the real memory bandwidth of the card, a test program carries out a single memory transfer of 128 MB from GPU mem-

2 http://www.nvidia.com/object/tesla-servers.html

ory to another location in GPU memory using the `cudaMemcpy` function. This process is repeated ten times to obtain an average effective memory bandwidth B_{eff}. This is the same process which is used in the *bandwidth-test* program, part of the NVIDIA SDK (Software Development Kit). Although that program uses a 64 MB by default which was found to give slightly smaller estimation of the effective bandwidth on some cards. On the Tesla K40, with ECC and Boost disabled (see sections 5.2.3 and 5.2.4 for details) the effective memory bandwidth is measured to be $B_{eff} = 240$ GB/s, that is 16.7 % smaller than the theoretical bandwidth B_{th}.

Based on the effective memory bandwidth B_{eff}, the maximum possible performance of the LBM program in MLups can be estimated by supposing that the time spent on the calculations is negligible and that the kernel achieves all of the available memory throughput. This is obviously an idealised scenario, and the calculation will only give an upper limit to the performance of the code. For example, the three-dimensional isothermal LBM benchmark involves nineteen distribution functions and each one is read and written once per time-step. In single precision, each distribution function uses 4 Bytes of memory, so there is a total of $2 \times 19 \times 4 = 152$ Bytes of memory accessed per each node per time-step. Based on this number, each node can be updated for a maximum of $240\text{GB/s}/152\text{B} = 1.579 \times 10^9$ times per seconds, hence the maximum number of nodes updated per second is 1579 MLups.

This number can be compared with the measured performance of the program, i.e. 1515 MLups, which is only 4% slower, proving that the program is definitely limited by the memory bandwidth and that any additional optimisation would only improve the performance by maximum of 4%. Doing the reverse calculation shows that the LBM kernel achieves a memory throughput of 230GB/s, which is 80% of the theoretical memory bandwidth.

A similar technique can be used to estimate the maximum resolution of a simulation based on the lattice type and the size of the available memory on the GPU. For instance, for a D3Q19 lattice, each node requires $19 \times 4 = 76$ Bytes of memory in single precision. The Tesla K40 features 12GB of memory, so it can store up to $12 \times 1024^3/76 \simeq 170 \cdot 10^6$ nodes. As the current kernel implementation requires two lattices for the streaming step (see section 4.3) the maximum resolution that could fit in GPU memory is $\sqrt[3]{170 \cdot 10^6/2} \simeq 439$. In two dimensions with a D2Q9 lattice the maximum square resolution is 13377^2.

In cases where higher resolutions are required, it might be tempting to use the CPU RAM as a buffer memory, but doing so would require the distribution functions to be transferred over the PCI Express port, limiting memory bandwidth to approximately 6GB/s (from measurements). Based on that bandwidth, the maximum achievable performance would be around 40 MLups, thus it is not recommended.

5.1.5 *Performance of other models*

Until now, the performance study considered the isothermal BGK LBM model on a D3Q19 lattice. It is expected that the LBM is memory bound on the GPU for all models and all lattice structures. Consequently, performances are strongly linked to the number of distribution functions used in a model, and less related to its computational complexity.

Prefer computations over memory access. It is faster to use a lattice with fewer neighbours (e.g. D3Q15 over D3Q19) and to use an improved collision model (e.g. MRT over BGK) to maintain the same accuracy level.

For instance, the three-dimensional thermal model used for the results in chapter 6 and chapter 7 is based on a D3Q19 lattice for the velocity field and a D3Q6 for the temperature field, so it requires 6 more read and write access to global memory and its performs performs 31 % slower than the isothermal model. On the other hand, the addition of the Smagorinsky turbulence model (see section 2.5.1) that requires many more computations but no additional memory access only slows the code by a further 8%.

The performances of each model are summarised in table 4.

Model	Performance in MLups
3D Isothermal	1518
3D Thermal	1041
3D Thermal and Turbulent	957

Table 4: Performance of various D3Q19 BGK LBM models in single precision on the Tesla K40 GPU.

5.1.6 *Effect of the streaming model*

As described in section 4.4.3, the streaming step can be seen from two different perspectives either the distribution functions are streamed from one node to its neighbours (push-out streaming) or they are streamed from the neighbours to the centre node (pull-in streaming). Physically the two views are equivalent, but computationally they are different and result in different performances.

- The pull-in method, which is the preferred method, performs some uncoalesced reads but no uncoalesced writes, and gives a performance of 1518 MLups.
- The pull-out method, on the other hand, performs no uncoalesced reads but does some uncoalesced writes, resulting in a performance of 1385 MLups, i.e. a 10% slow down of the program.

Moreover, the Tesla K40 uses the L1 and L2 cache memories to hide the latency related to uncoalesced writes, and it is likely that the slow down would be even more severe on older GPU.

5.1.7 *The issue of branch divergence*

As described in section 4.4.4, boundary conditions can create a divergence between the threads of a same block and result in a significant performance lost. In order to illustrate this problem, this section considers a worst case scenario where half of the node are solid (using the full-way bounce-back scheme) and half of the nodes are fluid. The solid nodes are organised on one node thick layers, each separated by a layer of fluid cells. These layers can be either aligned with the x-direction, which is also the direction of the blocks of threads (see section 4.4.2), or they can be aligned along the y-direction, perpendicularly to the blocks of threads. In the code, this can be easily achieved using the following expression,

Listing 10: Boundary alignment and thread divergence.

```
//Case 1 : x-aligned walls
if( y % 2 == 0 )
{
   //Bounce-back...
}
//Case 2 : y-aligned walls
if( x % 2 == 0 )
{
   //Bounce-back...
}
```

- In case 1, threads withing a block, i.e. constant x, are either solids if the block has an even y-coordinate or fluid otherwise. Hence, this type of boundary condition does not introduce divergence between threads of a same block, only between different blocks. As a result, the performance of the program is only mildly affected, with the kernel achieving at 1448 MLups, i.e. only a 4.6% decrease in performance compared with the empty domain.

- On the other hand, case 2 is a worst case scenario for branch divergence, as half of the thread within a block are solid and half are fluid. Moreover, withing each warp (i.e. a group a 32 threads within a block) there is also the same divergence. The performance is greatly affected, falling to 1025 Mlups, i.e. a 32.5% slow down compared to the empty domain benchmark.

In conclusion, is a better to align boundary conditions with thread layout whenever possible to avoid divergence. Although that is not always practical.

5.2 OPTIMISATION TRICKS AND TWEAKS

As the performance of LBM is mostly limited by the available memory bandwidth, the optimisation described in section 4.4, namely minimise memory access and increase data coalescence, are enough to achieve good performances on a GPU. But some fine tweakings of the kernel configuration and some simple tricks on the GPU settings can be applied to squeeze every last bit of power out of the GPU. The two tricks described in the subsections 5.2.3 and 5.2.4 require root access to be applied, but they are worth the investment as they are only one line long and each can improve the computational speed by 14%, resulting in a total improvement of 35%.

5.2.1 *Using the NVIDIA Visual Profiler*

NVIDIA provides useful tools to help programmers develop their GPU applications. One of these tools is Nsight[3], it comes as an add-on for Visual Studio on Windows or to Eclipse on Linux and it is a powerful tool for the debugging and profiling of a CUDA code in order to fully optimise its performance. It is well worth investing some time in learning how to use it because it can really help track a bug or find a performance bottleneck and save some time down the line. In particular, its integrated Visual Profiler can also be used independently to extract precious running time information on the GPU kernel. It was used heavily for the results of this section. When used independently, the following steps can be followed to profile a CUDA program.

1. Compile the program by passing the option `-lineinfo` to `nvcc`. This allows to access line numbers later when profiling a kernel.

2. Run the program through `nvprof` with the following command:

   ```
   nvprof --analysis-metrics -o data.prof \
           ./program_name
   ```

   The program will be run several times in various configurations to build its profile. It is expected for it to run much slower than usual during this process.

3. Import the profiling file data.prof into the NVIDIA Visual Profiler, which can be started with the command `nvvp`.

3 http://www.nvidia.com/object/nsight.html

The NVIDIA Visual Profiler allows for a guided analysis of the applications, it provides global measurements like the GPU usage or the average time taken by each kernel, but it also allows to do a detailed kernel analysis. As an example, this section looks at some of the graphs generated by the profiler during the analysis of the optimised compute kernel for the D3Q19 BGK LBM running on a K40 GPU. Another example of profiling and optimising an LBM application with nvprof can be found in reference [257].

Compute utilisation vs memory utilisation

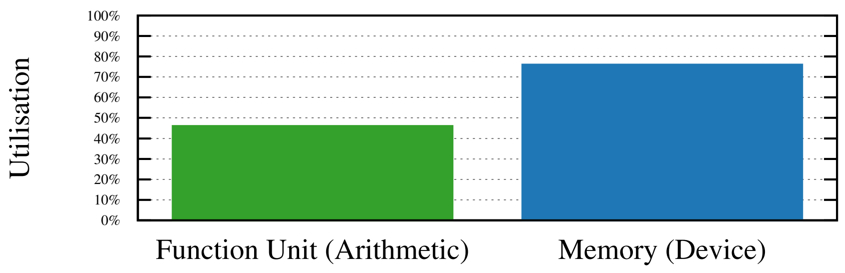


Figure 24: Percentage of the time spent by the kernel in the computations and in memory transactions.

Figure 24 shows that as expected the memory utilisation level is higher than the arithmetic one, thus the kernel is limited by the memory bandwidth. The two levels add up to more than 100% because some of the computations are overlapped with the memory access, i.e. the GPU performs some computations while waiting for the memory access to finish. Moreover, the profiler estimates the achieved global memory bandwidth at 230 GB/s (i.e. 96% of the effective memory bandwidth), confirming hand-made measurements.

Utilisation per memory type

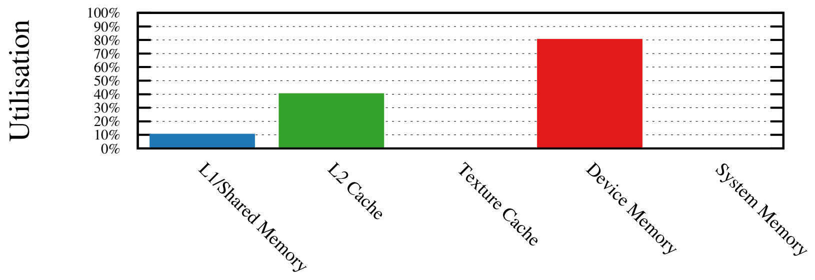


Figure 25: Ratio (in percentage) of achieved memory throughput to theoretical memory bandwidth for each type of memory.

Figure shows the different usage levels for each memory type. The constant/texture memory is not used in this kernel and neither is the system memory (CPU RAM) apart from a short time during the initialisation. The profiler detects that the kernel is limited by the bandwidth available to the device memory, as expected, and also detects 10 uncoalesced memory access caused by the streaming of the distribution functions in the x-direction. The use of L1 and L2 cache memory was automatically generated by the compiler to improve some of the memory transfers to the global memory.

Function Unit Utilisation

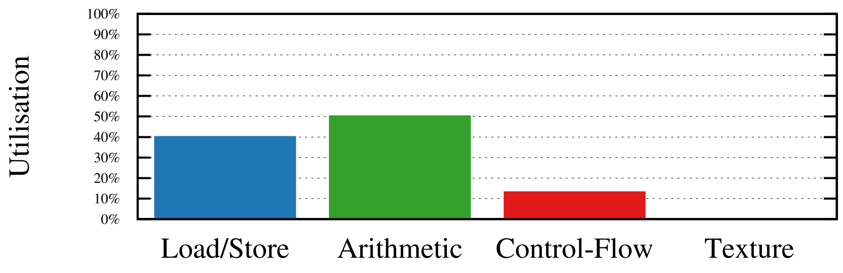


Figure 26: Utilisation level of each GPU function unit.

Different types of instructions are executed on different functions units within each streaming multiprocessor. The profiler shows that no function unit is overused, which would limit the kernel's performance. Instead, arithmetic instructions (such as add, multiply, etc.) are roughly of the same order of utilisation as load and store instructions (for all types of memory), and some control-flow instructions are needed for the periodic boundary conditions.

Instruction Execution Counts

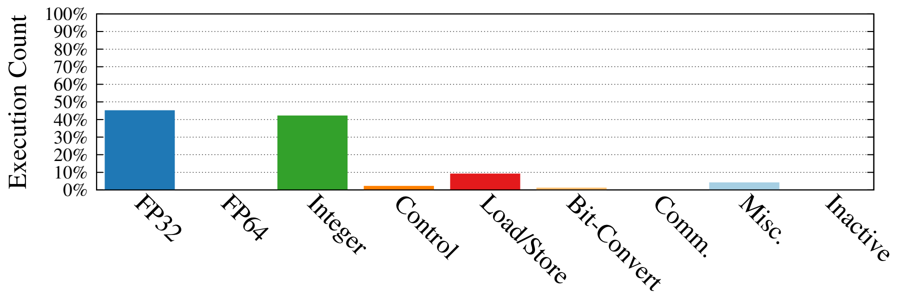


Figure 27: Percentage of execution cycles spent in each class.

The chart displayed on figure 27 shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each class, the chart shows the average percentage of thread execution cycles spent in that class. Interestingly, while the written kernel seems to mostly involve (single precision) floating point operations to compute the macroscopic quantities and the 19 equilibrium distribution functions, the index computations (with integers) end up using a similar amount of cycles. Floating points operations are more efficient on the GPU than integer operations.

Instruction Latency

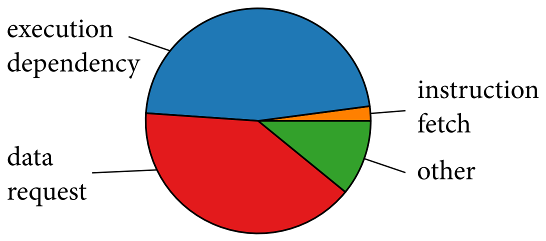


Figure 28: Stall reasons, limiting instruction level parallelism.

A good way to improve performances is to increase instruction level parallelism. Within the kernel itself, some instructions can be computed independently in parallel while some rely on the results of previous instructions and need to wait for these to finish first, this is called *instruction stalling*. The profiler allows to investigate the reasons of these stalls in order to aid the developer in restructuring the kernel's code to reduce stalling and increase instruction parallelism. According to the pie chart on figure , the performance of the LBM kernel are limited by all the accesses to global memory (data request) and by the dependencies of the computations (for instance ρ and $\vec{v}$ needs to be computed first before computing the equilibrium distribution functions).

Register Usage

The D3Q19 BGK LBM kernel uses 56 registers per thread. For this test, the domain size is 256^3 and block of threads are one dimensional and expand along the x-direction, thus the block size is 256 and the kernel needs $56 \times 256 = 14336$ register for each block. This register usage can prevent the kernel from fully utilising the GPU. Indeed, the Tesla K40 provides up to 65536 register per streaming multiprocessor (SM), so based on the kernel's need, each SM is limited to simultaneously executing 4 blocks (i.e. 32 warps). In theory, the number of registers required per thread can be reduced (either by using the `-maxrregcount` flag or the `__launch_bounds__` qualifier) to improve the occupancy of the GPU. But in practice, the achieved occupancy is already good enough (47.2%) and the performance would not be significantly improved by a slight increase in occupancy.

5.2.2 *Tweaking for the best performance*

In addition to the general recommendations given in the previous section, the following "rules of thumb" can be followed to tweak the code to achieve the best possible performance.

Use a block size that is a multiple of 32

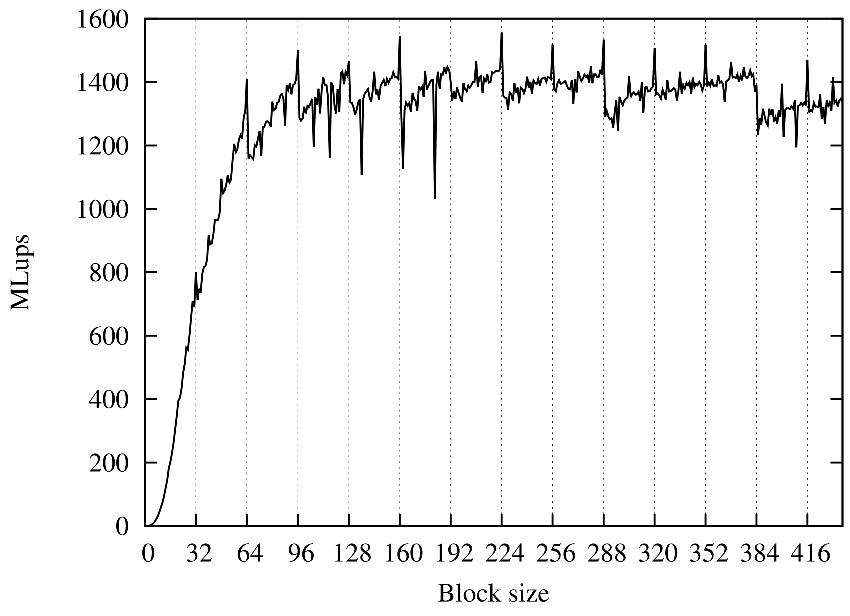


Figure 29: Performance of the D3Q19 LBM benchmark with varying block size.

With the thread layout described in section 4.4.2, CUDA threads are organised in one dimensional blocks aligned with the x-axis and the length of a block is the same as the number of nodes along that axis. The program is run for a wide range of domain sizes from 1^3 to 438^3 which is the largest resolution possible on a single Tesla K40 (see comment in subsection 5.1.4), and the block size increases with the domain size, the results can be observed on the figure 29. The benchmark program averages at about 1400 MLups, but sizes that are a multiple of 32 (the dotted lines on the figure) always achieve a significantly higher performances. This is because threads are executed in group of 32 (a warp) and when the block size is not a multiple of 32 some threads from the last warp are wasted. For unexplored reasons, some random sizes, like 182 give significantly smaller performances.

Lattice sizes smaller than 64^3 would benefit from a two dimensional thread layout, i.e. grouping a few one-dimensional blocks into one, so that the number of threads per block would be increased and so would the GPU occupancy.

Ensure single precision

As discussed previously, the use of single precision floating point operations makes the computation twice as fast, while keeping a sufficient level of accuracy (see chapter 6). In the code, floating point numbers are defined as real, where real can be defined either as float or double, by using the following command,

Listing 11: Define the type *real* as single precision floating point

```
typedef float real;
```

This allows to easily change the accuracy of the program.

However it easy to miss out constants left in the kernel code. For instance in the following code sample, `1.0`, `18.0`, `4.5` and `1.5` will be cast as double by the compiler, introducing involuntarily double precision calculations. One solution is to use `1.f`, `18.f`, `4.5f` and `1.5f` to specify them as single precision floating point numbers.

Listing 12: Examples of hidden double precision calculations

```
//double precision calculations
f1eq  = rho/18.0 * (1 + 3*vx + 4.5*vx*vx -1.5*v2 );
//single precision calculations
f1eq  = rho/18.f * (1 + 3*vx + 4.5f*vx*vx -1.5f*v2 );
//cast to the chosen precision
f1eq  = rho/real(18) * (1 + 3*vx + real(4.5)*vx*vx -real
    (1.5)*v2 );
```

A good way to check if there are some hidden double precision numbers in a kernel is to compile it for the CUDA capability 1.0, using the flag `-arch=sm_10`. This old version did not support double precision and the compiler will print a warning message during the compilation.

```
g++ -arch=sm_10 program_name.cu
ptxas warning : Double is not supported. Demoting to float
```

Avoid non-standard data types

This is more of a general advise when programming for GPU. As of today, even the latest version of CUDA, the use of advanced C++ functionalities (like classes, lambda functions, function template specialisation, etc.) often result in performance lost. Thus, in order to achieve the highest performance possible, it is advisable to limit the program to standard C data type like `float`, `double` and `int` and to avoid function calls within a kernel. Of course, this is not always practical and the use of classes and functions is an inherent part of C++ programming that massively simplifies programmer's job. So programmers should not completely avoid these functionality if they need them, but instead employ them with parsimony, knowing that each time they are utilised they risk reducing the performance of the program, sometimes only slightly and sometimes drastically.

For example, the lattice size, that is passed to the kernel as a parameter could be passed as `unsigned int` as it is always strictly positive. But doing so results in a performance loss of 5.5 % because the GPU is less efficient with an `unsigned int` than with a simple `int`. It could also be suggested to abandon integer instructions altogether, replacing them by floating operations with a cast to integer at the end.

Help the compiler optimise the code

It can be sometimes useful to rearrange mathematical expressions to help the compiler optimise the code. And this can have a significant impact on the performance. For instance for the computation of the macroscopic velocity $\vec{v}$, the sum of the distribution functions needs to be divided by the density ρ, this can be implemented in two ways,

Listing 13: The order of the terms within an expression matters.

```
//Expression 1
vx = (sum along x...)/rho;
vy = (sum along y...)/rho;
vz = (sum along z...)/rho;
//Expression 2
vx = (1/rho)*(sum along x...);
vy = (1/rho)*(sum along y...);
vz = (1/rho)*(sum along z...);
```

In expression 2, the compiler is able to recognise that the same factor is multiplying each sum and optimise the code accordingly. In terms of performances, the kernel using expression 1 runs at 1324 MLups, i.e. 13.3 % slower than with expression 2. This kind of simple code restructuring can be worth the effort.

Try different compilers and architectures

As mentionned previously, benchmarking a GPU program is difficult because the performance can depend on the compiler used and the video driver. To showcase some of these difficulties, the same benchmark code is compiled for different target GPU architectures (using the compilation flag `-arch=sm_XX`, where `XX` is either `10`,`20`,`30` or `35`), but tested on the same Tesla K40 GPU (native CUDA capability 3.5) with CUDA 6.0 and version 346.59 video driver. The measured performances are summarised on the table 5. More recent target architectures, like the Maxwell architectures (CUDA capability 5.0) cannot be tested on the Tesla K40.

CUDA Capability	Performance
1.0	1531 MLups
2.0	1515 MLups
3.0	1519 MLups
3.5	1518 MLups

Table 5: Performance of the D3Q19 BGK LBM benchmark compiled for different target GPU architectures.

As can be seen from the performances, the optimisations performed by the compiler are not the same depending on the target architecture, and the now deprecated capability 1.0 (it has been removed since CUDA 7.0) gives the highest performance. However, the differences in this test are infinitesimal, i.e. of the order of 1%. Also although CUDA 7.0 removed the capability 1.0, the default 3.5 capability gives a slightly better performance (1521 MLups).

5.2.3 *Error Correcting Code*

The error correcting code (ECC) is a piece of software implemented directly into the GPU that allows for error checking on each memory access. It is only available on the high-end GPU (i.e. the Tesla product line), but it can be disabled to improve the effective memory bandwidth. This feature is mostly unknown in the LBM community, only [258] reported on it.

It is possible, although very unlikely, that during an access to memory, some bits of the data become falsified, resulting in an erroneous value being read, which can significantly affect the output of the program and give false results. To remedy this problem, CPU implements an inbuilt tool that adds redundant data to every memory access, such that erroneous bits can be detected and corrected. This is not such a problem on gaming GPU, as a memory error will likely result into some pixel behind of the wrong colour for one frame of an animation, i.e. it is a minor, barely noticeable artefact. However, the situation is very different for scientific computing, as these erroneous bits can compromise the accuracy and the stability of simulations. As a result, NVIDIA has added an error correcting code functionality to its Tesla GPUs. When this functionality is enabled, the GPU reserves 12% of its total memory for the checking and performs some additional operations for each memory access. This obviously reduces the effective memory bandwidth, and it also has side effects like making the context synchronisation more expensive and reducing the efficiency of uncoalesced memory transactions [259]. The ECC implementation from NVIDIA can correct single-bit error and will throw an `cudaErrorECCUncorrectable` message for larger errors.

To show the performance boost provided by disabling the ECC, the D3Q19 BGK LBM program is run twice. Once with the ECC enabled, and once with the ECC disabled after rebooting the computer. The following command can be used to enable/disable the ECC, `nvidia-smi -e MODE` (need root permissions), where `MODE` is replaced by `0` to disable ECC and `1` to enable it. When the ECC mode is enabled, the command `nvidia-smi -q -d ECC` allows to check for the number of ECC errors detected since the last boot for all the types of memories.

The measured performances are summarised in the table 6. It can be seen that the effective memory bandwidth is increased by 13.2% when disabling ECC, while the program speed is increased by 19.2%. Not only the program is faster because of the additional available bandwidth, but the computations are also more efficient. In other words, the LBM program is achieving 91% of the effective bandwidth when ECC is enabled while it reaches an impressive 96% when disabled. In that case, only 4% of the time is spent on the computations.

	Memory Bandwidth	Program Speed	Efficiency
ECC on	212 GB/s	1274 MLUPS	91%
ECC off	240 GB/s	1518 MLUPS	96%

Table 6: Performance of the D3Q19 BGK LBM benchmark program with and without ECC and relative efficiency based on the effective memory bandwidth. GPU clock rates are set to 3004,875MHz.

Disabling ECC might seem like a risky thing to do, but actually running a program with ECC turned off is not such an issue. During the four years of this Ph.D., and after using the GPU almost daily, not even a single memory error has been detected. And based on the discussion with an NVIDIA engineer at the GPU Technology Conference, it is less likely to run into a memory error than to have a defective GPU (in which case, the ECC should detect that).

In conclusion, it is advised to turn the ECC option off, because of the performance improvement it provides, and to only turn in on if the GPU starts behaving strangely.

5.2.4 *GPU boost*

NVIDIA GPU Boost is a feature available on both gaming cards (i.e. GeForce) and professional cards (i.e Tesla) that dynamically improves application performance by increasing the cores and memory clock rates when sufficient power and thermal headroom are available. The Tesla K40 GPU is designed for a specific *Thermal Design Power* (TDP) of 235W, this TDP rating is an upper limit and in practice the GPU never reaches this power limit (or it would crash). For example, during the D3Q19 BGK LBM benchmark test, the GPU reaches a maximum of 185W with the highest frequency setting (see after).

Alternatively, the clock rates can be chosen manually amongst a list of supported clocks, either by running the command line tool `nvidia-smi` or programmatically using the NVIDIA Management Library[4]. The list of supported clocks can be accessed with the command `nvidia-smi -q -i ID -d SUPPORTED_CLOCKS`, where `ID` is the identification number of the GPU. On the Tesla K40, it outputs the following.

```
Supported Clocks
    Memory                      : 3004 MHz
        Graphics                : 875 MHz
        Graphics                : 810 MHz
        Graphics                : 745 MHz
        Graphics                : 666 MHz
    Memory                      : 324 MHz
        Graphics                : 324 MHz
```

There a are two main modes, the idle mode, characterised by having both memory and core clocked at 324MHz, and the normal mode for running computations where the memory is at 3GHz and various clock speeds are available. The default clock speed is 745 MHz, and in theory the GPU boost technology should increase it to 875MHz when the GPU is under stress. The user can alternatively force the GPU to run at its highest clock rates using the following command, `nvidia-smi -ac 3004,875`. For this command to take effect, the user needs root access and he needs to have previously enabled the perseverance mode on the GPU via the command `nvidia-smi -pm 1`.

This increase in clock rates results in faster computations but also better memory bandwidth, and improves the performance of the LBM benchmark by another 14%. It is probable that even better performance could be achieved by overclocking the GPU, i.e. running at higher frequencies than it was designed for, but NVIDIA does not allow overclocking on its Tesla GPU. This is not the case with the GeForce GPUs and that is why gaming card can sometimes provide better performances than the professional card (see figure 23) .

5.3 REAL-TIME CAPABILITY

The previous sections studied the efficiency of the LBM algorithm on the GPU, and it was shown that the performances achieved are close to the maximum theoretically possible on the given hardware. This section takes on the problem of real-time capability and whether these performances are sufficient to enable the real-time simulation of complex flow problems.

4 https://developer.nvidia.com/nvidia-management-library-nvml

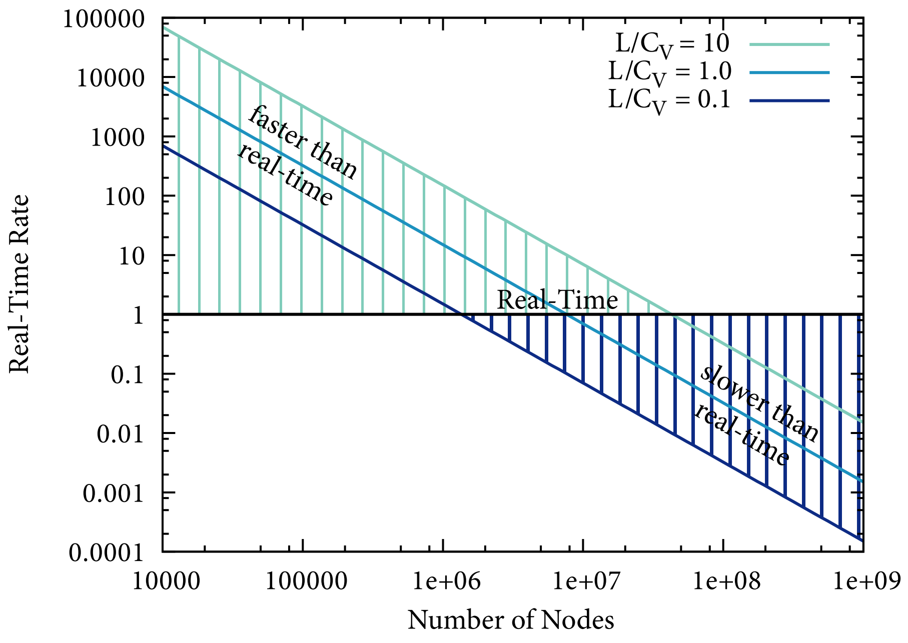


Figure 30: Estimated real-time capability for several ratios of L/C_V.

By definition, a flow can be simulated in real-time (or faster than real-time) if the physical time corresponding to a simulation step is equal (or greater than) the time taken by the computer to simulate that time-step. In LBM, all the quantities are dimensionless and the time-step needs to be recovered by a process of conversion of units (see Appendix B), but in short, with a convective scaling, the physical time-step δt can be related to the physical grid spacing δx and the ratio of physical and lattice velocity C_V as

$$\delta t = \frac{\delta x}{C_V} = \frac{\delta x}{V_{phys}} V_{lb} \tag{118}$$

The lattice velocity V_{lb} should be kept smaller than 0.1 to minimise compressibility errors and insure stability, while the physical velocity depends on the problem being simulated, but in the case of indoor air flow ventilation, it is typically of the order of 1m/s to 10m/s. Thus the velocity conversion factor is around 10 up to 100, i.e.

$$10 \lesssim C_V \lesssim 100. \tag{119}$$

The grid spacing δx can be expressed from the number of nodes $N = N_x \times N_y \times N_z$ and the physical length scale L, which can be between a few meters for small offices up to a hundred of meters for the largest data centre,

$$\delta x = \frac{L}{\sqrt[3]{N}} \tag{120}$$

The time in seconds required to simulate a flow problem comprised on N nodes with the optimised LBM running on GPU can be estimated from the number of nodes updated per second (i.e. 1500 MLups),

$$\Delta t = \frac{N}{\text{MLups}} \tag{121}$$

The previous equations can be combined to estimate the *real-time rate*, that is the ratio of computational time to physical time,

$$\frac{\delta t}{\Delta t} = \frac{\text{MLups}}{N^{4/3}} \frac{L}{C_V} \tag{122}$$

Using the equation, the real-time rate can be computed for a given problem and resolution, if higher or equal to 1, then the problem can be simulated in real-time. Figure 30 shows the variation of the real-time rate with the resolution for different ratios of L/C_V. For a typical indoor air-flow simulation, L/C_V is of the order of 1, in which case the critical value for real-time performance is 7.6 million nodes. That should be sufficient for accurately simulating rooms with simple geometry.

5.4 SUMMARY

Measuring the computational performance of a LBM program is a delicate matter. Not only it depends on the optimisation level of the source code, but also on the computing architecture, the compiler options, the problem size and geometry, the physics at play... etc.

On top of the obvious "amount of computations per second", one might prefer to consider over performance criteria. For instance, for the development of a *green* system, the performance per watt is often cited.

Moreover, the comparison of CPU versus GPU performances is always an risky task: should the GPU results be compared against a CPU using a single thread, or all the threads, or the vector units? And then, should the development time be taken into consideration? Is a ten percent increase in performance worth a year of development?

On the other hand, the CUDA language offer a simple yet efficient programming environment, and using the straightforward optimisation techniques explained in the previous chapter, can achieve really high performance for the LBM on GPU. Higher than a finely tuned CPU implementation, and the simple tweaks introduced in this chapter can help to maximise the usage of the available resources. More importantly, the achieved performance is sufficient for real-time simulations in the context of most indoor air flows.

The validation of the program for benchmark CFD problems and for several indoor air flows will be provided in the next two chapters.

6 VALIDATION

The purpose of this section is to validate the LBM solver implemented on GPU for the simulation of both isothermal incompressible problems and thermal problems under the Boussinesq approximation, as described in section 2.3.5. The first section takes an interest in the flow through a channel, also known as Poiseuille flow, which has a simple analytical solution. The second section focuses on the flow in a lid-driven cavity, both as a two-dimensional and three-dimension problem. It is a standard benchmark for incompressible Navier-Stokes solvers, and the LBM will be validated against results from the literature.

The last three sections study investigate thermal flows using three test cases of increasing complexity. The first one is a simple purely diffusive flow problem which has a simple analytical solution. The second test case is a purely advected flow where the Boussinesq force has been turned off; it has an analytical solution as well. The third case is that of the natural convection of a flow in a differentially heated cavity, also know as the *double blazing problem*. This last test case does not possess an analytical solution, thus the LBM will be compared against the results of a commercial CFD software, Fluent.

6.1 2D POISEUILLE FLOW

The Poiseuille flow, named after the French physicist Jean Poiseuille, consists in the flow between two parallel plates, separated by a distance L, subjected to a constant body force ($\vec{F}$). It is a representation of the stationary flow within a pipe subjected to a constant pressure difference at each end. The established flow is time independent and has a simple quadratic profile solely dependant on the distance to the walls (y). The component of the velocity which is normal to the wall in null ($u_y = 0$), so the velocity can be expressed as $\vec{u} = u_x(y)\vec{e_x}$. The Navier-Stokes equation can thus be reduced to a simple one dimensional second order PDE that can be solved analytically.

Jean Poiseuille (1797-1869) French physicist and physiologist, he studied the flow of liquids through small pipes, such as blood flows in capillaries and veins.

$$\nu \frac{\partial^2 u}{\partial y^2} = \frac{\partial p}{\partial x} = \text{cste} \qquad (123)$$

Using the no-slip boundary conditions on each plate, i.e. $u(0) = u(L) = 0$, it can be shown that the solution to equation (123) takes the form

$$u(y) = \frac{4u_{max}}{L^2} y\,(y - L) \qquad (124)$$

where u_{max} is the maximum flow velocity, located at the centre of the pipe, and it can be expressed from the pressure gradient,

$$u_{max} = \frac{L^2}{8\nu}\frac{\partial p}{\partial x}. \tag{125}$$

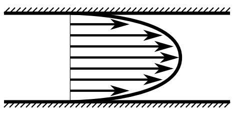


Illustration of the velocity field in a Poiseuille flow.

In the simulation, it is more convenient to drive the flow using a constant body force $\vec{F}$ and periodic boundary conditions along the streamwise direction, which avoid the need to implement pressure boundary conditions. In this case, the force can be chosen so that the velocity at the centre line is the desired u_{max} (once the flow is fully developed),

$$\vec{F} = \frac{8\nu}{L^2} u_{max}\vec{e_x}. \tag{126}$$

Due to its simplicity and the availability of an analytical solution, the Poiseuille flow is often used to study the accuracy of boundary conditions in LBM [225, 215]. Here, the goal is not to reproduce the results of these studies (only the bounce-back method will be shown for illustration), but rather as a mean to introduce the concepts of *grid convergence* that will be applied in the next section (see 6.2).

The process of grid convergence, also known as a grid refinement study, is a technique to evaluate discretisation errors in a CFD simulation. It involves performing multiple simulations of the same system using successively finer grids (i.e. higher resolutions) until the relevant simulated physical quantities converge towards their asymptotic values, which should be free from discretisation errors. The *order of convergence* of the method is determined by how fast these quantities converge.

As the resolution of the simulation (N) is increased, the grid spacing (Δx) becomes finer and it important that other physical quantities are scaled accordingly, so that the Reynolds number remains constant,

$$Re = \frac{U_{lb} N}{\nu_{lb}}, \tag{127}$$

where the subscript lb denotes lattice units.

Clearly, to keep the Reynolds number constant while N varies, one must either vary the velocity U_{lb} (this is known as *diffusive scaling*) or the viscosity $\nu_{lb} = (\tau - 1/2)/3$ (*convective scaling*). The convective scaling process is similar to that of the Chapman-Enskog expansion by which the compressible Navier-Stokes system is recovered. In contrast, the diffusive scaling views compressibility effects as numerical effects and it is the natural choice if the LBM is viewed purely as a numerical method to solve the incompressible Navier-Stokes equation [260].

In a diffusive scaling, the parameters are scaled as follows:

$$\begin{aligned} N &\to 2N, \\ \nu_{lb} &\to \nu_{lb}, \\ U_{lb} &\to U_{lb}/2. \end{aligned} \tag{128}$$

In this scaling, the error due to compressibility effects decrease as the same rate as the error due to grid discretisation. On the other hand, as $\Delta t \sim \Delta x^2$, the required number of time steps can become prohibitive for very large simulations. A more crucial issue is that in the continuum limit $N \to \infty$, the velocity tends to zero $U_{lb} \to 0$ and the round-off errors due to limited machine precision can become predominant.

On the other hand, in a convective scaling, the parameters are scaled as follows:

$$\begin{aligned} N &\to 2N, \\ \nu_{lb} &\to 2\nu_{lb}, \\ U_{lb} &\to U_{lb}. \end{aligned} \tag{129}$$

Using this scaling, the compressibility errors that are of the order $\mathcal{O}(U_{lb}^2)$ do not disappear even in the continuum limit $N \to \infty$, but they can be kept small by choosing a small value for U_{lb}. However, it assumes the time to scale linearly with the number of grid points, i.e. $\Delta t \sim \Delta x$, rather than quadratically as in the diffusive scaling. So the required number of time steps is much less demanding.

To illustrate the process of grid convergence, this section evaluates the convergence under a diffuse scaling of the so-called half-way and full-way bounce-back boundary conditions for the simulation of the Poiseuille flow in LBM using the simple BGK collision operator. As the flow is independent of the direction x, the number of nodes along x can be chosen very small. The grid convergence is established by considering the following resolutions : 3×21, 3×31, 3×51, 3×71, 3×101, 3×201, 3×301 and 3×501.

In order to establish when the flow has fully developed, the maximum flow velocity u^c_{max} is

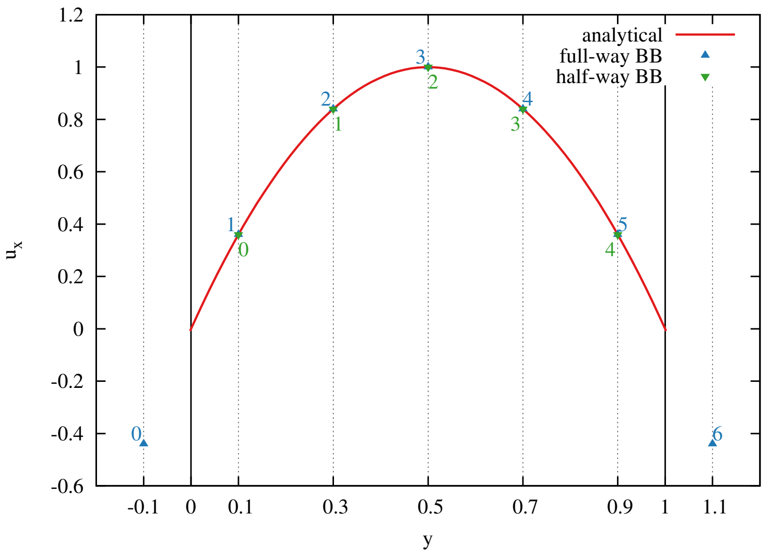


Figure 31: Illustration of the node locations and corresponding flow velocities for half-way bounce-back using N=5 and full-way bounce-back using N=7. The walls are located at $y = 0$, and $y = 1$, they reside half a distance away from the simulation nodes.

monitored every 10000 time-steps, and the program runs until the fluctuations are smaller than 10^{-10}. This computed maximum velocity u^c_{max} is then used to evaluate the analytical quadratic profile with the equation (124) rather than the target analytical velocity u^a_{max} that was used to estimate the force, as the two can differ slightly. To make sure that the pipe centre aligns with a node location and avoid interpolations, N is always set as an odd number, and u^c_{max} is measured at the node $i = (N-1)/2$, see figure 31.

As explained in section 3.2.3, the location of the solid wall is always half-way between the nodes, both in the 'half-way' and in the misnamed 'full-way' boundary conditions and only the actual location of the walls varies. In the half-way method, boundary nodes are fluid nodes and the walls are located outside of the simulation domain. In the full-way bounce-back, boundary nodes are solid nodes which reside outside of the fluid domain; thus they act as *ghost-nodes* for the simulation, the velocity is not defined on these nodes and need to be discarded. It is important when exporting the data and computing the analytical velocity profile to use the right node location, as illustrated in figure 31,

$$y = \begin{cases} \frac{i+\frac{1}{2}}{N} & \text{half-way} \\ \frac{i-\frac{1}{2}}{N-2} & \text{full-way} \end{cases} \tag{130}$$

The viscosity is kept constant and the relaxation time is set to $\tau = 0.55$. The flow is driven by a constant body force $\vec{F} = (F_x, F_y)$, where F_x is set according to equation 126 and $F_y = 0$. For a constant body force like this, it is sufficient to use a simple scheme, like adding the following term during the collision process :

$$F_i = \frac{w_i}{c_s^2} \vec{e}_i \cdot \vec{F}. \tag{131}$$

The velocity at the centre line (u_{max}) is computed so that the Reynolds number of the flow is always 100 (i.e. diffusive scaling), $u_{max} = Re \cdot \nu_{lb}/N$. The global relative error, that quantifies the difference between the LBM velocity computed at the

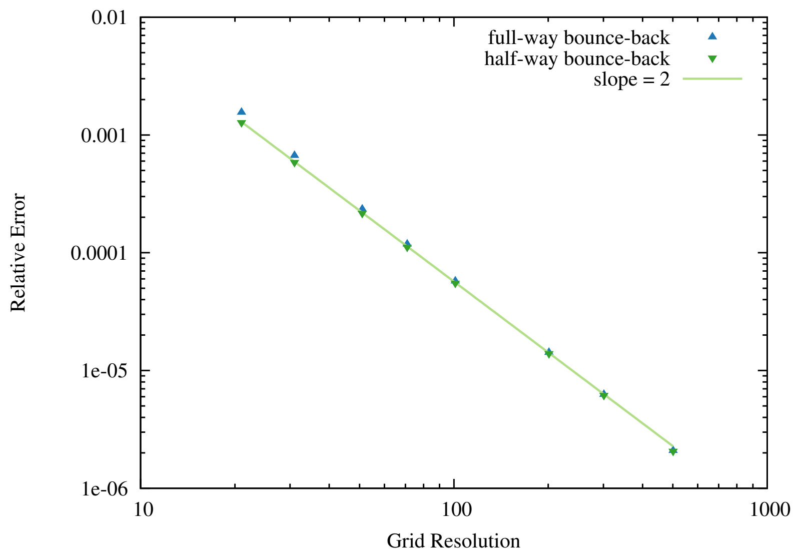


Figure 32: Grid convergence of the full-way and half-way boundary condition in the BGK LBM for a 2D Poiseuille flow with constant body force and diffusive scaling.

i^{th} node (u_i^c) and the analytical solutions evaluated at the location of the i^{th} node (u_i^a), is defined as

$$\text{Relative Error} = \sqrt{\frac{\sum_i \left(u_i^c - u_i^a\right)^2}{\sum_i \left(u_i^a\right)^2}}, \tag{132}$$

note that u is a generic (u_x or u_y) velocity component so the relative error can be evaluated for each component. However in a Poiseuille flow, the y component is always null, thus only the x component is considered. As seen on figure 32, the relative errors have a slope almost exactly equal to 2, both for the half-way and full-way boundary conditions, although the later is slightly less accurate on coarse mesh. This proves that the LBM is indeed second-order accurate for this problem.

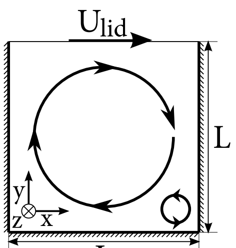
6.2 LID-DRIVEN CAVITY

The lid-driven cavity flow takes place in a square box and is driven by imposing the constant translation of the top lid, while a no-slip condition is imposed on the other walls (stationary walls). It is a popular benchmark test for CFD methods. Its popularity comes from its ability to generate complex vortex structure while retaining a simple geometry that allows for an easy implementation of the boundaries. The dynamic of the vortex structures in the lid-driven cavity highly depends on the Reynolds number and has been the subject of many investigations using a variety of methods, both experimentally [261, 262] and numerically [263, 264, 265].

This section focuses on the simulation using the lattice Boltzmann method, for a large range of Reynolds number, both in two and three dimensions. It should be emphasised that a real cavity would develop three-dimensional features due the no-slip conditions on the end walls. As a result, two-dimensional simulations provide only a first approximation, but they will be used to test the numerical stability and precision of multiple collision models and boundary conditions, because they allow faster computations for a larger range of resolutions than three-dimensional simulations. The LBM results are compared amongst themselves as well as against results from benchmark simulations available in the literature.

6.2.1 *Problem description*

The flow considered is that of an incompressible fluid in a square (2D) or cubic (3D) cavity of side length L. The flow is driven by the wall at $y = L$, moving tangentially in the x-direction at a constant velocity U_{lid}, the other walls are stationary. The movement of the top lid generates a large primary central vortex and several secondary vortices rotating in the opposite direction, arising near the corners (see sketch in the margin). The small vortices at the bottom corners can be difficult to capture numerically as they are weak and small. The top corners are both singular points (see section 6.2.2), and can be the cause of catastrophic instabilities, particularly at high Reynolds number. It is assumed for this section that the flow has steady state, but it in practice, as the Reynolds number increases passed a certain critical value, the flow becomes unsteady, it first shows an oscillating behaviour and eventually becomes turbulent. As LBM is intrinsically a time-dependent method, the flow is set to a uniform zero velocity initially, then it is allowed to evolve in time until it does not change anymore, this final state is *the steady-state solution*. Simulations are assumed to have reached the steady-state once the L^2 norm of the velocity compared to an earlier snapshot in time is smaller than 10^{-12},



Geometry of the lid-driven cavity problem.

$$L^2 = \sqrt{\frac{\sum_i \left(\vec{u}\left(\vec{x}_i, t - 1000\delta t\right) - \vec{u}\left(\vec{x}_i, t\right)\right)^2}{\sum_i \left(\vec{u}\left(\vec{x}_i, t\right)\right)^2}} \leqslant 10^{-12}. \qquad (133)$$

The convergence criteria for reaching the steady-state in LBM is defined using the relative error of the velocity.

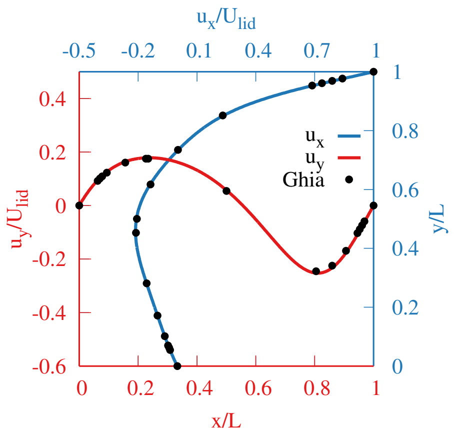
6.2.2 *Two-dimensional results*

Comparison with benchmark

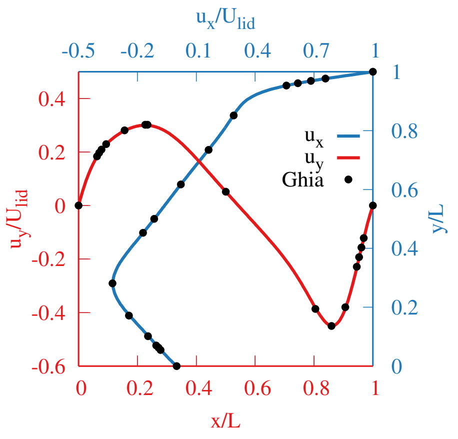
The two-dimensional (2D) lid-driven cavity can be accurately simulated by the LBM, even with the simple BGK collision operator, and for a large range of Reynolds number, as demonstrated by the various velocity profiles on figure 33. In order to match the LBM simulation to the benchmark data provided by [263], the Reynolds number is defined as $Re = 3U_{lid}(N-1)/(\tau - 1/2)$, where the number of nodes along the length of the cavity (N) is fixed to 256 (or 512 for high Reynolds number), the velocity of the lid (U_{lid}) is fixed to 0.1 and the relaxation time (τ) is computed to match Re. The simulation can become unstable as τ approaches 1/2, especially with the LBM BGK, so to avoid this problem, the resolution is increased to 512 if $Re \geqslant 1000$. The side and bottom walls are stationary and can be easily implemented with the half-way bounce-back method. The top wall is moving at the constant speed U_{lid}, there are multiple ways to implement such a boundary, the momentum exchange method described in [266] and based on [212] is preferred for its simplicity,

$$f_i = f_{\bar{i}} + 6\rho w_i e_{i,x} U_{lid}, \qquad (134)$$

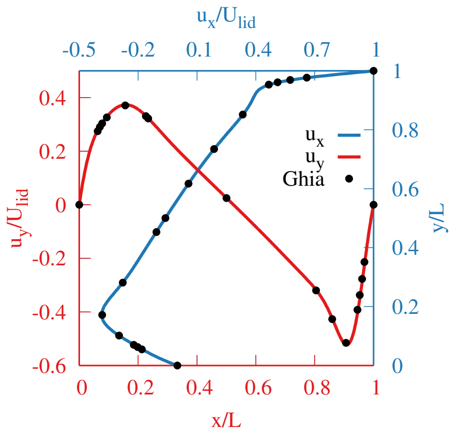
where i is the index of the unknown distribution function at the top wall, $\bar{i}$ is the index of the opposite distribution function (i.e. $\vec{e}_i = -\vec{e}_{\bar{i}}$), $e_{i,x}$ the x-component of the direction $\vec{e}_i$ and w_i the associated weighting factor.



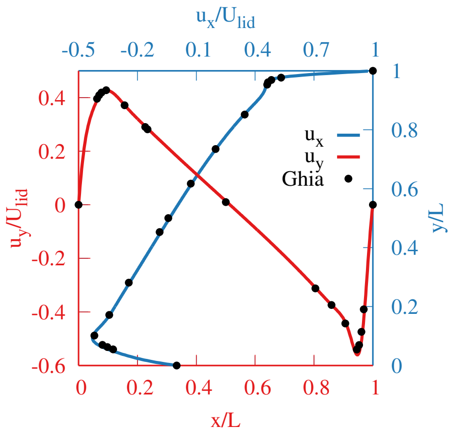
(a) Re = 100



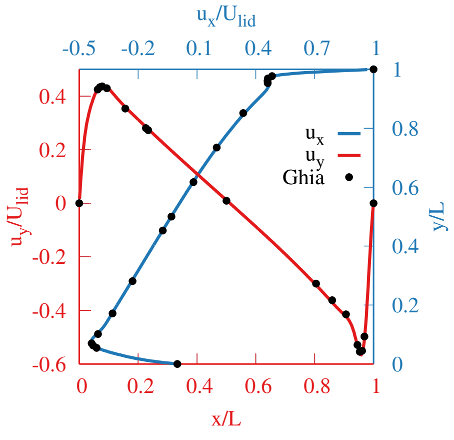
(b) Re = 400



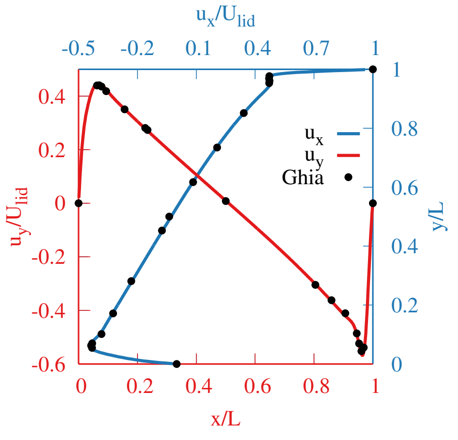
(c) Re = 1000



(d) Re = 3200

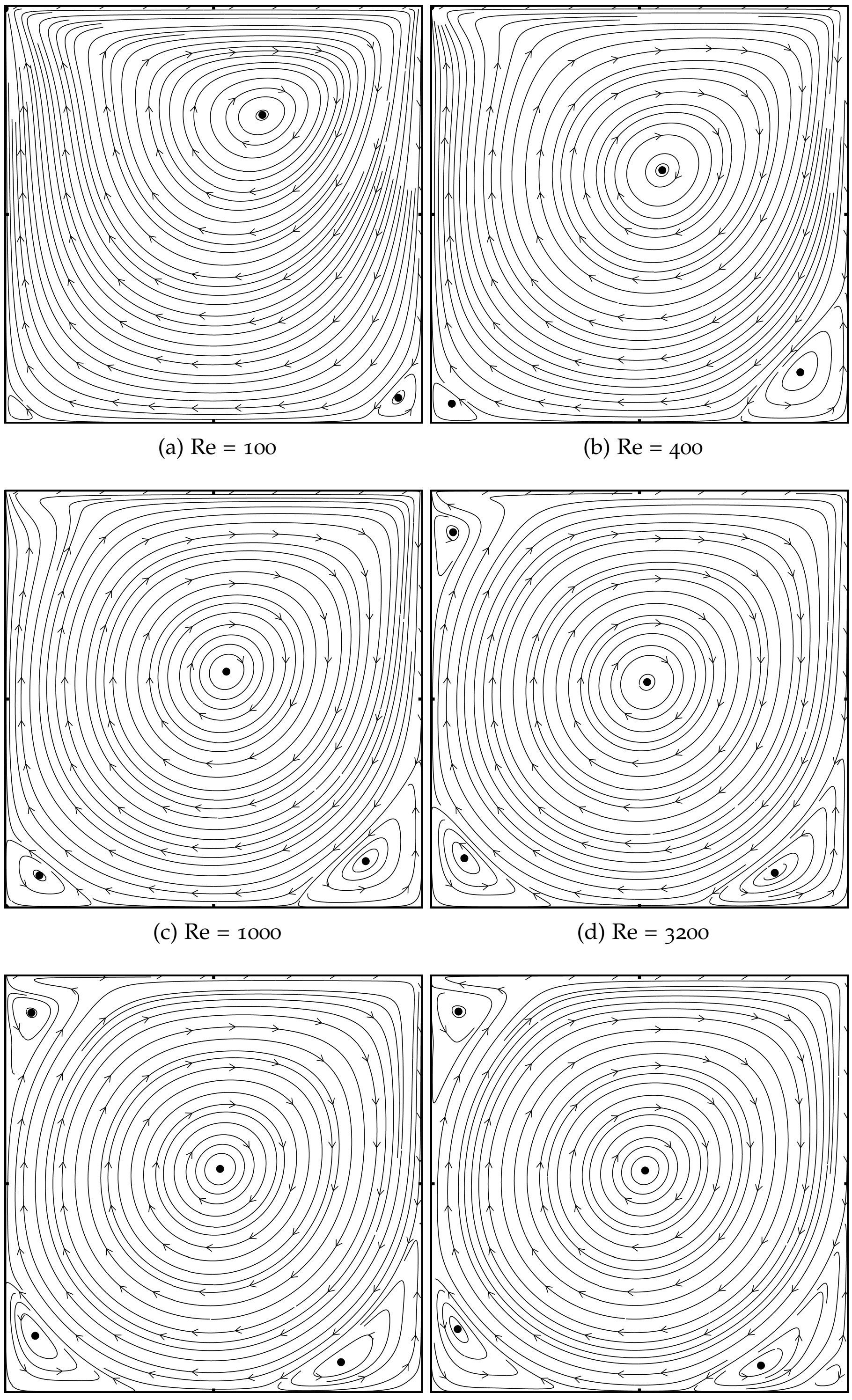


(e) Re = 5000



(f) Re = 7500

Figure 33: Velocity profiles obtained with the D2Q9 BGK LBM program of the x-velocity (u_x) across a vertical line and the y-velocity (u_y) across an horizontal line, crossing at the cavity geometric centre, for various Reynolds numbers. The dots represents the benchmark data from [263].


(a) Re = 100 (b) Re = 400

(c) Re = 1000 (d) Re = 3200

(e) Re = 5000 (f) Re = 7500

Figure 34: Comparison of the streamlines in a 2D lid-driven cavity at different Reynolds number. Vortex centres are located by a ●.

The figure 33 shows a very good agreement between the velocity profiles extracted from the converged LBM simulations and the benchmark results of [263]. However, it was found that a few of the points provided in this paper were erroneous (probably due to typing mistakes from the authors) as they were significantly off compared to the other points. The table 7 lists these erroneous points and provides their original and corrected values.

component	Re	location	original value	corrected value
x	3200	$y = 0.4531$	0.86636	0.086636
x	10000	$y = 0.5$	0.03111	-0.03111
y	400	$x = 0.9063$	0.23827	0.38

Table 7: List of erroneous point and corrected values for the lid-driven cavity profiles provided by Ghia et al.[263].

The figure 34 display the streamlines in the lid-driven cavity for each Reynolds number of interest. They show a large main central vortex, rotating clockwise, that creates smaller vortices at the bottom corners that are rotating in the opposite direction. The LBM properly captures these second vortices, for the whole range of Reynolds number. At $Re = 7200$, it also captures a third level of vortices, rotating clockwise, which appear in the bottom right corner. The streamlines visually agree with the ones provided in the references [263, 266, 59].

Grid convergence

In this section, the lid-driven cavity is used as a test case to study the order of convergence of the LBM under spatial refinement. The LBM is often claimed to be second order accurate in space [260], but that is usually without considering boundary conditions, which can sometimes reduce the order of convergence. The difficulty in studying the grid convergence for the lid-driven cavity comes from the lack of analytical solution, unlike for the Poiseuille flow from section 6.1. In place of an analytical solution, a numerical simulation conducted on a large grid is used as a "refer-

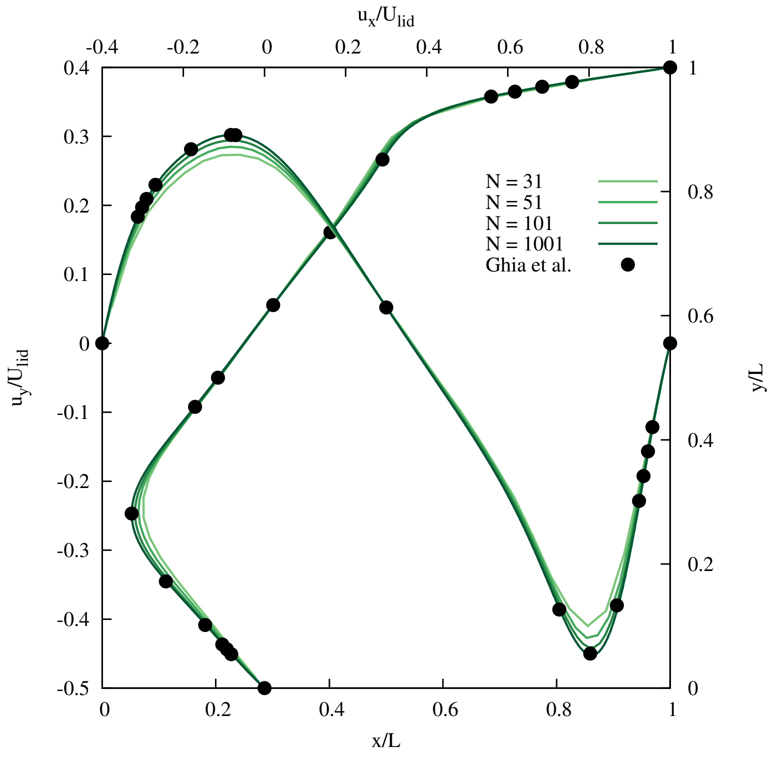


Figure 35: Velocity profile in the lid-driven cavity for $Re = 400$

ence" simulation. A diffusive scaling is employed to set the parameters for different resolutions consisting of 31×31, 51×51, 71×71, 101×101, 301×301, 501×501, 701×701 and 1001×1001, this last large resolution is taken as the reference simulation. These resolutions are significantly larger than what is typically done in the literature[266, 267, 268, 59] and is only made possible by the optimisation of LBM on GPU described previously (see section 4.4).

Figure 35 shows the improvements on the velocity profiles for $Re = 400$ as the resolution of the simulation is increased. From $N = 101$, increasing the resolution does not significantly improve the results and the profiles all merge into one. Thus, to avoid overcrowding the figure, the velocity profiles for intermediate resolution between $N = 101$ and $N = 1001$ are omitted.

It is clear that the larger resolutions results in more accurate simulation, as their velocity profiles align perfectly with the sample points of the reference, but it is important to estimate how fast these simulations tend towards the reference to estimate the order of convergence. To do so, multiple profiles at different resolutions need to be subtracted (see equation 132). The resolution is chosen as an odd number, so the geometric centre matches exactly with $x/L = y/L = 0.5$, i.e. $i = j = (N-1)/2$ in lattice units. But that is not sufficient for all the node locations to match exactly, even if the resolution is doubled, because of the half-way location of the walls. To compensate for that, the velocity of the reference simulation is extrapolated at the location of the coarse one using linear interpolations. This added interpolation does not affect the resulting convergence order.

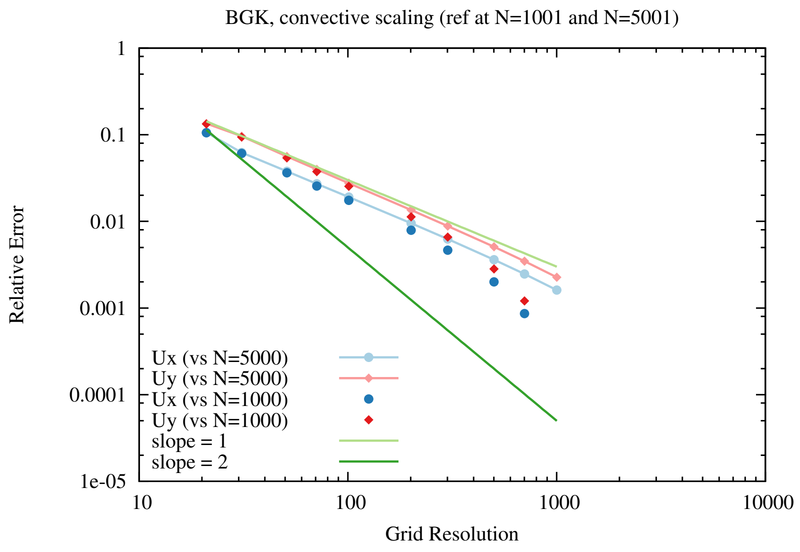


Figure 36: Grid convergence of the BGK LBM for the x-velocity (along the vertical centre line) and for the y-velocity (along the horizontal centre line) in a 2D lid-driven cavity flow for $Re = 400$.

Figure 36 presents the convergence results for the lid-driven cavity simulated using the BGK LBM. The Reynolds number is again $Re = 400$, and the change in resolution is done under a diffusive scaling with the relaxation time fixed to $\tau = 0.55$. The relative error is computed according to formula 132 where u_i^a is replaced by the velocity taken from the 1001^2 simulation. The relative error on the x-velocity is measured on the vertical centre line while that of the y-velocity is taken on the horizontal centre line. The simulation seems to be converging more like a first order scheme (top light-green line) rather than a second order scheme (bottom dark-green line) and the estimated value for the order of convergence is approximately 1.2. The convergence order appears to increase in the second half of the plot (i.e. for $N \geqslant 301$), but it is likely due to the closeness to the reference located at $N = 1001$. This is indeed confirmed when using an higher resolution (5001×5001) simulation as the reference, as shown in light blue and light red colour on the figure.

The convergence order observed here is much smaller compared to what is usually claimed in the literature[266, 267]. There are multiple factors that can affect the convergence and might explain this difference: boundary conditions for the top lid and the walls, relaxation time value, use of a convective scaling, modified collision operator (cascaded, MRT), measuring the relative error using the whole field rather than lines, etc... Many combinations of these factors were tested and ultimately none yield a significant change in the order of convergence. The convergence of the LBM for the lid driven cavity problem appears to be better than first order, but is still closer to first order than second order.

The second order convergence reported in the literature could originate from the use of relatively smaller lattice resolution. *Comparing the velocity profile against an under-resolved reference simulation tends to overestimate the convergence order of the method.* This statement is clearly illustrated in the figure 37. By increasing the resolution of the reference simulation from 128 to 1024, the convergence order decreases from 1.93, which would imply quasi second order, to 1.3, which is closer to first order. By comparing the coarse simulations against a small resolution reference, that does not fully recover the correct velocity profile, the relative error is underestimated and consequently the convergence order is overestimated. This important aspect in the measure of convergence orders and it has not been commented on before (to the best of the author's knowledge).

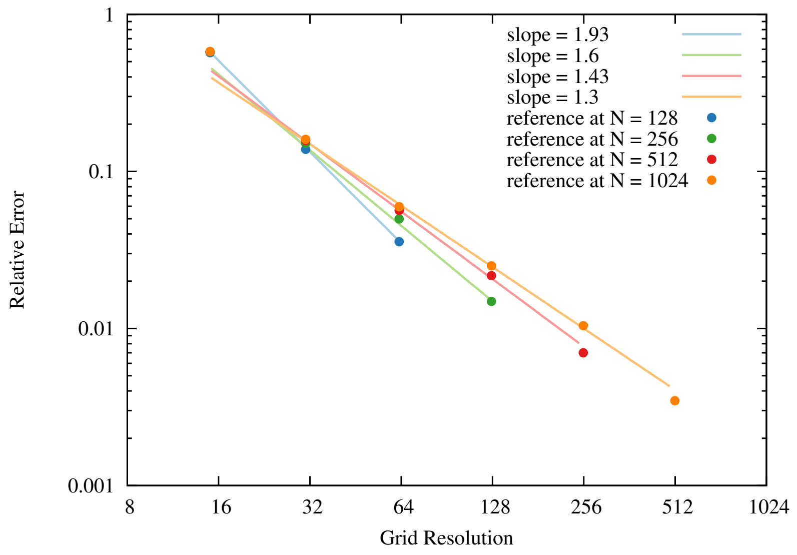


Figure 37: Grid convergence of the cascaded LBM in a 2D lid-driven cavity flow for $Re = 1000$. Under-resolved (i.e. small resolution) reference simulation gives an incorrectly high convergence order.

Convergence to steady state

For the range of Reynolds numbers considered, the flow in the 2D lid-driven cavity is a steady-state, i.e. the flow does not vary with time. However, the LBM is intrinsically time-dependent and the flow needs to be solved as a time dependant flow. The flow starts with a uniform zero velocity and evolves until the velocity does not vary significantly any more, according to the convergence criteria defined in the equation 133. It is interesting to study the evolution of this convergence criteria over time, as plotted on figure 38, to see how fast the LBM converges towards the steady-state. This is not to be mistaken with a convergence study under time refinement (i.e. using smaller time-steps).

The simulation parameters used to create the figure 38 are: $Re = 400$, $N = 256$, $\tau = 0.6$, $U_{lid} \approx 0.05$. The computations are performed both in single precision (in blue on the figure) and double precision (in green). The wall clock time (that is the time taken by the computer to complete the simulation) for a given time-step and for a given precision can be found by following the light blue or light green line. The double precision LBM reach a convergence criteria as small as 10^{-13} after approximately 6.10^5 times-steps, i.e. 40 seconds of computation. While the best convergence for single precision is about 10^{-4}, which is reached after 2.10^5 time-steps, i.e. 8 seconds. Although the convergence for single precision is not as good as in double precision (due to larger rounding off errors), a relative error of 10^{-4} is likely to be a sufficient accuracy for real-time applications, while the required computing time is four times smaller than that of double precision. Moreover, the overall quality of the velocity profiles does not appear to be significantly affected by the lack of floating point precision.

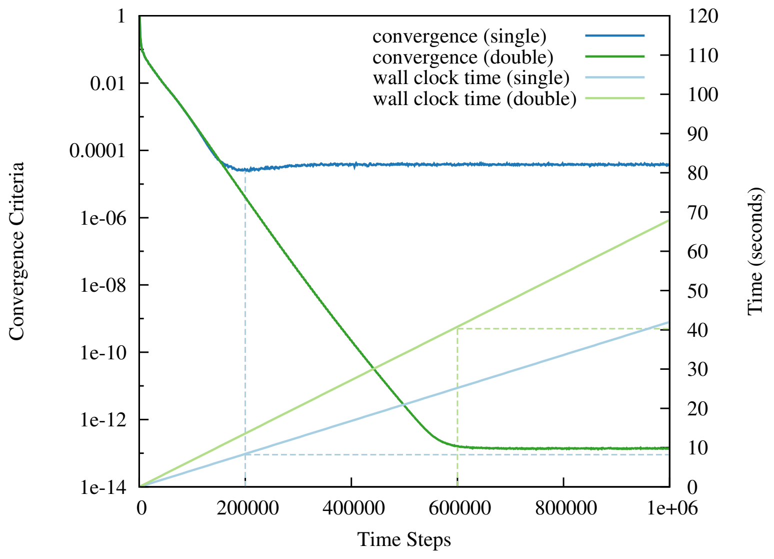


Figure 38: Convergence of the LBM towards the steady-state solution for $Re = 400$ in single or double precision computations.

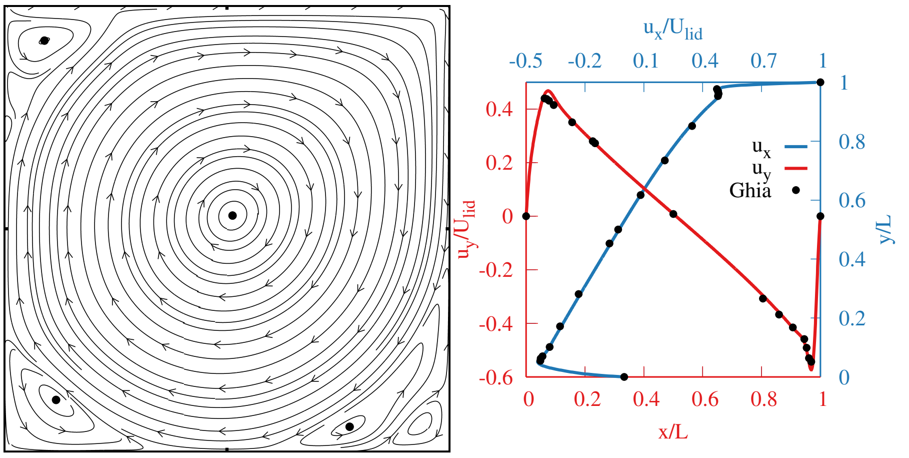
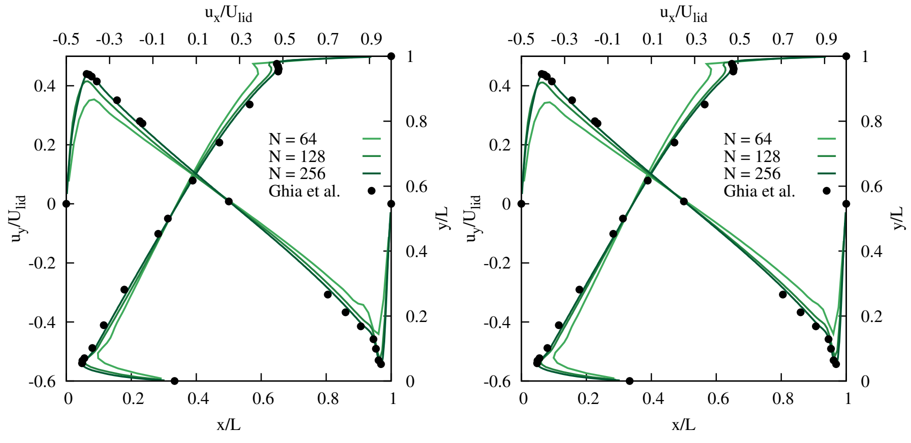


Figure 39: Streamlines and velocity profile in the 2D lid driven cavity for $Re = 10000$ using the BGK LBM.

In their original paper, Ghia et al. [263] provide data of the lid driven cavity simulated for a Reynolds number $Re = 10000$. Simulating such a high Reynolds number using the LBM is challenging and normally requires a high resolution mesh, because small relaxation times result in unstable simulations. Moreover, the algorithm struggles to converge towards a steady state, as the resulting flow displays time varying features. This behaviour is expected, $Re = 10000$ has been show to be beyond the first Hopf bifurcation, i.e. the Reynolds number at which the flow changes from steady to unsteady characteristics, for the 2D lid driven cavity flow, which is estimated to be around $Re = 8000$ [269]. Nevertheless, the BGK LBM manages to solve the flow with a reasonable accuracy, as displayed on figure 39, provided that a large resolution (1023×1023) is used. The streamline plot shows the formation of a large third vortex in the bottom right corner in agreement with the benchmark data.

For small resolutions (smaller than 256×256), the BGK LBM is unstable and gives unreliable results. This section studies the effect of changing the collision operator (MRT and cascaded) and using a turbulence model (Smagorinsky LES) in order to accurately simulate the lid-driven cavity for large Reynolds number with a small resolution.

The MRT and cascaded LBM both provide almost identical results. Figure 40, displays the velocity profile for three different grid resolutions 64×64, 128×128, 256×256 for each model. These two models, that do not rely on a sub-grid turbulence model, perform surprisingly well for these small resolutions where the BGK LBM would not be stable. The smallest resolution $(N = 64)$ is too coarse for accurately capturing the boundary layers, which is very thin, yet the resulting velocity profile is not very far off the reference. However, for even smaller resolutions, both methods become unstable (although it is



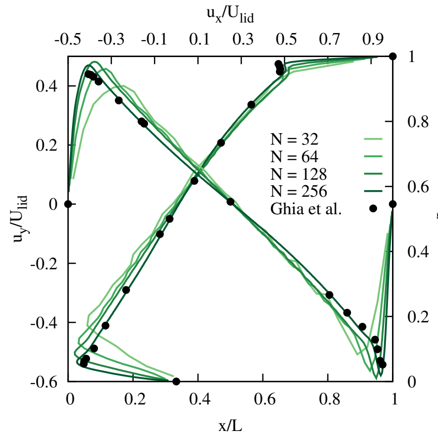
(a) MRT LBM

(b) Cascaded LBM

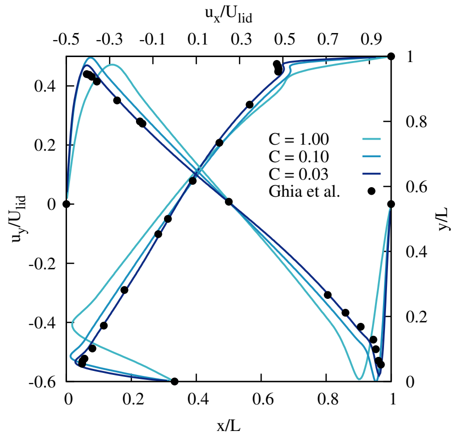
Figure 40: Velocity profile in the 2D lid driven cavity for $Re = 10000$ using the MRT (left) and cascaded LBM (right).

possible that the cascaded LBM could be made unconditionally stable with different boundary conditions). The two higher resolutions ($N = 128$ and $N = 256$) both resolve the boundary layers correctly and the resulting profiles are very close to the reference. At the resolution $N = 128$, δx is about ten times the Kolmogorov length scale (which varies as $Re^{3/4}$) and this resolution is apparently sufficient to solve the flow accurately without the need for a turbulence model.

Nevertheless, the LES turbulence model described in section 2.5.1 can be used to stabilise BGK simulations on these small resolution lattices, as shown on figure 41. The introduction of a turbulence model allows the resolution on small grids, even smaller than what is possible with MRT or cascaded, but it comes at the price of additional diffusion. Indeed, the sub-grid turbulence model introduces an additional *eddy viscosity* whose significance is controlled by a constant C, known as Smagorinsky's constant (for further details, see section 2.5.1). For the resolution $N = 256$, it is found that the value of C required to accurately solve the flow is around $C = 0.03$, which is smaller than the value of $C = 0.1$ usually advised in the literature[189, 24]. As depicted on the figure 41b, the accuracy of the result is improved as the Smagorinsky constant C is decreased from 1.0, to 0.03. The value $C = 1.0$ is clearly too large, it overestimates the effect of sub-grid eddies and the effective viscosity is too large, as a result the velocity profile represents better $Re = 1000$ than the reference. Using $C = 0.1$ gives a reasonable agreement with the reference, although it shows a larger thickness of the boundary layers. Lowering the constant to $C = 0.03$ allows for a more accurate resolution of the flow. Actually, it is likely than reducing C even more would further improve the accuracy, but it appears that C cannot be lower than 0.03 (for this setting), otherwise numerical instabilities arise. This means the eddy viscosity is not sufficient to damp the high-frequency fluctuations at small scale, and it would be interesting to see if combining the turbulence modelling with an alternative collision operator (like MRT or cascaded LBM) allows for a better stability. In this problem, the Smagorinsky constant C should be chosen as small as possible, as long as the resulting flow is numerically stable. Doing so, C can be adjusted to allow for stable simulations on smaller grids even on grids as small as 32×32 where MRT and cascaded loose their stability. The effect of reducing the resolution can be observed on figure 41a. The smallest resolutions display spatial oscillations (see next section) and do not resolve the boundary layers accurately. The use of a wall turbulence model would be necessary to recover the velocity profile more accurately for such coarse simulations.



(a) Changing resolution.



(b) Changing Smagorinsky constant C with fixed resolution $N = 256$.

Figure 41: Velocity profile in the 2D lid driven cavity for $Re = 10000$ using the Smagorinsky BGK model.

Pressure field and corners issues

As mentioned previously, most of the issues related with the lid-driven cavity arise from the top corners. These two corners are singular points to the flow: the velocity at these points is both that of the lid (U_{lid}) and that of the walls (zero because of the no-slip property of the wall). This is purely a modelling issue, a physical experiment in the lab would likely see a gradient of velocity at each corner: zero at the wall and U_{lid} some distance away. For a simulation, in practice, a choice must be made for the corner nodes. Here, they are treated as if they are part of the solid walls. Nevertheless, there is a large gradient (the infinity actually!) between the velocity at the corner nodes and their neighbour nodes. Numerical methods do not like large gradients, they often result in spurious oscillations that can lead to severe numerical instabilities.

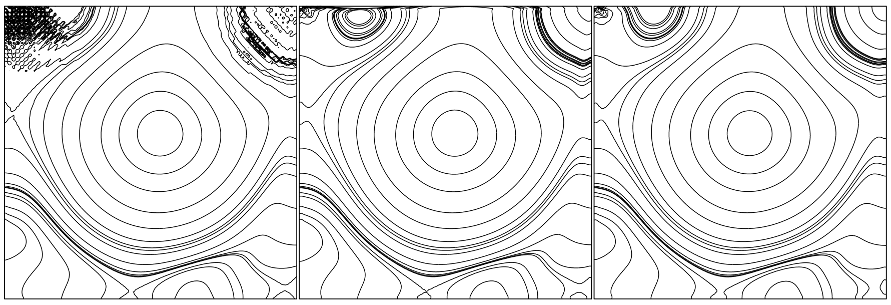


Figure 42: Contour of pressure in the 2D lid driven cavity for $Re = 1000$ using the resolution $N = 128$. From left to right: BGK, MRT and cascaded LBM scheme.

In the LBM, the velocity field is not affected as much by these instabilities as the pressure field. Thus, it is sensible to look at isocontours of pressure to illustrate the problem. The figure 42 provides a comparison of the isocontours of pressure for the BGK, MRT and cascaded LBM models for $Re = 1000$ using a 128×128 lattice and a lid velocity $U_{lid} = 0.1$. The values for the contours are hand picked to achieve a relatively uniform spacial coverage[1], they are centred around 1, which is the initial pressure. The BGK LBM shows strong oscillation near the upper corners, which are mostly fixed using either MRT or cascaded LBM. The results obtained by these last two methods are very close, but it can be noticed that the cascaded LBM gives smoother contours near the top lid.

The streamlines on figure 42 are constructed as contours of constant values (known as isocontours) of the stream function ψ. For a 2D flow, the stream function ψ is also the vector potential of the flow $\vec{\psi} = (0, 0, \psi)$ defined by

$$\vec{u} = \vec{\nabla} \times \vec{\psi}, \tag{135}$$

which can be projected along each axis to give

$$u_x = \frac{\partial \psi}{\partial y} \qquad \text{and} \qquad u_y = -\frac{\partial \psi}{\partial x}. \tag{136}$$

In LBM, the stream function can be calculated by numerical integration:

$$\psi\left(x_i, y_j\right) = \psi\left(x_{i-1}, y_j\right) + \delta_x u_y\left(x_{i-1}, y_j\right), \tag{137}$$

and ψ is set to zero on the domain borders ($\psi(x = 0, y) = 0$) to provide initial values for the integration. The above formula only provides a first order integration (which appears sufficiently accurate

1 the hand-picked pressure values are the following : 0.9980, 0.9982, 0.9985, 0.999, 0.9995, 1, 1.00025, 1.00035, 1.0005, 1.0008, 1.00094, 1.00098, 1.000993, 1.001, 1.00106, 1.0011, 1.0012, 1.00125, 1.00128, 1.002, 1.004, 1.008.

for the needs of this study) but the Simpson's rule [59] can be applied instead if second order integration is required:

$$\psi\left(x_{i+1}, y_j\right) = \psi\left(x_{i-1}, y_j\right) + \frac{\delta_x}{6}\left[u_y\left(x_{i-1}, y_j\right) + 4u_y\left(x_i, y_j\right) + u_y\left(x_{i+1}, y_j\right)\right]. \quad (138)$$

6.2.3 *Three-dimensional results*

Physically, the flow in a lid-driven cavity is unlikely to be two dimensional (because of the effect of end walls) nor steady (especially for high Reynolds number). The two dimensional steady state flows at high Reynolds numbers presented in the previous section should be considered as a "fictitious" flow. They were crucial, however, in the study of the numerical stability and convergence of the LBM. This section focuses on the three-dimensional lid-driven cavity. The geometry is the same as in the previous section, but the domain is now extended in the third dimension, enclosed by two additional no-slip walls. It is expected that most of the results established in two dimensions are amenable to three dimensions, that is why this section focuses solely on validating the 3D LBM with benchmark data and estimating the convergence order.

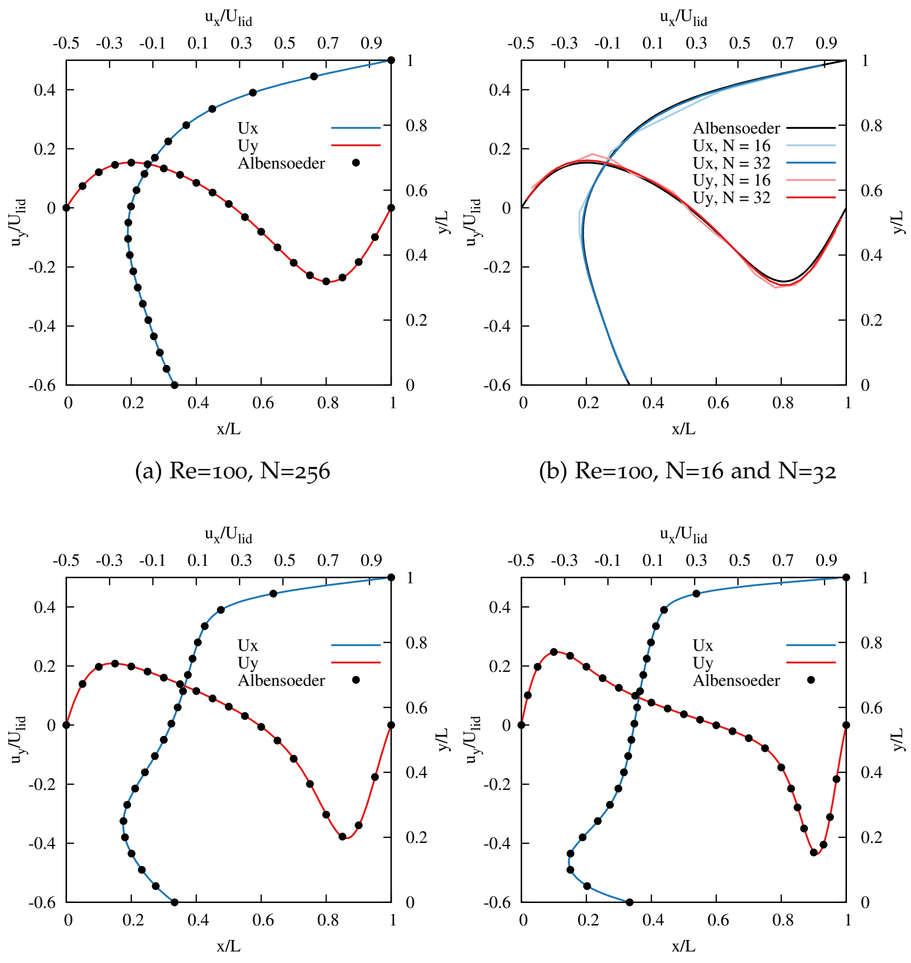
Comparison with benchmark

There are relatively less numerical study in the literature considering the three dimensional driven cavity than the two-dimensional one. While the 1987 paper from Ku et. al. [264] is often considered as a reference, this section uses the work of Albensoeder and Kuhlmann[265], as their 2005 paper provides the tables of data required to plot the velocity profiles. Their simulations were carried out by solving the incompressible Navier-Stokes equation using a Chebyshev pseudo-spectral solver on a $96 \times 96 \times 96$ grid, thus they wield a very high accuracy.

Here, the simulations are performed with the BGK LBM model using a D3Q19 node, the solid walls are implemented using the half-way bounce-back scheme, and the top wall velocity is imposed using 134, like in the previous section. The velocity of the lid is set to $U_{\text{lid}} = 0.1$ and the Reynolds number Re for a given resolution N is adjusted by modifying the relaxation time τ as follows (convective scaling)

$$\tau = \frac{1}{2} + 3\frac{U_{\text{lid}}\left(N-1\right)}{Re}. \quad (139)$$

As in the previous section, the velocity profiles are extracted along a vertical line (for the x-component) and a horizontal line (for the y-component). These profiles, shown on the figure 43, are in perfect agreement with the benchmark data. While most of the figures use a



(a) Re=100, N=256

(b) Re=100, N=16 and N=32

(c) Re=400, N=256

(d) Re=1000, N=256

Figure 43: Velocity profiles in the 3D lid-driven cavity using the D3Q19 LBM BGK model.

relatively large cubic resolution of 256^3, figure 43b shows that small resolution simulations (like 16^3 and 32^3) can also achieve good accuracy, although $N = 16$ is rather noisy, as can be seen by the sharp fluctuations on the figure. It is clear that as the Reynolds number increases, the effect of the additional end walls (in 3D) cannot be neglected; both $Re = 400$ and $Re = 1000$ show differences in the 3D velocity profiles compared to the 2D results presented in the previous section in figure 33.

To conclude, the 3D isothermal LBM gives a satisfactory accuracy for the lid-driven cavity problem, even with the BGK collision operator and even for small lattice resolutions, while providing an easy and stable implementation. It is also very computationally efficient, the running time ranges from a few seconds for the smallest resolution ($N = 32$) to a few dozens of minutes for the largest resolution ($N = 256$). It would be interesting to see how the LBM performs for higher Reynolds number, but no validated benchmark is available, because the flow quickly becomes unsteady and turbulent.

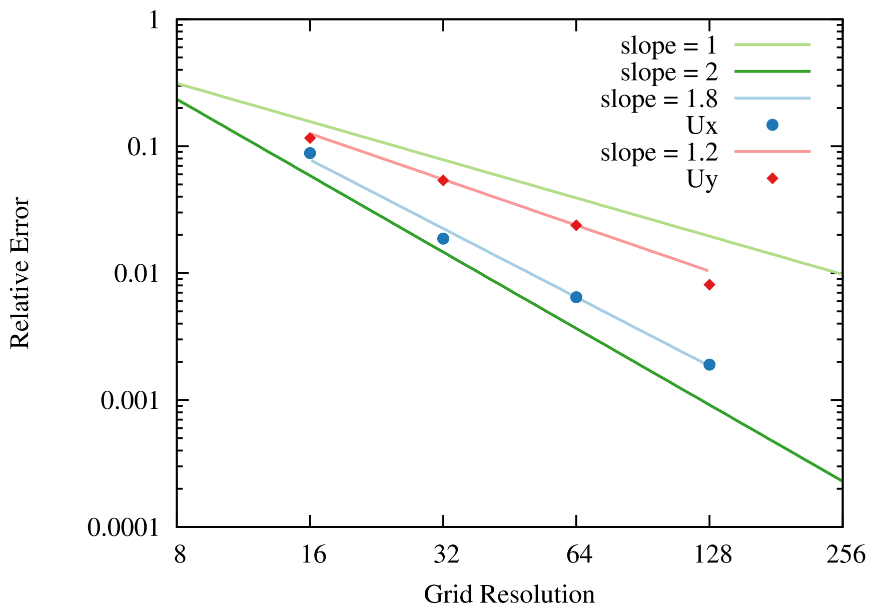


Figure 44: Grid convergence of the BGK LBM for the x and y velocity in a 3D lid-driven cavity flow for Re=100.

The convergence study is performed using the same procedure as for the two-dimensional case. The lid-driven cavity for $Re = 100$ is simulated with a range of resolutions: 16^3, 32^3, 64^3, 128^3, 256^3, the simulations are carried on until a steady-state is reached. Afterwards, the relative error is estimated by computing the L^2 norm of the velocity profiles compared against the highest resolution simulation ($N = 256$), which is assumed to be the most accurate and is taken as the "reference" (in the lack of analytical solution). A notable difference with the previous study is that this time a convective scaling is applied to limit the requirement on the computing time for the largest resolution.

The results for the grid convergence of BGK LBM for the three-dimensional lid-driven cavity are shown on the figure 44. It appears that the convergence order higher than for the two-dimensional case. The convergence order for the y-component of the velocity is approximately 1.2, like in the two dimensional case. The relative errors on the x-component are significantly smaller, and the convergence order appears to be higher, around 1.8. It is unclear what is the source of these differences, but the convergence order may be overestimated and may decrease if the resolution of the reference was increased further (see section 6.2.2).

Ultimately, this issue on the small convergence order of the LBM is more of an academic problem than a practical one. In practice, the LBM recovers the flow characteristics with a reasonable accuracy for lattice sizes that are sufficiently small to allow real-time CFD.

6.3.1 *Problem description.*

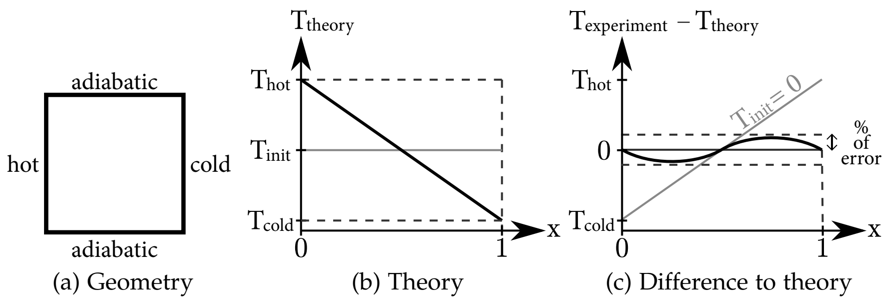


Figure 45: Thermal diffusion in square cavity.

This system is one of the simplest thermal problem possible: it is that of a 2D cavity of size $L \times L$, the left wall is heated at a constant temperatureT_{hot} and the right wall is cooled at a constant temperature T_{cold}, the Boussinesq term is turned off, i.e. no gravity ($\vec{g} = \vec{0}$), thus there is no buoyancy. The domain is initially set to a rest state with a constant temperature equal to the average temperature, i.e. $T_{init} = (T_{hot} + T_{cold})/2$. It is expected that after some time, a gradient of temperature appears in the cavity, created by the diffusion of the boundaries to the inside of the domain, while the velocity field stays at rest.

This system is simulated using a D2Q9 node for the velocity, and a D2Q4 node for the temperature. The boundary conditions are set to half-way bounce-back for all four walls for the velocity, i.e. no-slip boundaries. For the temperature, the top and bottom boundary conditions are set to half-way bounce-back, i.e. they are adiabatic; and the temperature is fixed on the left and right boundary conditions using either a *force equilibrium scheme* (eq. 141) or a *macroscopic recovery scheme* (eq. 142), in order to compare their accuracy.

The effect of the initial temperature T_{init}, lattice resolution N, relaxation time τ, boundary scheme and floating point precision are investigated. In theory the temperature profile in the room should be equal to

$$T_{theory}(x) = T_{hot} + (T_{hot} - T_{cold}) \cdot x \tag{140}$$

where x is the normalised position, in the simulation the location of the i^{th} node is computed by $x = (i + 1/2)/N$.

As the problem is essentially one dimensional, a profile of the temperature along the x-axis is taken from the simulation at the centre line of the cavity once the temperature has converged to a steady state. This profile is then compared with the theoretical expression for the temperature. As the two are very close under most settings, it is more practical to plot the difference between them rather than a superposition of the two plots. Initially, the difference is of the order of $(T_{hot} - T_{cold})/2$, but over time the boundaries diffuses into the domain and the difference reduces to a value close to zero. The percentage of error is then defined as the maximum of the absolute value of the error (see figure 45c).

6.3.2 *Effect of the choice of boundary condition*

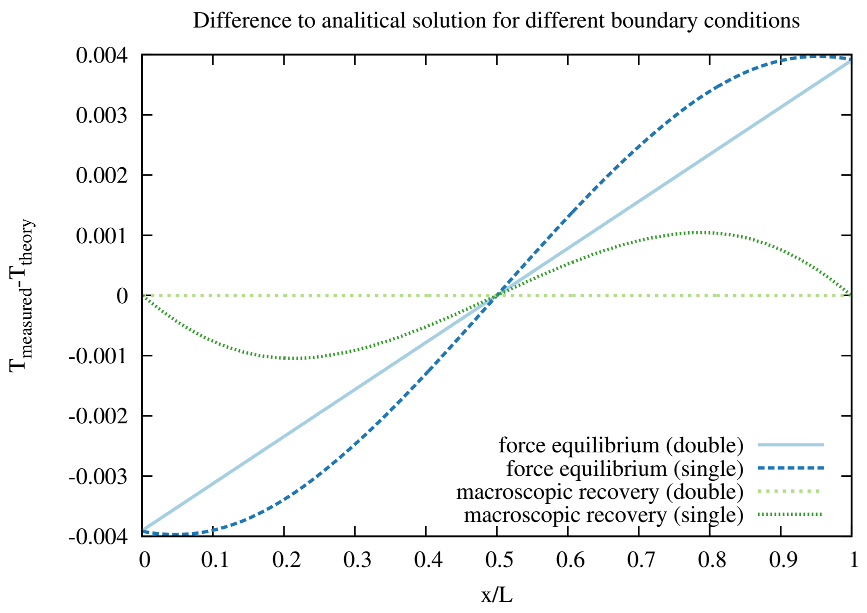


Figure 46: Effect of the boundary condition on the diffusion error.

The adiabatic walls for the temperature and all four walls for the velocity are using half-way bounce-back as described in section 3.2.3. The effects of using a full-way bounce-back are not studied. The temperature on the left and right wall is fixed, and two different boundary implementations for these walls are investigated:

1. Boundary type 1: “force equilibrium” scheme
   Description: the unknown distribution function is replaced by its equilibrium value, where the velocity of the fluid is zero.

$$f_{\text{unknown}} = \frac{1}{4} T_{\text{wall}} \tag{141}$$

2. Boundary type 2: “macroscopic recovery” shceme
   Description: the unknown distribution function is replaced by the difference between the macroscopic temperature at the wall T_{wall} and the sum of the known distribution functions.

$$f_{\text{unknown}} = T_{\text{wall}} - \left(\sum_{i \in \text{known}} f_i \right) \tag{142}$$

The final temperature profiles for both boundary types in single and double precision are shown in Fig 46. Although the “force equilibrium” scheme allows for a simpler boundary implementation it does not actually enforce the wanted temperature T_{wall} but a slightly different temperature $T_{\text{wall}} + \epsilon$, where ϵ is of the order of 0.4% of T_{wall}. The errors caused by this boundary condition clearly dominates the other errors. The "macroscopic recovery" scheme is shown to properly enforce the wanted macroscopic temperature, and the error fluctuations seen in the single precision plot are unlikely to come from the boundary condition. To summarise, the "force equilibrium" scheme allows for quick implementation at the price of small inaccuracies on the macroscopic temperature, but for just a little more complexity, the "macroscopic recovery" scheme recovers exactly the macroscopic temperature. Thus, it is recommend to use the second type in most situations. For a discussion of the advantage of using double precision, read the next section.

6.3.3 *Effect of the initial temperature.*

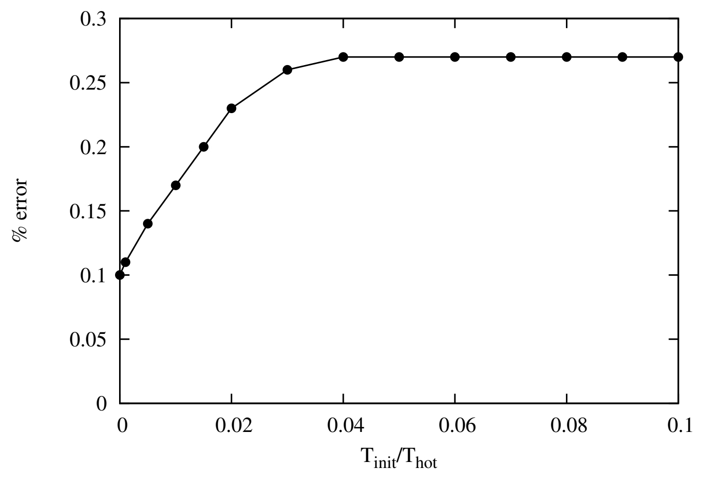


Figure 47: Effect of the initial temperature on the diffusion error.

To test the effect of the initial temperature on the accuracy of the results, several simulations are conducted using a 512^2 nodes lattice. The relaxation time for the temperature is set to $\tau_T = 1$ which is quite diffusive and allows to reduce the time to reach the final state. The temperature at the boundaries are set to $T_{\text{cold}} = -1$ and $T_{\text{hot}} = 1$ so that 0 is the average temperature and T_{init} can be easily expressed as a fraction of T_{hot}. The simulations are run in single precision.

As can be seen from Fig. 47, the diffusion error is minimum (0.10%) for $T_{\text{init}} = (T_{\text{hot}} - T_{\text{cold}})/2 = 0$ and quickly reaches its maximum (0.27%) as T_{init} varies. So apart from a very small interval $\Delta T_{\text{init}} = 0.06$ around 0 where the error is minimal, the diffusion error is always of the order of 0.27%. Running the simulation in double precision shows a similar behaviour, but both the size of the interval and the value of the maximum error are significantly smaller. Overall, the maximum percentage error measured in double precision is $5 \cdot 10^{-10}\%$. So the use of double precision significantly increases the accuracy of this simulation, but the 0.27% error of single precision is considered sufficiently small for the purpose of this thesis.

It is worth noting that if the temperature is initialised using the expression for the theoretical temperature, then the temperature in the cavity does not change with time, and the measured temperature profile equals exactly the theoretical formula up to floating point precision, i.e. 10^{-8} and 10^{-16} in single and double precision respectively.

In conclusion, the initial state of the flow has an impact (although minimal) on the resulting accuracy of the solution.

6.3.4 *Effect of the relaxation time*

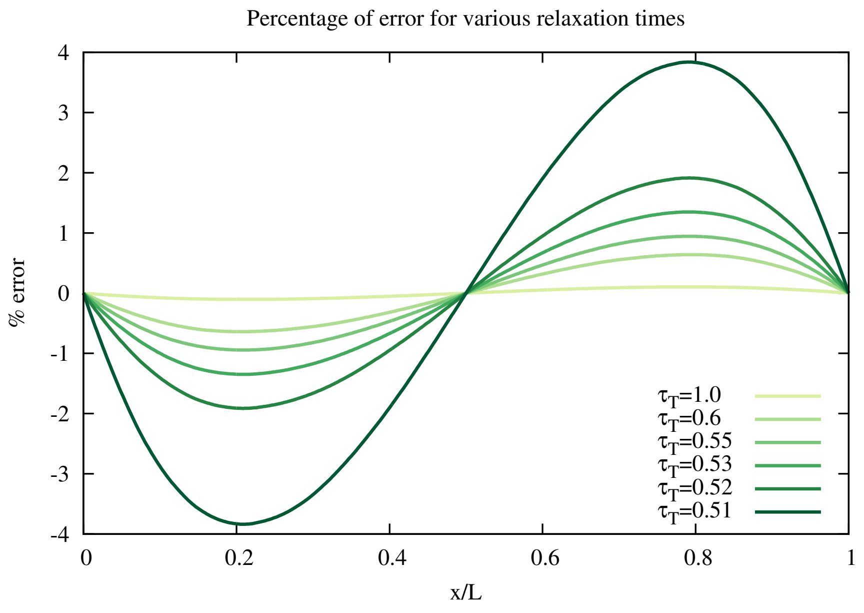


Figure 48: Effect of the relaxation time on the diffusion error.

The Figure 48 shows that as the relaxation time becomes closer to the limit $\tau_T = 0.5$ (no diffusivity), the temperature moves away from the theoretical profile. This is due to the fact that smaller relaxation time means smaller lattice diffusivity ($\mathcal{D}_{lb} = (\tau_T - 1/2)/2$), and the flow converges slower towards the fully diffuse state.

It is expected that this issue is not restricted to the LBM.

6.3.5 *Effect of the lattice resolution*

In order to further illustrate the problem of the diffusion with a low diffusivity, several simulations are conducted using different resolution but the same extremely small relaxation time $\tau_T = 0.501$ which

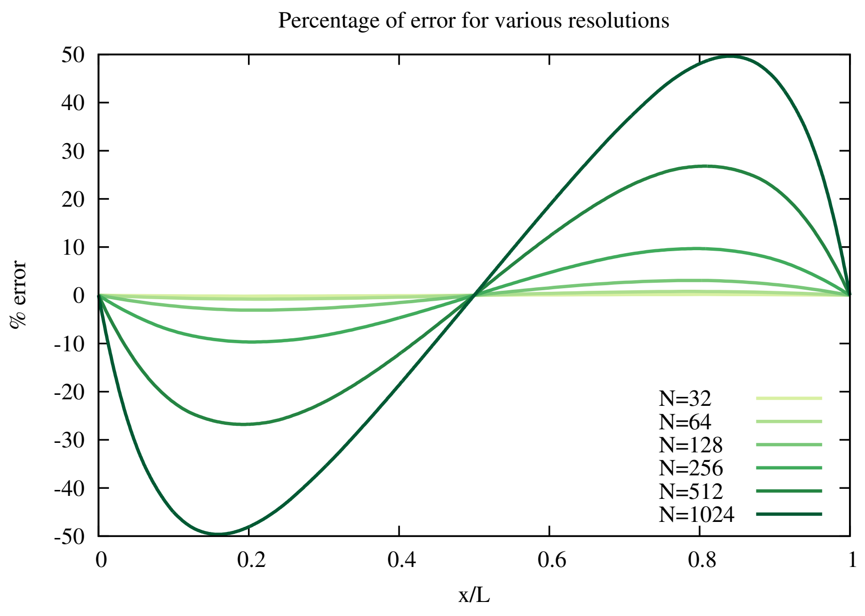


Figure 49: Effect of the lattice resolution on the diffusion error.

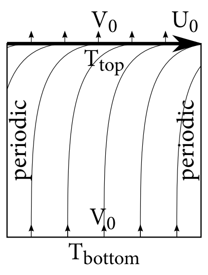
correspond to a lattice diffusivity of $\mathcal{D}_{lb} = 0.0005$. With this small diffusivity, it is difficult for the temperature at the boundary to diffuse to the inside of the domain. Thus the boundaries only diffuse up to a certain number of nodes, and as the lattice size increases so does the deviation to the linear temperature profile, as shown in Figure 49.

6.3.6 *Conclusion*

In conclusion to this section, it was shown that the program recovers the expected temperature profile with an error usually smaller than 1%. The program does struggle to recover the correct profile for very small relaxation time on large lattices. This shows that small relaxation time should be avoided when simulating diffusion driven flows, and that in such cases, increasing the resolution of the simulation will not improve the accuracy.

6.4 THERMAL ADVECTION IN A CHANNEL

6.4.1 *Problem description*



The previous section studied the accuracy of the program for a purely diffusive problem. The system studied in this section includes thermal advection as well as thermal diffusion but the buoyancy is still turned off so that an analytical solution still exists. It is again a 2D cavity of size $L \times L$, the bottom and top walls are maintained at constant temperatures, T_{bottom} and T_{top} respectively. The bottom wall has a constant normal velocity V_0 but no parallel component, the top wall has the same constant normal velocity V_0 but also has a parallel component U_0. This problem represents a flow through a porous media submitted to a gradient of temperature and a shear stress. The left and right walls are periodic, thus the flow is independent of x. It is assumed that a steady state exists, and that once the steady state is reached, the y-component of the velocity becomes constant and equal to V_0.

6.4.2 *Analytical solution*

For this problem, the solution to the incompressible Navier-Stokes equations can be found analytically. The equation on the x-component of the velocity can be written as

$$\rho\left(\frac{\partial u_x}{\partial t}+u_x\frac{\partial u_x}{\partial x}+u_y\frac{\partial u_x}{\partial y}\right)=-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u_x}{\partial x^2}+\frac{\partial^2 u_x}{\partial y^2}\right) \tag{143}$$

All the derivatives in x are null as there is no dependence on x, and as $u_y = V_0$ for a steady state, this equation can be simplified into

$$\frac{\partial^2 u_x}{\partial y^2}=\frac{V_0}{\nu}\frac{\partial u_x}{\partial y} \tag{144}$$

A general solution to this second order differential equation has the form

$$u_x(y)=Ae^{\mathrm{Re}\frac{y}{L}}+B \tag{145}$$

Where $\frac{V_0}{\nu}y$ has been rearranged to $\frac{V_0 L}{\nu}\frac{y}{L}=\mathrm{Re}\frac{y}{L}$. Using the boundary conditions $u_x(y=0)=0$ and $u_x(y=L)=U_0$, the values of A and B can be determined. The final expression for the velocity is then obtained as

$$u_x(y)=U_0\frac{e^{\mathrm{Re}\frac{y}{L}}-1}{e^{\mathrm{Re}}-1} \tag{146}$$

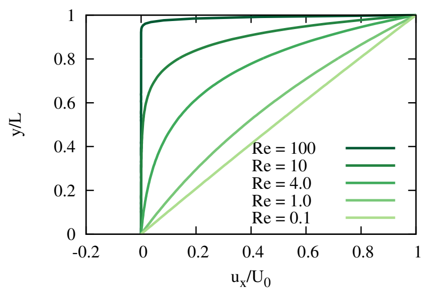
The advection diffusion equation on the temperature can be solved in a similar way.

$$\frac{\partial T}{\partial t}+\frac{\partial u_x T}{\partial x}+\frac{\partial u_y T}{\partial y}=\mathcal{D}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right), \qquad \frac{\partial^2 T}{\partial y^2}=\frac{V_0}{\mathcal{D}}\frac{\partial T}{\partial y} \tag{147}$$

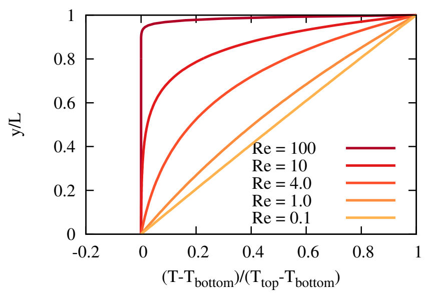
$$\frac{V_0}{\mathcal{D}}y=\frac{\nu}{\mathcal{D}}\frac{V_0 L}{\nu}\frac{y}{L}=\mathrm{RePr}\frac{y}{L} \tag{148}$$

$$T(y)=T_{\mathrm{bottom}}+\left(T_{\mathrm{top}}-T_{\mathrm{bottom}}\right)\frac{e^{\mathrm{RePr}\frac{y}{L}}-1}{e^{\mathrm{RePr}}-1} \tag{149}$$

The flow in the cavity can be simulated for various parameters and compared to the analytical solution.



(a) Normalised velocity.



(b) Normalised temperature.

Figure 50: Profiles across the channel for varying Reynolds number.

6.4.3 *Results and Discussion*

Figure 50 shows several profiles for various Reynolds number of the x-velocity and the temperature. The simulations are computed with a resolution $N = 128$, a normal velocity $V_0 = 0.01$, a parallel velocity $U_0 = 0.1$ and the following temperatures: $T_{bottom} = 0$, $T_{top} = 1$, $T_{init} = 0.5$.

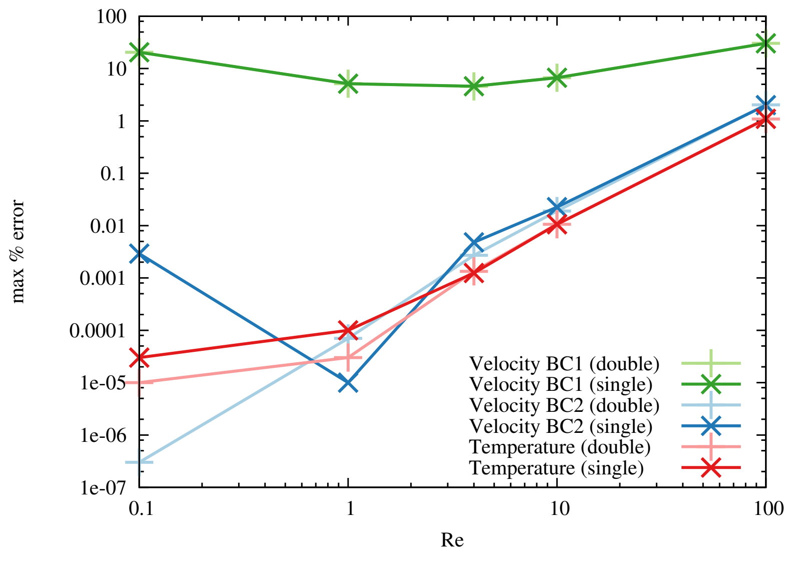


Figure 51: Percentage of error compared to the analytical solutions for various boundary conditions and precision.

The viscosity τ is computed to achieve the required Reynolds number, and the diffusivity τ_T is adjusted to obtain a Prandtl number of 0.75, similar to that of the air. The profiles are extracted once the velocity and temperature have both reached a steady state and they are normalised to range between 0 and 1.

For small Reynolds number, u_y is close to zero and the channel behaves like a Couette flow, the velocity has a linear profile. As the Reynolds number increases, the thickness of the boundary layer near the top wall decreases. This problem only allows to study relatively small Reynolds number, as the exponential term e^{Re} appearing in the analytical solution quickly exceeds the machine accuracy for $Re > 700$, i.e. $e^{700} = 10^{304}$, even in double precision. The effect of the Prandtl number are not displayed because the Prandtl number does not affect the velocity profile and it has the same effect on the temperature profile as the Reynolds number, as they only appears in conjunction with each other, i.e. through the term e^{RePr}. The sharpness of the boundary layer for high Reynolds number makes the solving of the flow in the region of the top boundary more difficult, and can be a source of errors, as shown in Figure 51.

Figure 51 compares the maximum percentage of error on the velocity and temperature profiles for different Reynolds number and boundary condition and it uses a logarithmic scale because of the massive fluctuations. The maximum error is defined as the largest absolute difference between the simulation and the analytical solution for all values of y. The first boundary condition (BC1) corresponds to the "force equilibrium" scheme and deviates from the analytical solution between 5% and 30%. This can be significantly improved by using a Zou-He scheme (see section 3.2.5), which correspond on the Figure 51 to BC2. The use of single or double precision floating point operation is shown to make little to no difference in the results.

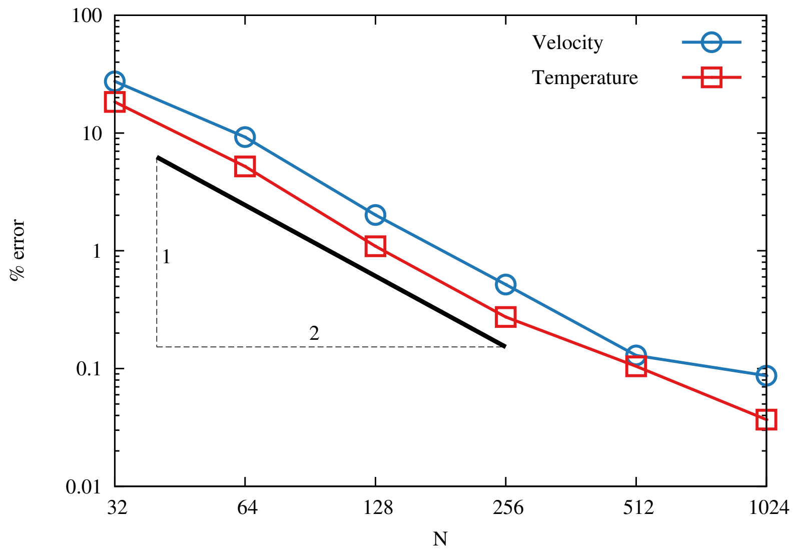


Figure 52: Percentage of error at $Re = 100$ for various resolutions.

Using the correct boundary scheme, the error is always smaller than 2% even for high Reynolds number, and falls under 0.02% for $Re < 10$. A 2% accuracy is considered sufficient for the needs of a real-time solver, but this can be improved by increasing the resolution, as shown in Figure 52. Indeed the sharpness of the boundary layer is the main source of inaccuracies for this problem, so increasing the resolution increases the number of nodes available to model this boundary layer, thus improving the accuracy. The slope of the curve indicates that the error is proportional to δx^2, i.e. the LBM shows a second order convergence under spatial refinement for this problem.

6.4.4 *Conclusion*

The thermal LBM performs reasonably wells for this benchmark. The analytical profile is recovered with less than 2% difference for a range of Reynolds number. The LBM solver can actually outperform the analytical solution and simulate Reynolds number of the order of the thousand (although with some oscillations) while the analytical solution cannot be computed for $Re > 700$. The use of single precision does not reduces the accuracy significantly, thus it is advised to use with single precision for such problems, as the computations will be twice as fast.

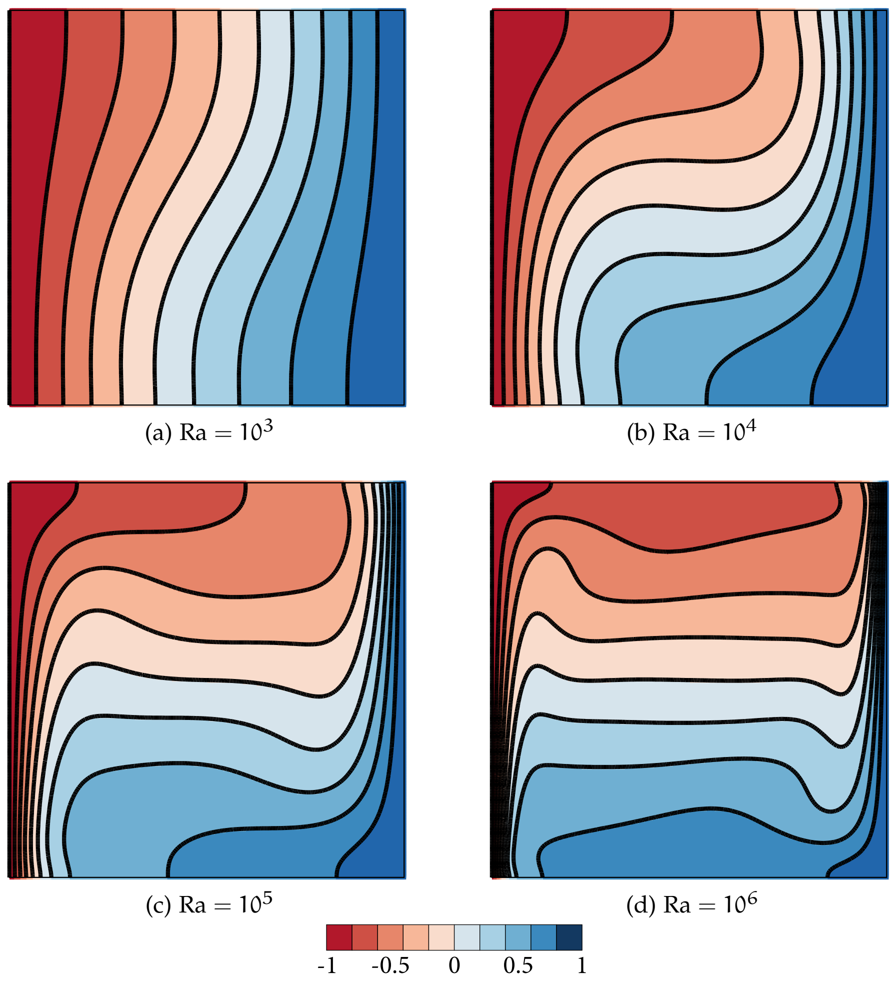
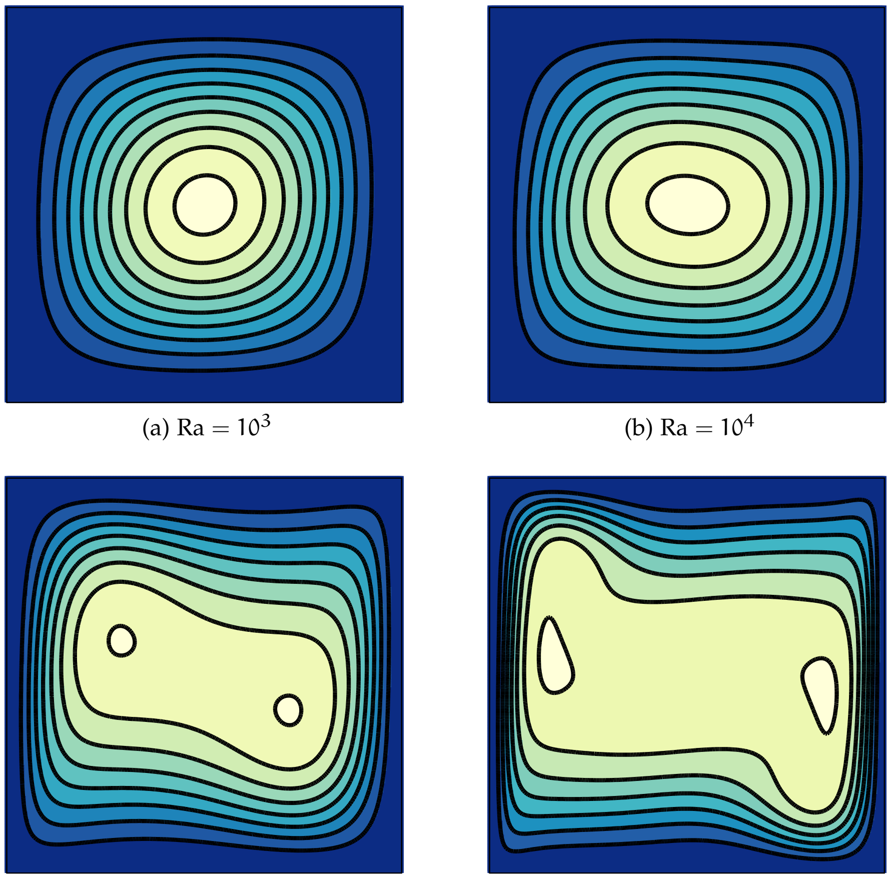


Figure 53: Contour maps of temperature.

In the previous test cases, the Boussinesq term was turned off, thus there was no buoyancy and the temperature was either purely diffusing or solely being advected by the flow. This new test case is radically different, as the dynamics of the flow will be mostly driven by the buoyancy.

The system is very similar to that of Section 6.3, i.e. a cavity with a cold and hot walls on opposite sides, but in this section the system is under the influence of a constant gravity field $\vec{g}$ resulting in the Boussinesq force $\vec{F}_B = -\vec{g}\beta\,(T - T_0)$. The thermal gradient $\Delta T = T_{hot} - T_{cold}$ between the hot and cold walls leads to varying buoyancy forces: the fluid near the hot wall expand and rises while the fluid near the cold wall contracts and falls. A global circulation of the fluid appears in the cavity : the fluid rises on the hot side, moves along the top of the cavity then cools down when reaching the cold side and falls, moves along the bottom of the cavity and reaches the hot wall, etc...


(a) Ra $= 10^3$ (b) Ra $= 10^4$

(c) Ra $= 10^5$ (d) Ra $= 10^6$

Figure 54: Contour maps of stream function.

This problem does not have an analytical solution for the velocity or the temperature fields, but it is a particularly common benchmark for CFD solvers, and several numerical experiment can be found in the literature[270, 271, 272, 273, 274].

It can be shown that the behaviour of the fluid in the buoyancy driven cavity is determined by only two parameters : the Rayleigh number (Ra) and the Prandtl number (Pr). The Rayleigh number describes the ratio of the amount of heat transferred by convection to the amount of heat transferred by conduction, it is defined by the following equation.

$$\mathrm{Ra} = \frac{g\beta\Delta T L^3}{\nu D} \tag{150}$$

It can be rewritten in term of the lattice Boltzmann parameters as

$$\mathrm{Ra} = 6\frac{g\beta\Delta T\,(N-1)^3}{(2\tau-1)\,(2\tau_T-1)} \tag{151}$$

The Prandtl number is the ratio of the kinematic viscosity ν to the thermal diffusivity κ.

$$\mathrm{Pr} = \frac{\nu}{\kappa} \tag{152}$$

It can be rewritten in term of the lattice Boltzmann parameters as

$$\mathrm{Pr} = \frac{2\left(2\tau - 1\right)}{3\left(2\tau_T - 1\right)} \tag{153}$$

The results presented in this section compare velocity profiles obtained using the thermal LBM solver running on the GPU with the results obtained with the popular commercial finite volume solver Fluent, for a large range of Rayleigh numbers. Although the results obtained with Fluent are not taken as benchmark, they were confirmed by two other commercial software (CFX and Comsol), so they can be assumed to be reasonably accurate. Using the results from Fluent, rather than published results, for the comparisons allows to vary freely the parameters and to study a larger range of Rayleigh number.

6.5.1 *Methodology*

As in the previous sections, the simulations with the LBM were computed for a large number of time-steps, until the flow reaches a steady state, while Fluent solved directly for a time independent solution. In all the simulations, the Prandtl number is fixed to $\mathrm{Pr} = 0.789$, i.e., that of the air, and the resolution of the simulations is 256×256 for both LBM and Fluent apart from subsection 6.5.3 which studies the effect of varying the resolution. The LBM parameters for the simulation are set using the following methodology.

1. The boundaries temperatures are fixed.

$$T_{\mathrm{hot}} = 1, \quad T_{\mathrm{cold}} = -1, \quad T_{\mathrm{init}} = \left(T_{\mathrm{hot}} + T_{\mathrm{cold}}\right)/2 = 0 \tag{154}$$

2. τ is chosen arbitrary. (cf. discussion on the next section)

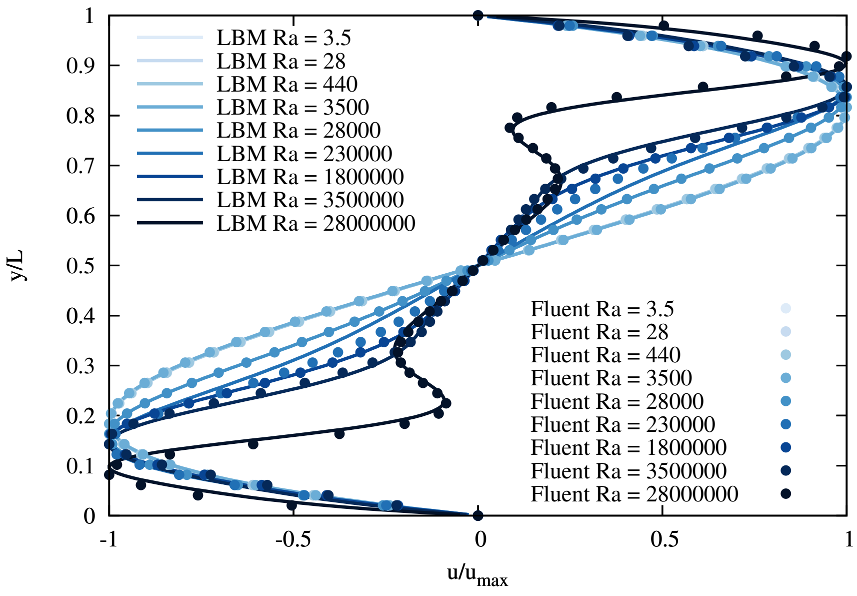
3. τ_T is computed to recover the Prandtl number.

$$\tau_T = \frac{1}{2} + \frac{2\left(\tau - \frac{1}{2}\right)}{3\mathrm{Pr}} \tag{155}$$

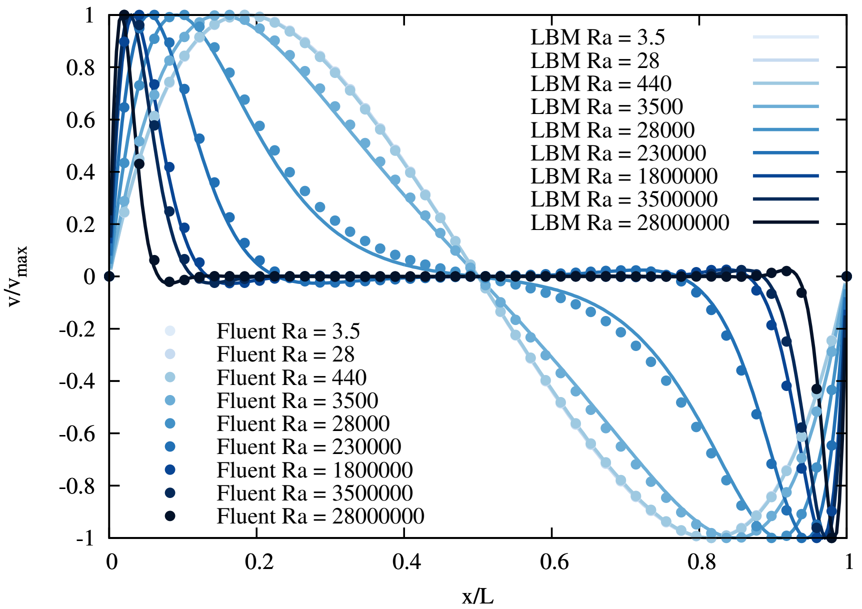
4. $g\beta$ is computed to recover the Rayleigh number.

$$g\beta = \frac{\mathrm{Ra}\left(\tau - \frac{1}{2}\right)\left(\tau_T - \frac{1}{2}\right)}{6\Delta T\left(N-1\right)^3} \tag{156}$$

6.5.2 *Effect of the Rayleigh number.*



(a) u-velocity



(b) v-velocity

Figure 55: Normalised velocity profiles through the cavity geometric centre for various Rayleigh number.

For small Rayleigh number, the effects of the buoyancy are small and the temperature in the cavity behaves like in the purely diffusive cavity of Section 6.3, the isocontours of temperatures are mostly vertical lines with only small sinusoidal deviations. For higher Rayleigh number, the effects of the buoyancy becomes higher at the wall, as the temperature diffuses less, and this results in the formation of a boundary layer at the walls, and the isocontours of temperature become more horizontals. The highest Rayleigh number are harder to solve because the thickness of the boundary layer becomes very small, requiring high resolution meshes. Contour maps of velocity stream function and temperature for Rayleigh numbers varying from 10^3 to 10^6 are shown in Figure 54 and Figure 53 respectively. They are visually in good agreement with [270].

The Figure 55 is a comparison of the normalised velocity profile in the cavity between LBM (solid lines) and Fluent (circles), both are using the same 256×256 resolution. The u-velocity is measured along a vertical line at $x = L/2$ and the v-velocity is measured along an horizontal line at $y = L/2$. They represents Rayleigh number varying from 3.5 to 2.8×10^7 and both simulations agree reasonably well within the whole range of Rayleigh numbers. Rayleigh number smaller than 440 give the same normalised profile, thus they overlap on the figure.

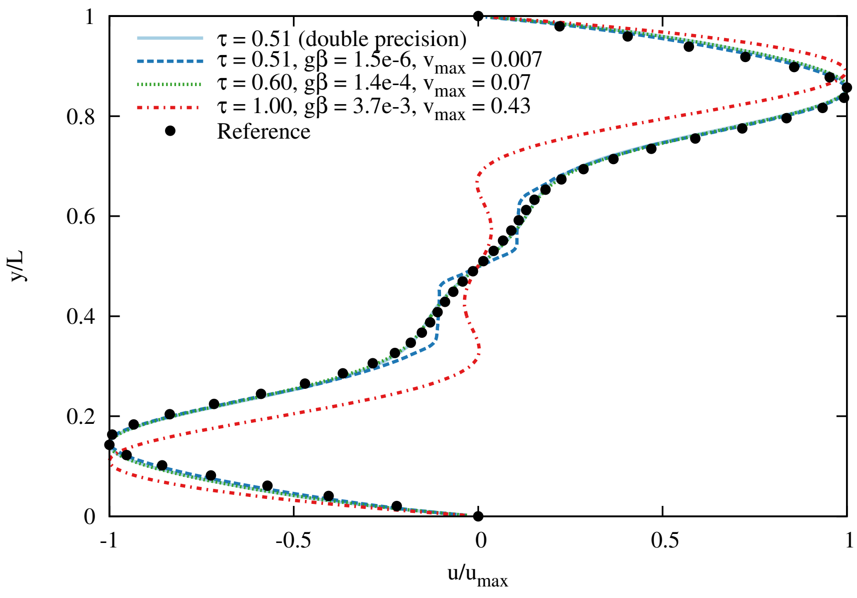


Figure 56: x-velocity profile though the cavity geometric centre for $Ra = 3.6 \times 10^6$ and various relaxation time τ.

Most simulations were achieved using single precision computations with the same relaxation time $\tau = 0.6$. However, two issues can arise from this choice of relaxation time.

1. For small Rayleigh number, the coefficient of thermal expansion $g\beta$ becomes very small (ex. $g\beta = 1.9 \times 10^{-8}$ for $Ra = 440$ and $\tau = 0.6$). It is thus necessary to either increase τ when Ra becomes small or to use double precision. The smallest possible

value allowed for $g\beta$ in single precision appears to be around 10^{-6}.

2. For high Rayleigh number, the opposite happens, $g\beta$ becomes large, resulting in high velocities near the wall. If these velocities are too high the LBM performs poorly, as it is only valid for small Mach number flows, and there is a risk that the simulation becomes unstable. It is thus advised to use smaller relaxation time for high Rayleigh number flows in order to reduce $g\beta$ and the resulting maximum velocity. If reduced too much though, the system falls back into issue number one (need for double precision). This behaviour is illustrated by the Figure 56 for $Ra = 3.6 \times 10^6$.

6.5.3 *Effect of the resolution.*

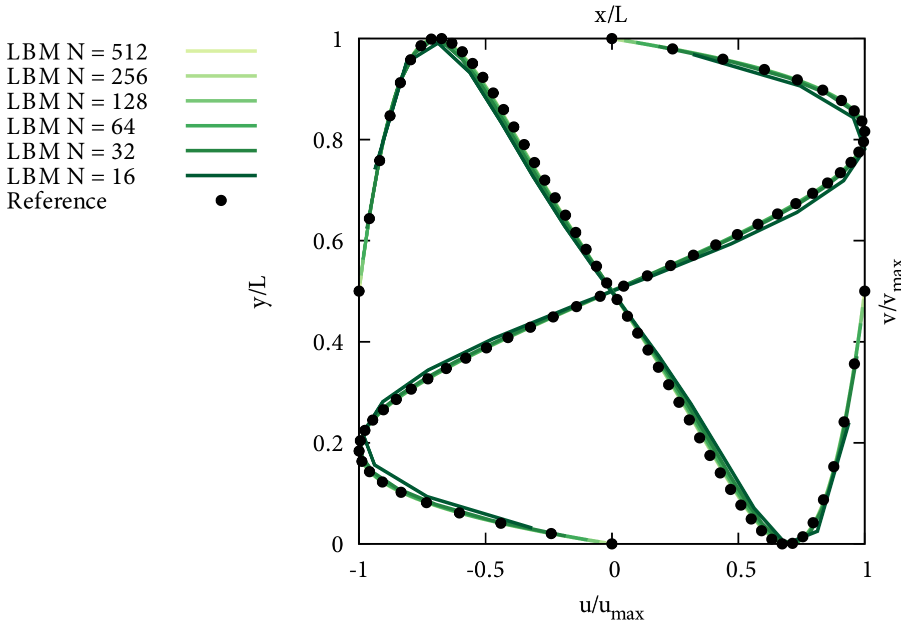


Figure 57: Normalised velocity profiles for $Ra = 3500$.

The resolution, i.e. the number of nodes used to simulate the cavity, affects the results differently depending on the Rayleigh number.

1. For small Rayleigh number, where the buoyancy effects are small, even very small resolutions ($N = 16$) can recover approximately the behaviour of the flow. And the velocity profiles match very closely the reference (Fluent) as soon as $N \geqslant 32$. As the simulation is using a half-way bounce-back to implement no slip at the walls, the actual location of the wall is outside of the domain. Thus it can be seen on the velocity profiles in Figure 57 that they never quite "reach the sides", especially for small resolution.

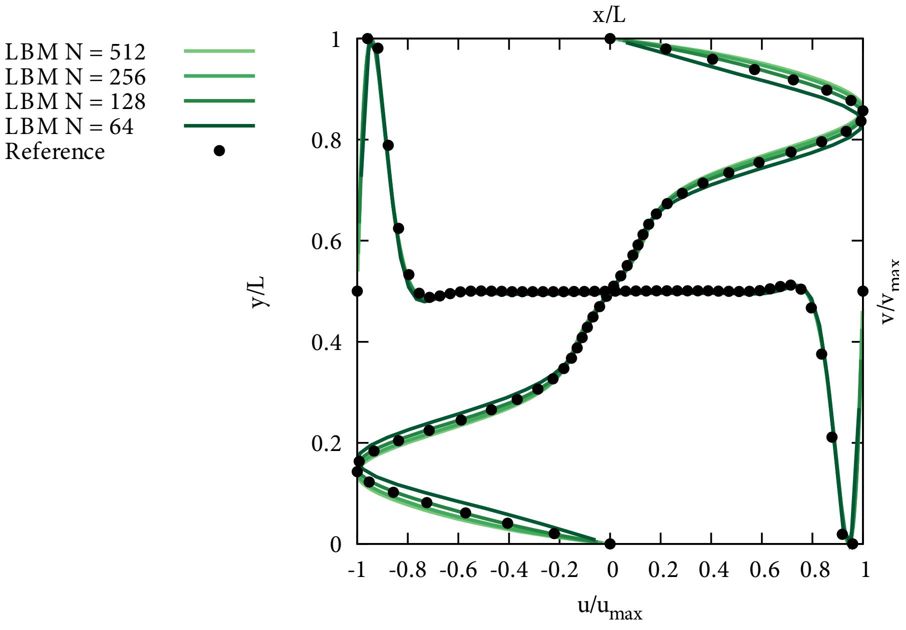


Figure 58: Normalised velocity profiles for $Ra = 3500000$.

1. For high Rayleigh number, the buoyancy effects are large, and the boundary layers are thin. High Rayleigh require either high value of $g\beta\Delta T$ or very small relaxation times, especially for small resolutions. Consequently, high Rayleigh number simulations are unstable when using small resolutions, and they could use the hep of a turbulence model. Figure 58 shows some simulations of $Ra = 3500000$ at various resolution starting at $N = 64$, smaller resolutions are unstable. For the profiles of the x-component of the velocity, $N = 128$ seems to give the best match with the Fluent simulation. For the y-component, there is no visible difference.

6.5.4 *Effect of the forcing scheme*

In section 2.3.5, it was shown that there are many possible schemes to add a forcing term to the LBM equations. In particular, Guo's scheme [111] was designed to reduced the lattice effects on the equivalent macroscopic equations compared to more simple schemes. The Figure 59 compares the velocity profiles obtained using the Guo's scheme and with the simple scheme that adds the force directly to the two distribution functions aligned with it. There is little to no difference in the profiles, although Guo's scheme does appear to be slightly more accurate. From this results, it is thus advised to use the simple scheme, as it is easier to implement and does not noticeably affect the quality of the simulation.

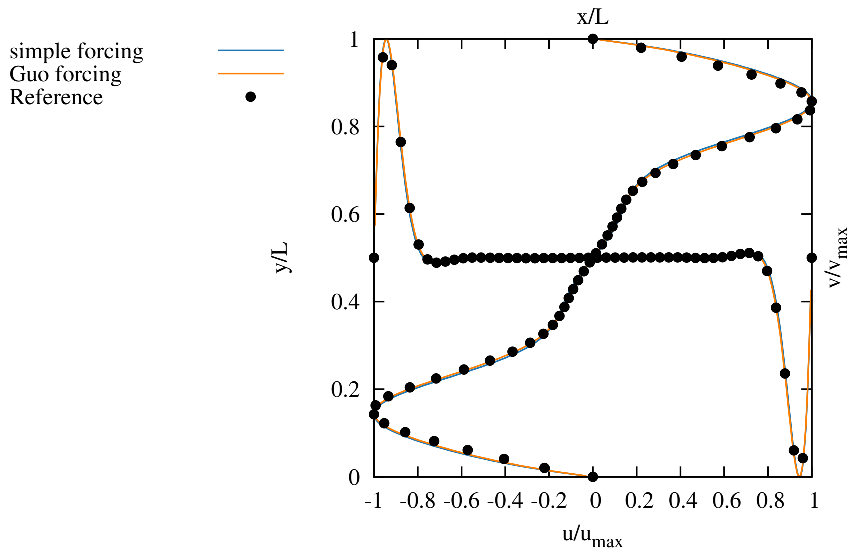


Figure 59: Normalised velocity profiles for Ra = 3500000 and different forcing scheme.

6.6 SUMMARY

Throughout this chapter the LBM on GPU program developed during this thesis has been successfully validated for several benchmark problems for which the simulation results (or analytical expressions) are readily available in the literature. The program performs very well for these test cases, as if often matches perfectly with the benchmark. And the use of single precision floating point for the calculation allows for faster computations while it did not show a significant decrease in accuracy compared to double precision, provided that the simulation parameters are chosen judiciously.

The deterioration of the spatial convergence from second to first order for large lattice resolution in the case of the lid-driven cavity is a worrying feature, but the simulation results are still close to the benchmark even for small resolutions. The MRT and cascaded LBM were shown to be superior to the BGK, especially in term of dissipating the fluctuations of the pressure field.

Further validation work could be done, such as investigating the flow behaviour in transient scenarios or validity of the turbulence model. However, they will both be discussed in the next chapter, for the simulation of transient turbulent indoor air-flows.

While it is important to validate a new simulation code against solid benchmarks, it should be noted that the models used in this thesis for the simulation of indoor air flows have been validated in the literature before. Hence the accuracy of the program should be expected to be identical to that reported by the original papers.

7 APPLICATION TO INDOOR AIR FLOWS

7.1 INTRODUCTION

The accurate simulation of indoor air flows for building performance is a challenging task. Buildings are complex systems interacting in multiple ways with their environment and the accurate modelling of the heat transfer and mass transfer between the outdoor and indoor environments requires large CFD simulations that can be of prohibitive computational cost.

As a result, it is common practice to resort to simplified empirical formulae [275] but they only provide a crude approximation [276], often in an averaged form, and suffer from important physical deficiencies [277]. Regarding indoor simulations, the most popular methods are nodal [278], zonal [279] and multi-zonal models [280]. These models represent the air in a room with either a single node or a few cells by approximating the influence of the bulk parameters. They are thus very computationally efficient but suffer in terms of physical accuracy.

The use of CFD, sometimes combined with multi-zonal models [280], in the assessment of building design has gained popularity over the past few years [281], despite its high computational cost. CFD has been applied at each stage of a building design and can provide detailed information about outdoor airflows around buildings as well as parameters in the indoor environment, such as air velocity, temperature and contaminant concentrations.

CFD has been used to analyse the thermal environment [282], design of ventilation systems [283] and for evaluating indoor air quality [284]. Furthermore, CFD methods have been used to study smoke dispersion in buildings [285] and model environment specific parameters such as airborne pathogen transport [286] inside hospitals.

While these CFD studies yield valuable information that informs building design and information, the computational cost and memory requirements are so high that the majority of models are restricted to steady-state scenarios which are not able to capture the transient effects due to factors such as the movement of people, changes to heat sources or ventilation speed. The computational complexity of transient CFD models is such that the calculation time might extend from hours to months [287, 288]. Hence, CFD based models are limited in their ability to provide quick evaluation at conceptual design stage, and in conducting risk assessments for early warning systems such as smoke management in case of building fire or transmission of airborne infectious contaminant spreading in hospital wards, or prediction and control of thermal loads in a data centre.

The computational time can be significantly reduced by accelerating the CFD simulation on high performance computing (HPC) facilities [289]. But since the access to these facilities is limited and expensive [288], novel alternative approaches must be developed in order to simulate indoor air distributions on a personal desktop computer, with a good physical accuracy and within an acceptable computing time.

Recently, the fast fluid dynamics (FFD) method (see introduction in section 1.3 and details in Appendix C) has been used to simulate indoor environment in real-time on both CPU and GPU [288, 287, 290]. Although more accurate than zonal models, the FFD algorithm in its standard form lacks the accuracy of CFD (see section C.5) but some recent works [287, 288] on improving the accuracy have shown some promising outcomes.

The present work explores the potential of GPUs to perform indoor airflow CFD simulations using the LBM. The LBM models described in chapter 2 implemented and optimised on the GPU (details in chapter 4) are used to investigate the airflow dynamics for practical engineering applications such as a ventilated room, a data centre, and a hospital room. The results of the ventilated room and data centre simulations, performed in real-time, are compared against the results obtained with some of the prevalent commercial CFD software (see brief in 1.2.5); while the hospital room provides solely qualitative results to study the applicability of the method in this type of problem.

7.2 VENTILATION CHAMBER

Two mains techniques are available for the mechanical ventilation of an office-sized room: mixing ventilation and displacement ventilation [291]. This section considers the case of mixing ventilation in an empty room, where the air is supplied to the room at relatively high velocity, causing a high degree of mixing to take place. The geometry used in the simulation is based on a $32\,\mathrm{m}^3$ bio-aerosol chamber (Class II) built in the school of Civil Engineering at the University of Leeds [292] that was used to study aerial dispersion and spatial deposition of pathogenic microorganisms in hospital rooms.

The simulation are performed using the D3Q19 LBM model with the LES Smagorinsky turbulence model and a D3Q6 lattice to solve for the advection diffusion equation for the temperature (see respective details in sections 2.5.1 and2.3.5). The temperature and velocity fields from the LBM simulation are compared against those obtained from the commercial CFD software ANSYS Fluent v13 (ANSYS 2010) based on the Navier-Stokes equations with an equivalent LES turbulence model.

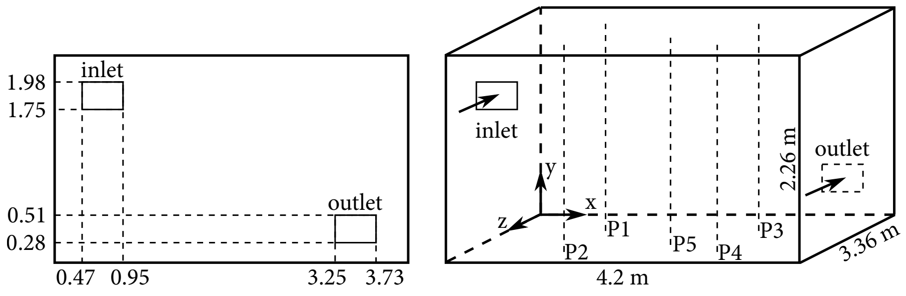


Figure 60: Ventilated test chamber geometry, showing the location of supply and extract vents and the position of the "poles" for velocity and temperature comparison. Left figure is a front view with inlet and outlet locations in meters. Right figure is a 3D representation.

7.2.1 *Problem Description*

The geometry of the room is shown in Figure 60. Warm air is supplied to the room through a high level wall mounted inlet on one side and exits the room through a free outlet (Neumann boundary) located at the bottom of the opposite wall. The volume flow rate is maintained constant so that the room is subject to 6 ACH (air changes per hour), i.e. an average inlet velocity of $u_0 = 0.48$ m/s normal to the inlet. Hence the Reynolds number computed from the hydraulic diameter of the inlet (i.e. $2\,(h \times w)\,/\,(h + w)$) is $Re = 10\,200$. The supplied inlet air temperature is $T_{in} = 22\,°C$, and the walls are maintained at a constant temperature $T_{wall} = 15\,°C$. The initial room temperature is the average of T_{in} and T_{wall}, and is taken as the reference temperature for the Boussinesq force $T_0 = 18.5\,°C$. The LBM simulation is performed on a regular cubic mesh of 1.7 million nodes, which was found to give a good trade off between speed and accuracy [46], while the Fluent simulation is using about half the number of nodes (533 918) but with inlet and outlet mesh refinements and the default wall function for near-wall treatment of the sub-grid turbulence.

The time-step used in the Fluent simulation is $\Delta t_{Fluent} = 0.01$ s, while the one used in LBM is about half of that (see table 8). Indeed to match the time-step for the chosen spatial resolution would have required to increase the inlet velocity in lattice units to $u_{0,lbm} = 0.48\Delta t_{Fluent}/C_L \simeq 0.18$, resulting in high numerical instabilities. More details on unit conversion can be found in the appendix B.

In both models, the flow is simulated for 460 seconds so that it can develop and become statistically steady; then the velocity and temperature fields are averaged for a further 100 seconds to allow for a steady-state comparison.

The LES results for the Fluent simulations are considered to be converged when the residuals reaches less than 10^{-5} at every time-step. These transient simulations are very computationally expensive, thus they were performed on a server with 16 CPU processor cores for

Parameter	Fluent (physical units)	LBM (lattice units)
Room height (H)	$2.26\,\text{m}$	86
Room width (W)	$3.36\,\text{m}$	127
Room lenght (L)	$4.20\,\text{m}$	160
Inlet height (h)	$0.23\,\text{m}$	9
Inlet width (w)	$0.48\,\text{m}$	18
Length conversion (C_L)	$C_L = 4.2/160 \simeq 0.026\,\text{m}$	
Inlet velocity u_0	$0.48\,\text{m/s}$	0.1
Speed conversion (C_V)	$C_V = 0.48/0.1 = 4.8\,\text{m.s}^{-1}$	
Time step Δt	$\Delta t_{\text{Fluent}} = 0.01\,\text{s}$	$\Delta t_{\text{lbm}} = C_L/C_V$ $\Delta t_{\text{lbm}} \simeq 0.005\,\text{s}$
Inlet Temperature T_{in}	$22\,°\text{C}$	7
Wall Temperature T_{wall}	$15\,°\text{C}$	0
Reference Temperature T_0	$18.5\,°\text{C}$	3.5
Fluid density ρ	$1.225\,\text{kg/m}^3$	1.0
Kinematic viscosity	$\nu = 1.46 \times 10^{-5}\,\text{m}^2/\text{s}$	$\tau - \frac{1}{2} = 3\nu\frac{C_L^2}{C_T}$ $\simeq 1.8 \times 10^{-6}$
Thermal Diffusivity	$D = 1.96 \times 10^{-5}\,\text{m}^2/\text{s}$	$\tau_T - \frac{1}{2} = 3D\frac{C_L^2}{C_T}$ $\simeq 7.3 \times 10^{-6}$
Prandtl number Pr	0.75	
Reynolds number Re	10 200	
Smagorinsky constant C_S	0.1	0.04
Turbulent Prandtl Pr_t	0.85	0.85

Table 8: Computational domain and simulation parameters used in ANSYS CFD and LBM to simulate the flow inside the ventilated chamber. The conversion factor for LBM are explicitly computed.

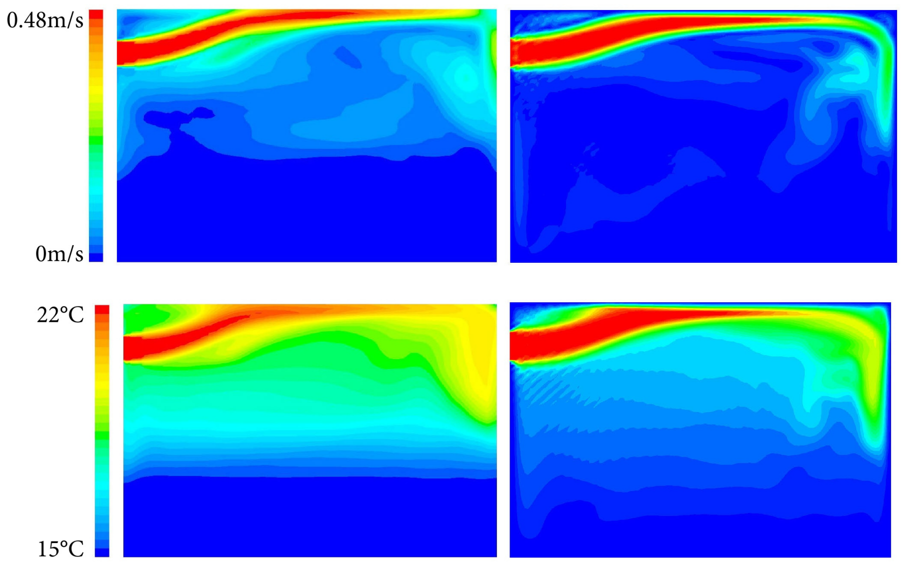
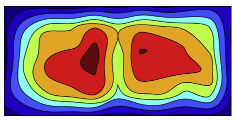


Figure 61: Steady-state comparison between FLUENT (left plots) and LBM (right plots) simulations of the ventilated chamber. Contours of normalised mean velocity (top plots) and normalised mean temperature (bottom plots) are shown on a place at the centerline of the inlet.

a total simulation times of 65 hours, i.e. 7 minutes of computations for each physical second. On the other hand, the LBM simulation running on a single Tesla K40 GPU requires 0.34 seconds to compute one physical second, i.e. three times faster than real-time for a total simulation time of only 3 minutes.

The second part of this study considers the transient behaviours of the flows in the first five seconds. In this part, the flow considered is isothermal and the inlet velocity has a varying profile (see figure) due to the presence of a grid vent in the experimental measurements [292] and the inlet jet is inclined at a 15 degree angle to the normal.



Inlet velocity field based on experimental data.

7.2.2 *Results*

Figure 61 shows the normalised contour plots of the time averaged velocity magnitude and temperature across a plane through the centreline of the inlet. While Fluent can directly export contour plots of the flow with a certain colour scheme, the contour plots of LBM were obtained by post-processing the images of the slice, using a similar colour scheme and limiting the number of colours, thus lacking the proper interpolations that could be achieved with a contour algorithm. Nevertheless, in both cases, the contours shows the effect of buoyancy on the incoming airflow, the warm inlet jet rises and comes in contact with the cold ceiling then drops towards the floor as it reaches the end of the chamber. While both LBM and FLUENT results show similar looking flow features, the LBM simulation obtains

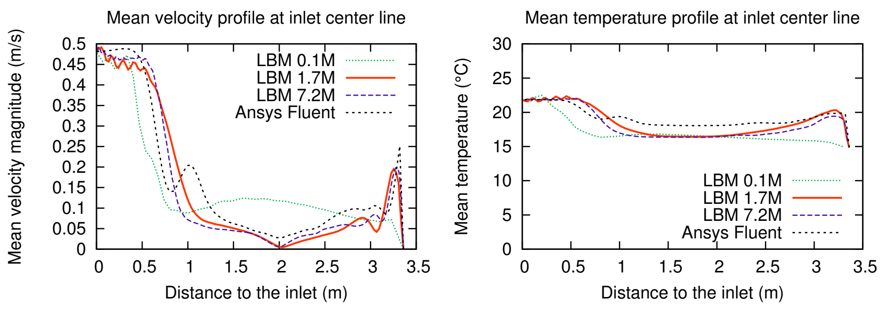


Figure 62: Comparison of the mean velocity profile (left) and the mean temperature profile (right) across the room at the inlet centreline, computed from LBM and ANSYS FLUENT simulations.

a larger mixing of the temperature while FLUENT results shows a more stratified temperature field, where the bottom half of the room is cold and the top half is hot. Moreover, the thickness of the boundary layer in FLUENT is significantly smaller, this partly due the way FLUENT export the data: the temperature and velocity at the walls are not displayed on the image, only the internal flow is displayed. This explains why the velocity in FLUENT does not appear to be zero (and why the temperature is not a constant) on the walls. Even so, the LBM still shows a thicker boundary layer which could be due to the lack of a special treatment for the turbulence intensity near the walls.

The LBM simulation shows important spurious fluctuations near the inlet boundary, that forms a checker-board pattern, due to the presence of high velocity gradient. These instabilities are characteristics of the BGK collision operator for high Reynolds number flow, but they do not appear to be harmful, as they are limited to the vicinity of the inlet and the main flow structures are still recovered.

Figure 62 shows a comparison of the average velocity and temperature profile in a cross section of the room that is aligned with the centreline of the inlet. The FLUENT results are compared against the LBM simulation for various resolutions (i.e. 0.1, 1.7 and 7.2 million nodes).

Clearly, the smallest resolution is not sufficient to resolve the flow accurately while the largest resolution only marginally improves the overall profiles although it does get rid of the spurious oscillations in the velocity profile near the inlet.

Figure 63 compares the mean velocity magnitude and temperature along the chamber height at five different locations inside the chamber (see figure 60), designated as poles 1-5. Table 9 sums up the x and z location of each pole. The velocity profiled from the LBM simulation shows a similar trend to the FLUENT results with some

Pole	x	z
1	1.1 m	1.00 m
2	1.1 m	2.36 m
3	3.1 m	1.00 m
4	3.1 m	2.36 m
5	2.1 m	1.68 m

Table 9: Location of the sampling poles used for comparing velocity and temperature.

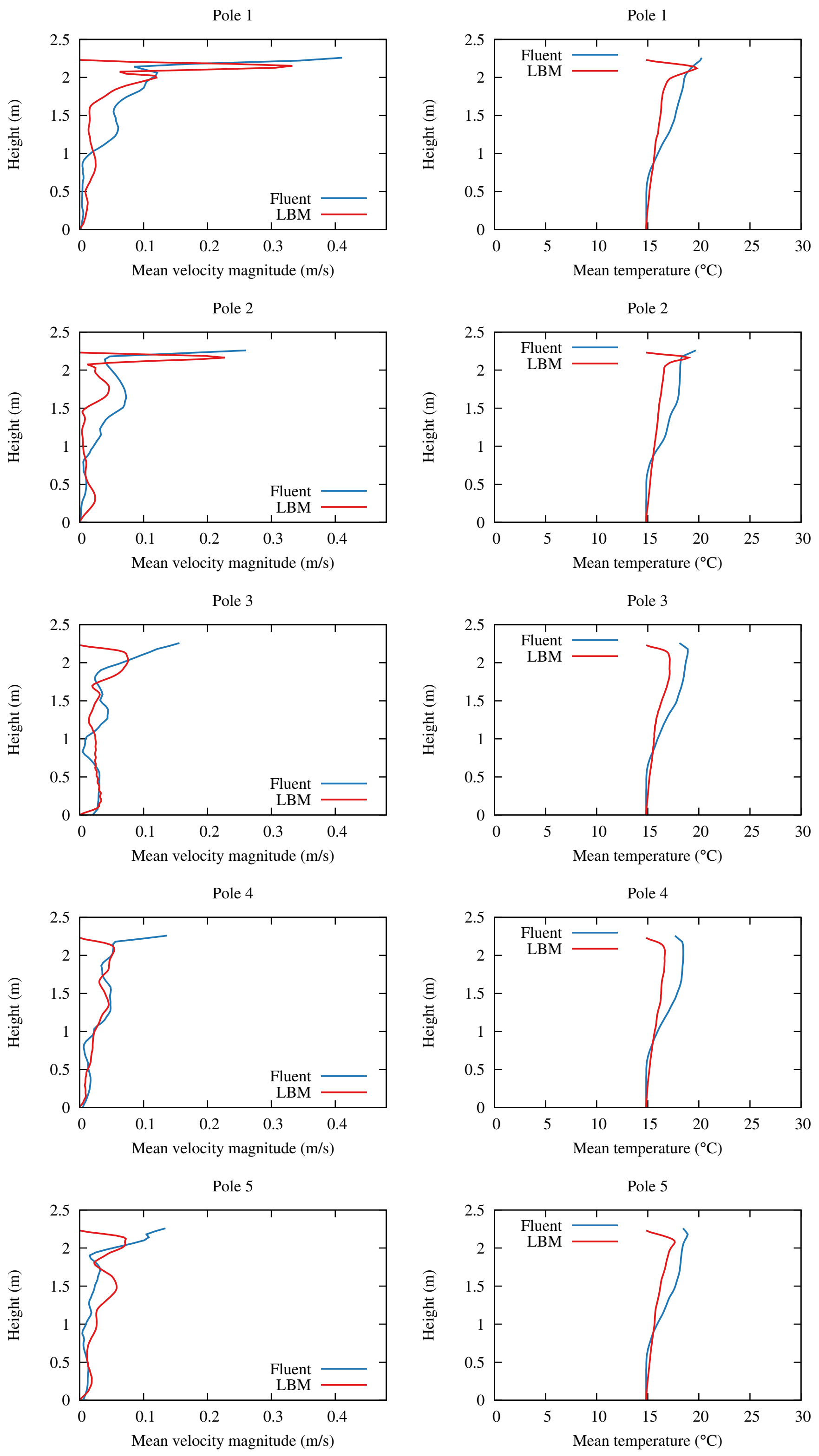


Figure 63: Comparison of mean velocity (left) and temperature (right) between LBM and FLUENT for each of the five sampling poles.

differences in term of the local spatial fluctuations. There are large differences near the top wall at $z = 2.26\,\mathrm{m}$, the velocity and temperature reported by FLUENT appear to be significantly different for that imposed by the boundary conditions. This again shows that FLUENT does not export the actual data at the wall, but rather the value of the first fluid node. LBM does produce the right values at the wall, but shows a thicker boundary layer due to the lack of special treatment for near-wall turbulence. Moreover, it appear that LBM achieves a larger mixing of the temperature, the temperature change between the top and bottom wall is mostly linear, while the FLUENT has a more pronounced drop in temperature around the height $z = 1\,\mathrm{m}$.

Figure 64 shows contours of instantaneous velocity magnitude (from both FLUENT and LBM) on a plane through the centreline of the inlet, during the initial transient phase, captured at various time between 0.5 and 5.5 seconds. The inlet airflow is based on the measured experimental profile discussed previously. During these first seconds of simulations, the LBM shows qualitatively a similar behaviour to the FLUENT simulation. Ultimately, for longer period of time, the instantaneous flow field of the two simulations cannot match exactly because of the chaotic nature of turbulence. The main noticeable difference is in the presence of evanescent pressure waves in the LBM simulation cause by the inlet and due to the artificial compressibility term. These artefacts can be mostly avoided by slowly increasing the inlet velocity over time until it reaches its target value. This was not implemented here to allow for the comparison with FLUENT.

7.2.3 *Conclusion*

While the ANSYS CFD simulation of the ventilated chamber took several days to complete, the optimised LBM running on GPU can simulate the same flow problem in real-time (and actually about three times faster than real-time). Although some discrepancies are observed between the two simulations, likely due to the the differences in the handling of the boundary conditions, the overall flow structure is recovered correctly.

Due to its efficiency, the LBM simulation allows to quickly vary the ventilation parameters (volume flow rate, inlet and outlet locations, etc.) and to instantly visualise the effect on the flow using the integrated visualisation tool (see section 4.5). That tool can extract velocity and temperature (either instantaneous or mean) along a two-dimensional slice free to move within the domain or along a line, such as the five measuring pole. The figure 65 is a screen-shot of the LBM program that shows a representation of the room with a slice of the instantaneous velocity field at the inlet, combined with the profiles of mean velocity at each pole, updated in real-time.

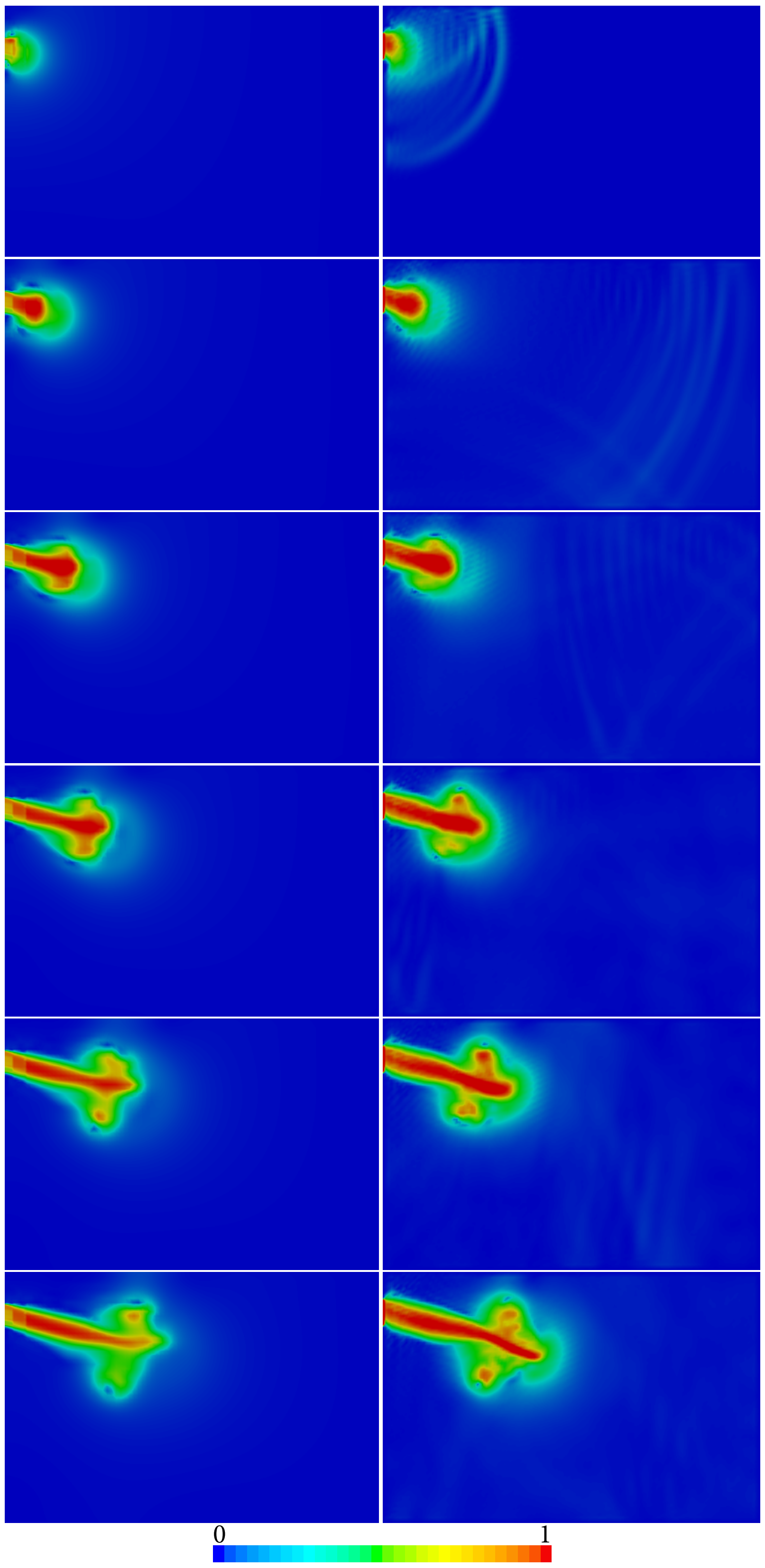


Figure 64: Transient comparison of the isothermal normalised velocity magnitude of the inlet jet of the ventilated chamber. FLUENT results are on the left and LBM on the right. From top to bottom, the simulation time is 0.5, 1.5, 2.5, 3.5, 4.5 and 5.5 seconds respectively.

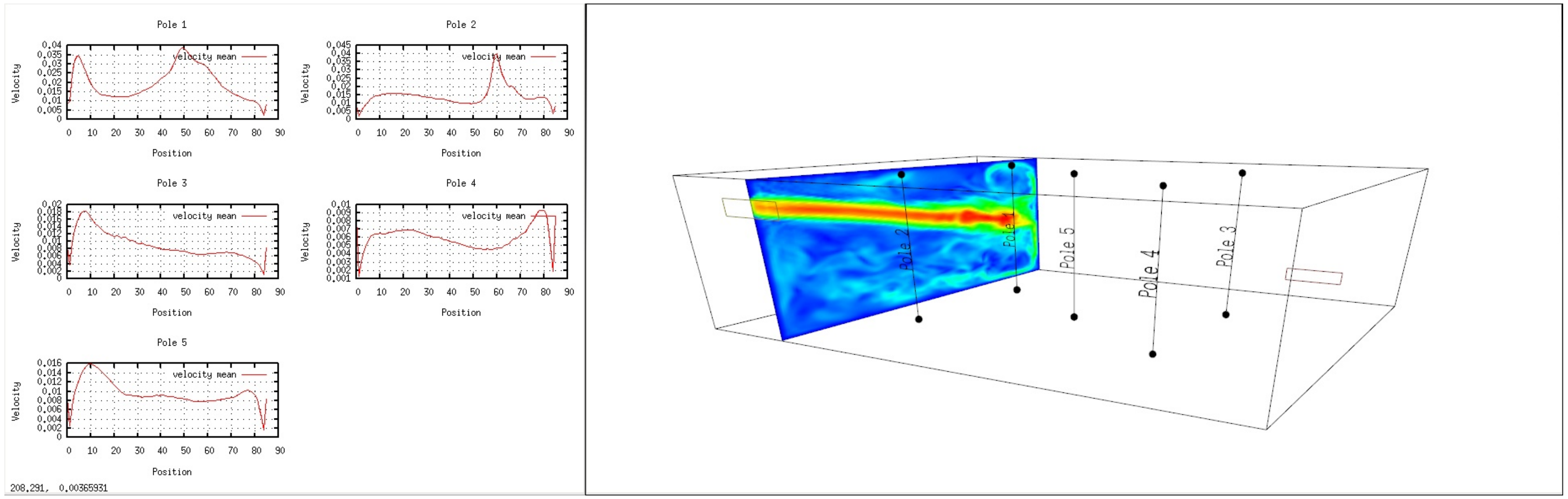


Figure 65: Screen-shot of the real-time LBM software. On the left, mean velocity profiles and on the right a 3D view of the room.

7.3 DATA CENTRE

A data centre is a facility that houses a large number of computer systems, that can be networked together as in a supercomputer. The high density of the computer systems results in a massive generation of heat that needs to be dealt with, usually by transporting the heat to the outside of the data centre using chillers and heat exchangers. Failure to do so might result in some of the servers overheating and having to shut down prematurely to avoid damaging the components, resulting in the interruption of the services being provided.

Data centres are intensive energy users, with the data centre industry accounting for approximately 1.5% of global electricity consumption [293]. Moreover about 45% of that energy is spend on the infrastructure, i.e., cooling, fans and pumps [293]. As a result, there is a desire both within the industry and amongst governments to improve their efficiency in order to reduce costs and environmental impact [294, 295].

CFD modelling can be used to improve cooling effectiveness, reduce power consumption and predict the presence of hot spots for both a new data centre projects or the upgrading of an existing one [296]. CFD has been used successfully to investigate the impact of floor grille and underfloor obstructions [297], optimise the placement of IT equipment [298] and study the effect of aisle containment [299, 300, 301]. The major challenges in producing accurate models are the multiple length-scales [296], from the chip (in millimetres) to the room level (tens of meters), and the various modes of thermal transport and flow regimes [302].

This section aims at simulating a simple data centre model using the developed LBM GPU program and to compare the simulation results against that of the open-source CFD package OpenFoam, the commercial CFD software Comsol and Future Facilities' 6Sigma, designed specifically for data centre modelling (see section 1.2.5 and reference for each software therein).

7.3.1 *Problem Description*

The geometry of the data centre is designed to allow for a simple implementation in each of the numerical method considered while keeping most the characteristics of a typical data centre.

The data centre room has a surface area of $28.8\,\mathrm{m}^2$, it is comprised of ten racks, organised in two rows facing each others as to provide a cold aisle in the centre. Each rack is separated into two servers along the height, so that the load between the top half and bottom half of the rack can differ. This allows to keep the geometry configuration relatively simple, but a more realistic model would divide the rack

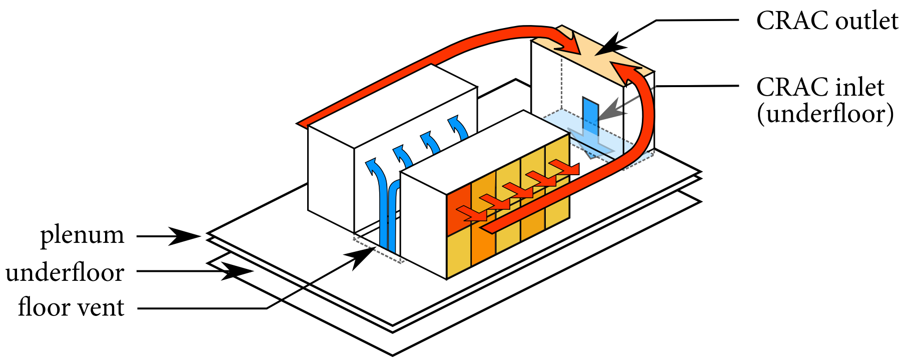


Figure 66: Sketch of the room geometry and air flows.

into each server blade, typically in the forties, depending on the size of the rack and the height of each server (measured in rack units).

The cold air is supplied to the room at a constant temperature $T_{supply} = 16\,°\mathrm{C}$ and a constant flow rate $Q_{supply} = 1.134\,\mathrm{m}^2/\mathrm{s}$ through the bottom of the CRAC[1] unit, which coincides with the bottom of the plenum (i.e. the separation between the underfloor and the room), and is placed placed against the back wall and aligned with the room geometric centre. The cold air travels through the underfloor domain and then enters the room through the floor vents, modelled as open holes and located between the two rows of servers. This is a simplified model, as most a real underfloor might be filled with cables and pipes that would noticeably affect the distribution of velocities arriving at the floor vent. Also, the floor grilles themselves are normally grid-shaped, thus straightening the flow and affecting its momentum, and they can sometimes be oriented to change the angle of the flow. Hence a more accurate modelling need to account for the effect of the floor grille on the flow [303, 304], here neglected to allow for an easier comparison between each software.

The air enters each server through their front-faces and exits from their back-faces after an increase of momentum and a rise in temperature. The volume flow rate through each server Q_{server}, as well as the temperature rise ΔT, are dependant on its load level (see table **10**).

Finally, the hot air is extracted by an outlet located at the top of the CRAC unit and modelled with a Neumann boundary condition, i.e., a constant static pressure p_{return} and zero normal-derivative (also called zero-gradient) for the velocity and temperature, i.e. $\frac{\partial \vec{v}}{\partial z} = \vec{0}$ and $\frac{\partial T}{\partial z} = 0$.

A sketch of the geometry with an approximative flow behaviour is presented on the figure 66 and a plan of the room with detailed measurements in meters can be found in figure 67.

1 Computer Room Air Conditioner

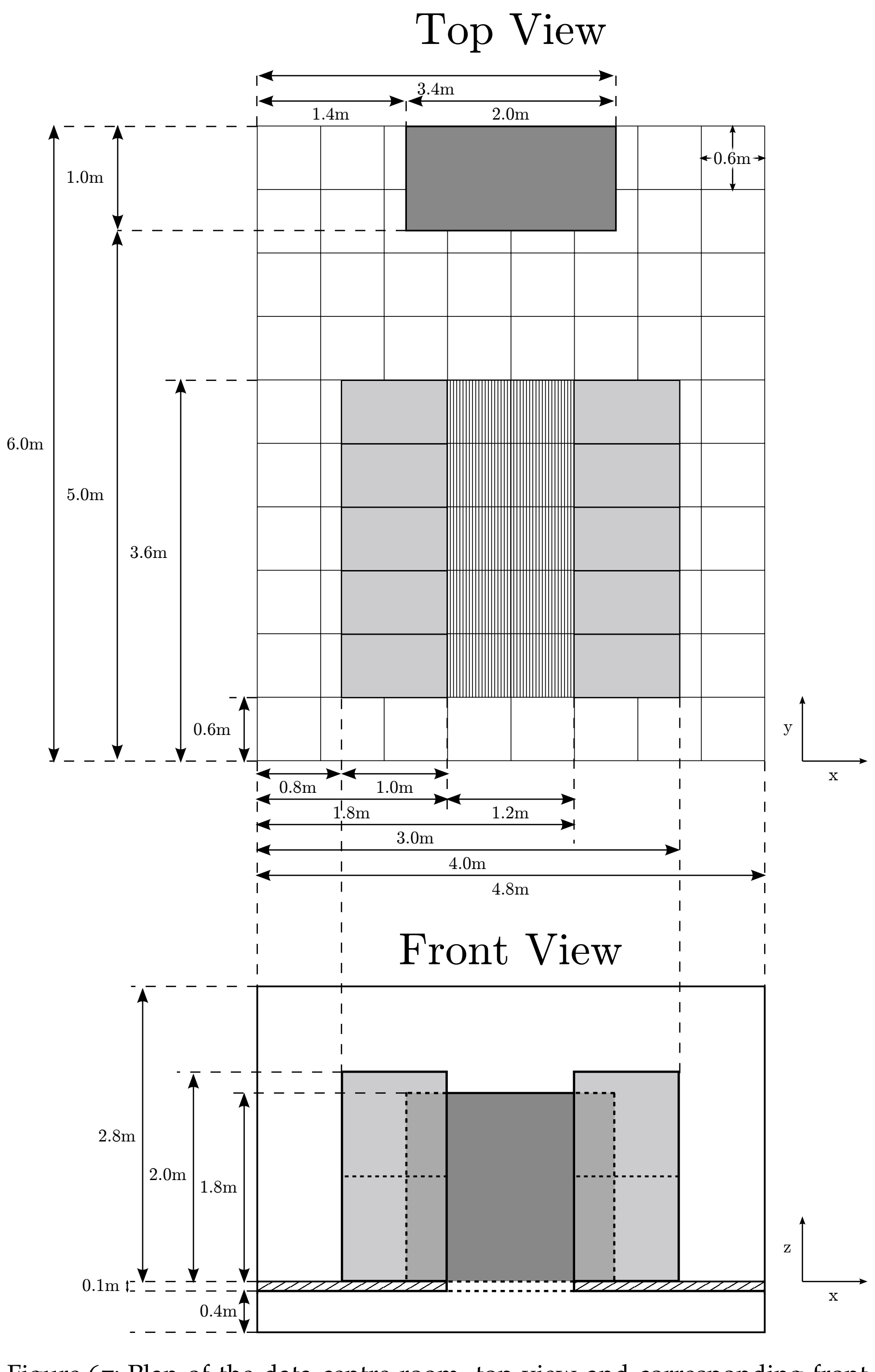


Figure 67: Plan of the data centre room, top view and corresponding front view. The CRAC is represented in dark grey, servers in light gray, plenum in dashed lines and floor vents as fine grids.

The data centre room is considered to be a closed system, thus the domain boundaries are modelled as wall using the so-called no-slip boundary condition, i.e. the velocity of the fluid is null at the wall,

$$\begin{aligned}\vec{v}(x = 0.0\,\mathrm{m}) = \vec{v}(x = 4.8\,\mathrm{m}) = \\ \vec{v}(y = 0.0\,\mathrm{m}) = \vec{v}(y = 6.0\,\mathrm{m}) = \\ \vec{v}(z = -0.5\,\mathrm{m}) = \vec{v}(z = 2.8\,\mathrm{m}) = \vec{0}\end{aligned} \tag{157}$$

The above macroscopic boundary condition is implemented with the half-way bounce-back for the velocity distribution functions (see section 3.2.3). The temperature distribution functions normal to the walls ($\vec{n}$) are also bounced-back, effectively implementing an adiabatic wall boundary condition,

$$\vec{\nabla} T \cdot \vec{n} = 0. \tag{158}$$

The same adiabatic no-slip boundary condition is applied to all the other solid walls in the simulation, i.e. the top and bottom faces of the plenum and the side faces of the racks and CRAC unit.

The servers require slightly more complex boundary conditions, which are dependant on their load, i.e., power usage. The normal-derivative of both the velocity and temperature is set to zero (i.e. $\frac{\partial \vec{v}}{\partial x} = \vec{0}$ and $\frac{\partial T}{\partial x} = 0$) for the front faces of the servers; this condition is sufficient to get an estimate of the rack inlet temperature T_{in}, and it is common practice to use this value for validation of a model or for comparison with an experiment. The temperature of the back face of a server (T_{out}) is set based on the average temperature at the front face (T_{in}) plus a temperature difference (ΔT) that varies with the server load and the velocity $\vec{v}_{\mathrm{out}}$ is set to achieve the specified volume flow Q_{server},

$$T_{\mathrm{out}} = \frac{\int T_{\mathrm{in}} dA}{\int dA} + \Delta T, \tag{159}$$

$$\vec{v}_{\mathrm{out}} = \frac{Q_{\mathrm{server}}}{\int dA} \vec{n}. \tag{160}$$

This boundary condition requires the integration of the temperature over each server inlet face, which is a costly operation on the GPU as it breaks data locality. Instead of integrating, it is possible to use node to node temperature to achieve similar results while avoiding a significant performance hit,

$$T_{(x_{\mathrm{oulet}}, y, z)} = T_{(x_{\mathrm{inlet}}, y, z)} + \Delta T, \tag{161}$$

however, the integral form is chosen in order to conform to the way the boundary is handled in the other CFD software.

Each of the servers can have a different load (either 0 kW, 1 kW, 2 kW or 4 kW) which is related to the temperature change ΔT and the volumetric flow rate Q_{server} by the table 10 . When the server load is *idle*, i.e. 0 kW, it is replaced by a solid wall with no-slip adiabatic boundary conditions. This corresponds to the situation where *idle* servers are blanked using an adiabatic lining. In reality, it is more likely that an *idle* server will be left untouched and thus will allow some of the air to go through.

IT Load (kW)	ΔT (°C)	Q_{server} (m^3/s)
1.0	2.7	0.083
2.0	4.1	0.133
4.0	5.5	0.221

Table 10: Relation between server load, temperature jump and volumetric flow rate.

The temperature jump across a server as reported on the table 10 was obtained using the following formula,

$$\Delta T = \frac{P \cdot \nu}{Q_{\text{server}} \cdot k \cdot Pr} \tag{162}$$

where P is the IT load (in kW), $\nu = 1.511 \times 10^{-5}$ is the kinematic viscosity of air (supposed constant), $k = 2.57 \times 10^{-5}$ is the thermal conductivity and $Pr = 0.713$ is the Prandtl number of the air (ratio of kinematic viscosity to thermal diffusivity).

In order to associate a specific load to each servers, they are given a distinct index, the bottom left row is numbered from 1 to 5, the top left row from 6 to 10, the bottom right row from 11 to 15 and the top right row from 16 to 20, as depicted on the figure 68. Then the IT loads corresponding to each server index are reported on the table 11.

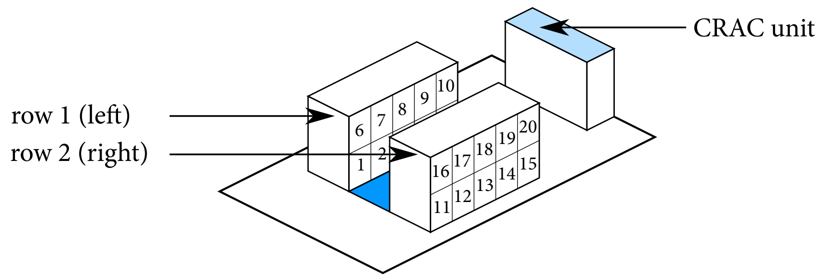


Figure 68: The 20 servers with their associated index (ID).

Left Row				Right Row			
Bottom		Top		Bottom		Top	
1	2.0 kW	6	0.0 kW	11	0.0 kW	16	4.0 kW
2	1.0 kW	7	1.0 kW	12	2.0 kW	17	0.0 kW
3	2.0 kW	8	2.0 kW	13	1.0 kW	18	1.0 kW
4	0.0 kW	9	0.0 kW	14	0.0 kW	19	1.0 kW
5	1.0 kW	10	1.0 kW	15	0.0 kW	20	1.0 kW

Table 11: IT load per server, see figure 68 to match an index against a server location.

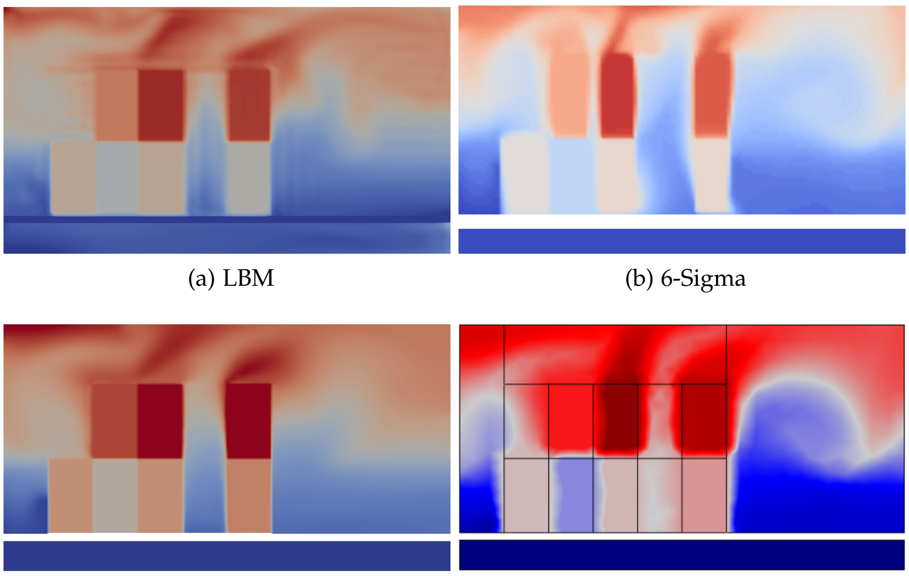

(a) LBM (b) 6-Sigma (c) Open-Foam (d) Comsol

Figure 69: Comparison of the time-averaged temperature field on a plane aligned with $x = 0.8\,\text{m}$, across each software.

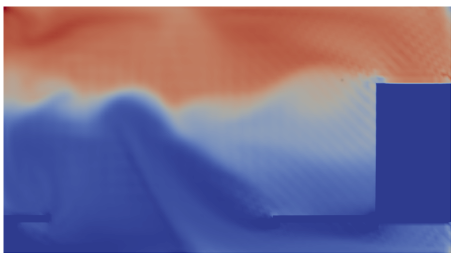
While the LBM solves for the velocity and temperature inside the data centre as they evolve over time (i.e., in a transient-state), the other methods used for the comparison solve directly for the time averaged Navier-Stokes equations (i.e., as a steady-state). To allow the comparison of the results between all the software, the LBM results need to be averaged over time. Hence these are averaged for 3 minutes (of physical time), starting after 2 minutes from an initial state with no velocity and a constant temperature of 20 °C. The effect of the initial temperature will be investigated in the future.

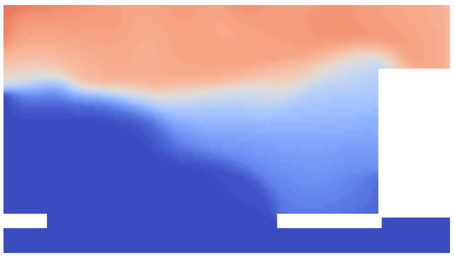
Model	Resolution	Time
LBM	754 K	5 minutes
6-Sigma	864 K	37 minutes
Open Foam	1.3 M	5.5 hours
Comsol	400 K	14.7 hours

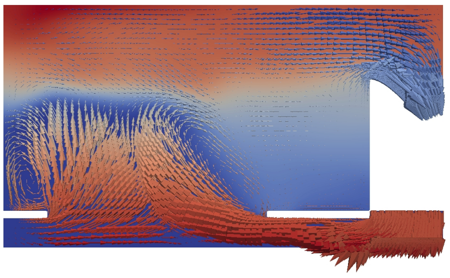
Table 12: Used resolution and computational time across the models.

Another fundamental difference between the LBM and the other models is in the meshing, as this thesis implementation does not support mesh refinement. Hence, in theory, the LBM should require a higher number of nodes to achieve the same level of accuracy as the other models, which are able to refine the mesh near the geometry. The results presented in this section were computed using a regular mesh of $96 \times 119 \times 66$ which took approximately 5 minutes (i.e. real-time) on the GPU. The used resolutions and time to convergence for the other models are summarised in table 12.

As showcased by the figures 69, 70, and 71, the time-averaged temperature fields compares favourably across all the methods : the temperature at the back of the servers shows a similar distribution, the cold air coming off the floor vents rises to the same height (approxi-


(a) LBM


(b) 6-Sigma


(c) Open-Foam

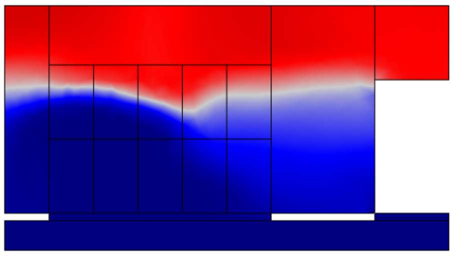
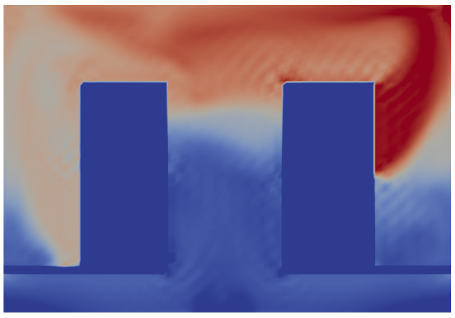
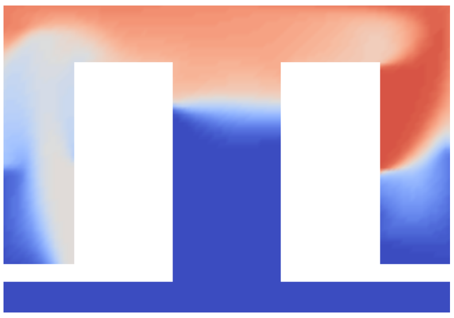
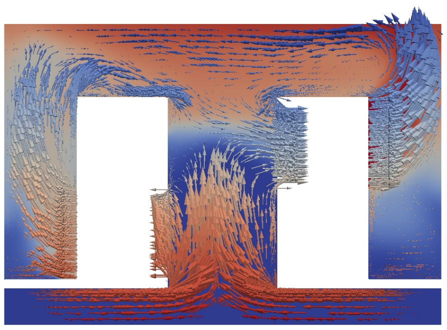

(d) Comsol

Figure 70: Comparison of the time-averaged temperature field on a plane aligned with $x = 2.4\,\text{m}$, across each software.


(a) LBM


(b) 6-Sigma


(c) Open-Foam

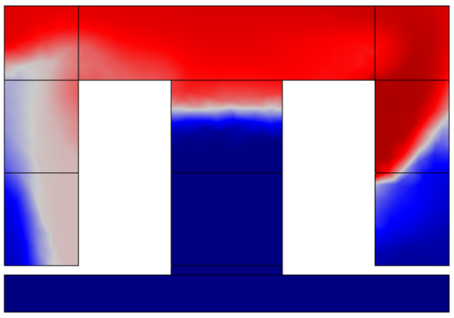

(d) Comsol

Figure 71: Comparison of the time-averaged temperature field on a plane aligned with $y = 0.9\,\text{m}$, across each software.

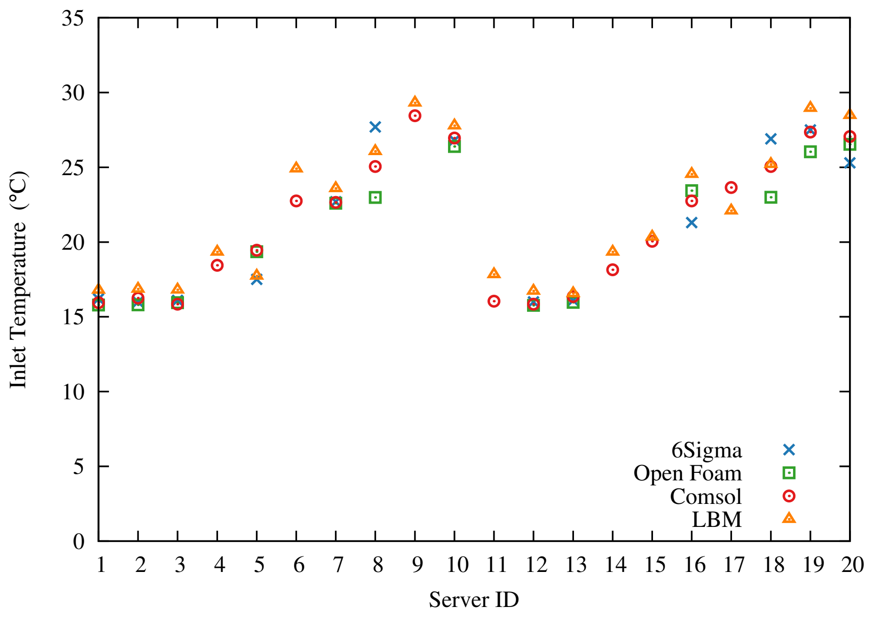


Figure 72: Comparison of the average temperature at the inlet of each server across each software.

mately 1.6 m), the hot air is projected at the back of the servers with the same angle and strength (indicating that the buoyancy is the same in each software). However, the variations in colour schemes used for the rendering of the slices does limit the comparison.

This trend is confirmed by comparing the average temperature at each of the twenty server inlets across the four programs, as demonstrated by the figure 72. For some servers (for instance server 1) all the methods give the same temperature within 1 degree, but for some servers (for instance server 8) there is up to a 5 degrees difference between the hottest and coldest prediction. Overall, the predicted temperature at the server inlets agree reasonably well. The LBM seems to give consistently a slightly higher temperature (by about 1 degree) than the other methods, this might be due to the choice of initial temperature.

7.3.3 *Conclusion*

Overall, the simulation results obtained with all four software are in a good agreement, both in term of the overall flow structures and in the average temperatures at each server inlet. A more detailed comparison, including grid convergence and computation times will be available in an upcoming jointly authored paper, but these preliminary results seems to indicate that the level of accuracy achieved by the LBM is similar to that of the other methods. Moreover, the LBM provides an access to the transient state of the flow in real-time (and faster) and could be used to inform dynamically the data centre management.

However, the simulation results do suffer from some limitations that will have to be tackled beforehand. First of all, the modelling of the floor vents as an *open hole* does not account for the effect of the floor tiles on the direction and momentum of the flow, which would cause significant changes to the flow structure and thermal distribution. Secondly, the LBM display spurious thermal fluctuations that appear as a chequerboard and are due to numerical instabilities. The removal of these fluctuations might be possible through the use of an MRT model (see section 2.1.5) or a flux limiter (see section 2.4.4) or simply as a post processing step. Finally, the effects of the initial state (i.e. initial temperature) and of the choice of start-time (and end-time) for the averaging of the macroscopic fields are not studied in this chapter, but will be the subject of future investigations.

7.4 HOSPITAL ROOM

The simulation of an hospital room (or other indoor air flows) is not significantly different from that of a data centre, as described in the previous section. The problem is still that of an enclosed room, with inlet(s) and outlet(s) for the ventilation and air conditioning, with some solid walls for the geometry (i.e. equipments, beds, etc.) and with some heated surfaces to represent humans. Additionally, an hospital room simulation might include the advection of a scalar field (representing the density of pollutants) or particle tracking to study aerial dispersion and deposition of micro-organisms. Both methodologies are attainable with the LBM.

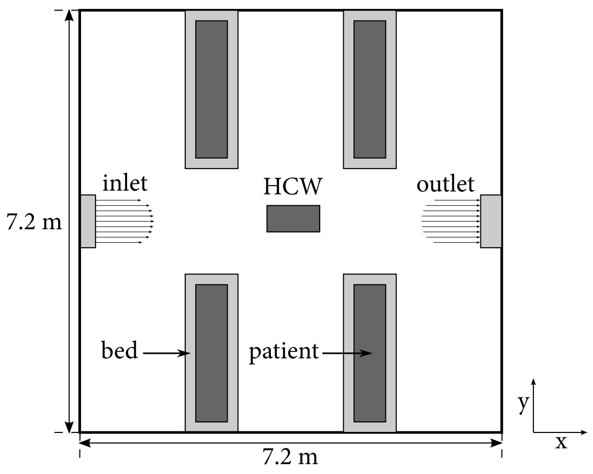


Figure 73: Hospital room layout (top view) with four beds with patients and a health care worker (HCW).

This section considers an hypothetical four-bed hospital room scenario to explore the potential insight that can be derived from a real-time transient model. The room geometry, as depicted on figure 73, has a surface area of 51.84 m^2 and comprises four beds (1.8 m $\times$ 0.6 m $\times$ 0.75 m each) and a patient lying on each bed represented by a cube (1.44 m $\times$ 0.3 m $\times$ 0.2 m) at a constant temperature of 37 °C, similar to human body temperature. In the centre of the room, another heated cube (0.6 m $\times$ 0.3 m $\times$ 1.8 m) has been positioned to represent a health care worker (HCW). Cold air at 15 °C is supplied to the room by a supply inlet mounted on the left hand wall, and extracted by a similar vent on the opposite wall. Both the inlet and extract vents are assumed to be simple rectangular (0.5 m $\times$ 0.2 m) openings without any grille. All the walls, including the floor, the ceiling and the beds have the same adiabatic no-slip boundary condition used for the data centre walls imposed on them.

Figure 74 shows three snapshots of the turbulent temperature field inside the room at different times. These images are post-processed from the real-time simulation results using the open-source 3D creation suite Blender[2] and uses a volume rendering technique to display the temperature as a shaded volume. This post-processing is slow and cannot be done in real-time, and while the real-time volume rendered integrated to the LBM GPU program can achieve similar results, they do not yield the same visual quality.

The process of mapping numerical values (i.e. temperatures) to colours is significantly different with volume rendering, each value not only needs a colour but also an *opacity* (or *emission*) that dictates how much light is absorbed (or emitted) by the voxels. This is how volume rendering can achieve a high degree of realism for smoke and fire animations. In an attempt to replicate the infamous 'rainbow' colour map, used for most scientific visualisation despite its known deficiencies [305], voxels with a small value of temperature (i.e. close to 15 °C) are set to a colour from dark blue to light blue with a high opacity and high temperature voxels (i.e. close to 37 °C) have varying colour from green to yellow to red, with increasing opacity. The opacity for the temperature in between is set to zero, so that only the cold and hot area appear in the final image.

The infamous rainbow colour map.

As shown by the images of the figure 74, the LBM is able to capture the complex interactions between the cold inlet airflow and the thermal plumes of the patients and the HCW. With its real-time capability, the positions of the inlet, outlet and HCW can be interactively moved (either with the keyboard or with a remote control) as the simulation is running to provide quantitative feedback into the effects of different ventilation strategy on thermal comfort and indoor air quality or the effects of a moving HCW on the mixing of the temperature. The program could easily be adapted for other ward designs and for the advection of both a scalar field or particles to study airborne infection risks in a ward.

2 https://www.blender.org/

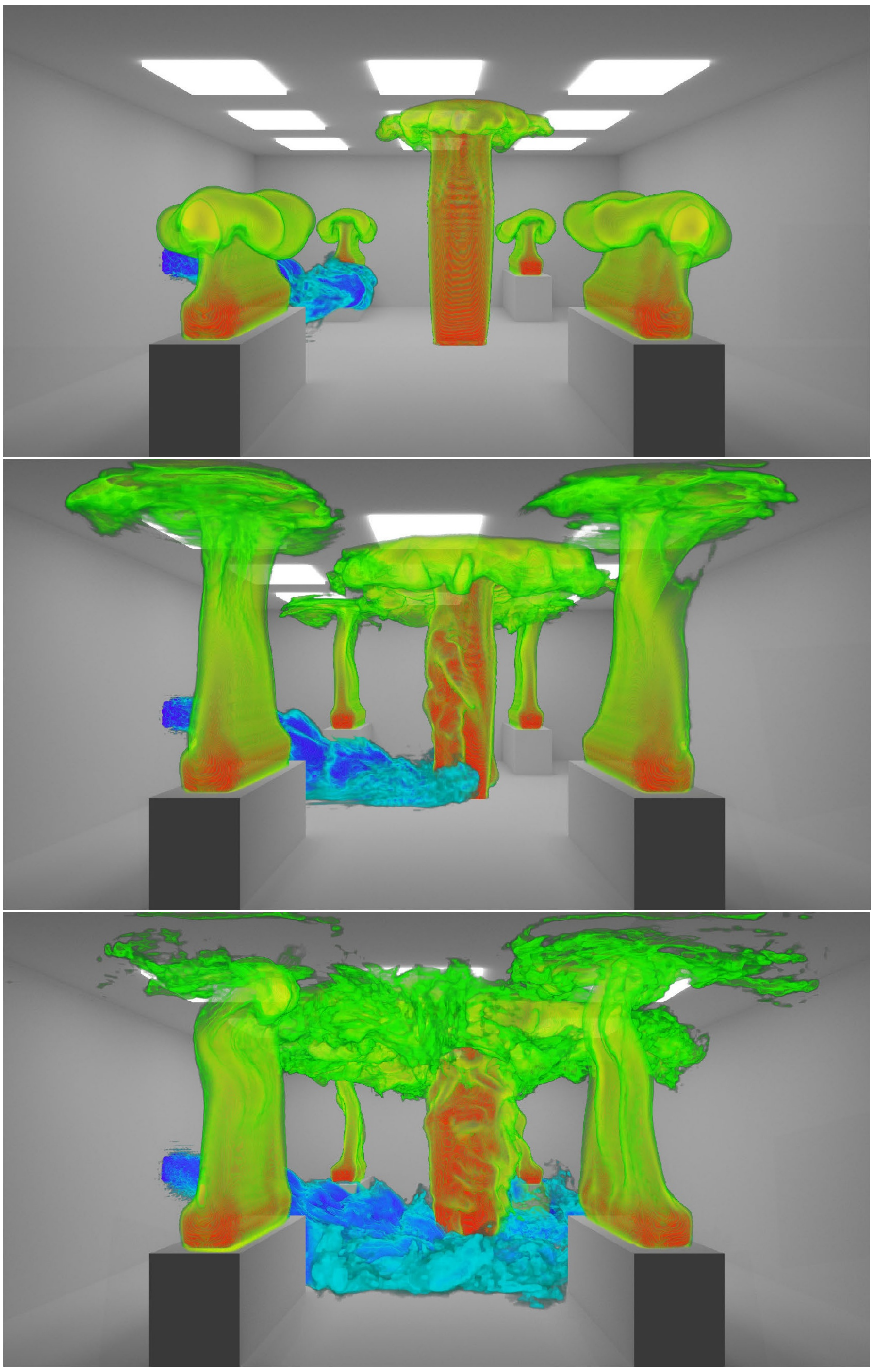


Figure 74: Time ordered snapshots at times 4, 10 and 20 seconds of the evolving turbulent temperature field in the four bed hospital room.

7.5 SUMMARY

As showcased by the three previous applications, the LBM program running on GPU is capable of simulating complex indoor air flows in real-time (or faster) while attaining a degree of accuracy in the simulation results similar to that of conventional CFD software. Moreover, the real-time capability of the LBM program is not restricted to steady-state scenarios and the simulation could potentially be coupled dynamically with sensors and actuators in the real system to provide a novel adaptive management solution for data centres and hospitals.

The current LBM model does suffer from some limitations, such as the lack of near-wall turbulence model or the rise of spurious chequerboard oscillations. Several strategies already exist to tackle these limitations (see sections 2.5.3 and 2.4.4) and their application will be the subject of future investigations.

8 OTHER APPLICATIONS

8.1 INTRODUCTION

While the previous chapter focused on the applicability of the LBM GPU program to indoor air flows problems, the same program can be applied to a larger variety of physical applications, with only some minor modifications to the code. This chapter gives a brief overview of some other applications that are not directly related to indoor air flows, but that were tackled during the thesis through various collaborations. The applications presented here are meant to showcase the large range of capabilities of the program without providing a detailed analysis of each problem. Instead the reader will be referred to the respective works of each collaborators.

While the physical equations and algorithms required for each model were developed in a collaborative effort, the actual implementation on GPU and the simulations presented in this chapter are the work of the author.

In the first part of this chapter, it is shown how the optimised LBM on GPU solver can allow the simulation of complex engineering problems through multi-physics models. The real-time capability was essential in the development of these models. Indeed, once the source code is modified to incorporate a new physical model or a new type of boundary conditions, it only takes a few seconds to update the flow and visualise the results. This improves the iteration time between code development and testing and allows for faster bug tracking and for the exploration of alternative implementations. The last section of this chapter discusses how the massive computational throughput achieved by the GPU can also benefits more theoretical studies by allowing the quick search of a large range of parameters with the goal of finding an optimum.

8.2 MULTIPHASE FLOWS

This section looks at two different multiphase problems : the behaviour of a water droplet falling under an impingement jet and impacting on a flat surface, and the coalesced of water droplets from by a mixture of water-diesel by an engine filter. Although both multiphase flows, these studies were motivated by two different collaborations and use two different models.

The source code for these models was developed based on the optimised single phase LBM code described in the previous chap-

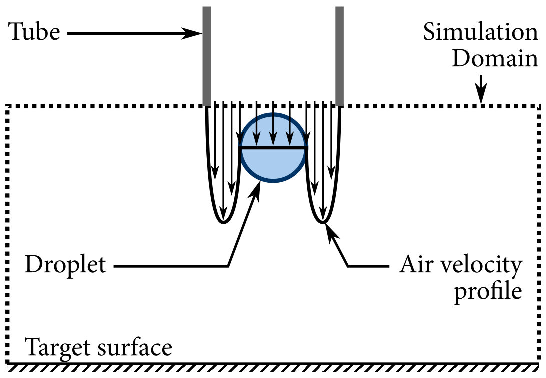


Figure 75: Schematic of the simulation domain for numerical modelling.

ters. Multiphase models introduce non local coupling via the inter-particle force that requires the evaluation of the pseudo-potential at the neighbouring nodes, usually as a function of the macroscopic density. Therefore, the density at each node must be computed first at the beginning of every time-step by a separate kernel and stored into a separate array. This array is then used by the stream-collide kernel for the force evaluation. These additional memory accesses significantly slow down the program compared to a single phase simulation, but is still sufficiently fast to allow for interactive simulations. It should be noted that while interactive, the simulations presented in this section are not computed in real-time as they involve micro-droplets whose time scale is in the order of a micro second.

8.2.1 *Droplet Impingement*

The problem of a falling droplet impacting on a flat plate has been extensively studied in the past, both numerically [306, 307] and experimentally [308, 309, 310], due to its application in spray cooling, liquid spray coating and inkjet printing, however there is still a lack of understanding of the complicated physics at play. Spray cooling is among the most promising methods for cooling high power electronic devices [311] and researchers at the University of Villanova have been investigating the problem [312].

Such systems are difficult to model as the simulation needs to accurately account for the effects of the impingement jet and surface tension forces on the droplet deformations during the impact as well as the heat transfer from the surface to the droplet and its resulting evaporation (i.e. phase change). Moreover, traditional CFD techniques fail to capture the dynamics of the contact angle during the receding phase, hence it was decided to replicate the Villanova's experiment numerically using a multiphase LBM model in a collaborative work with the University of Leeds. This section features some of the results of this collaboration, more details can be found in the thesis of the Ph.D. student from Villanova [313].

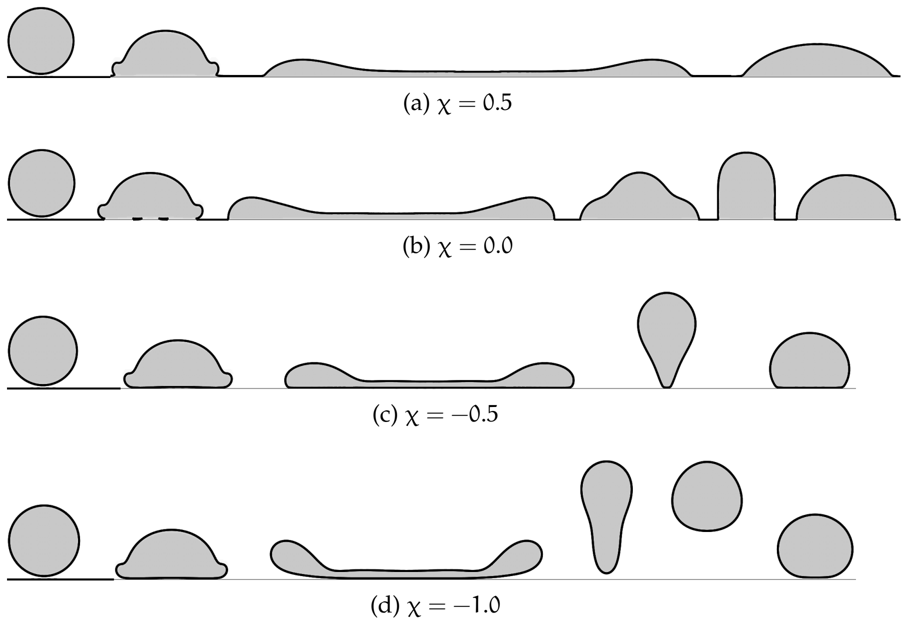
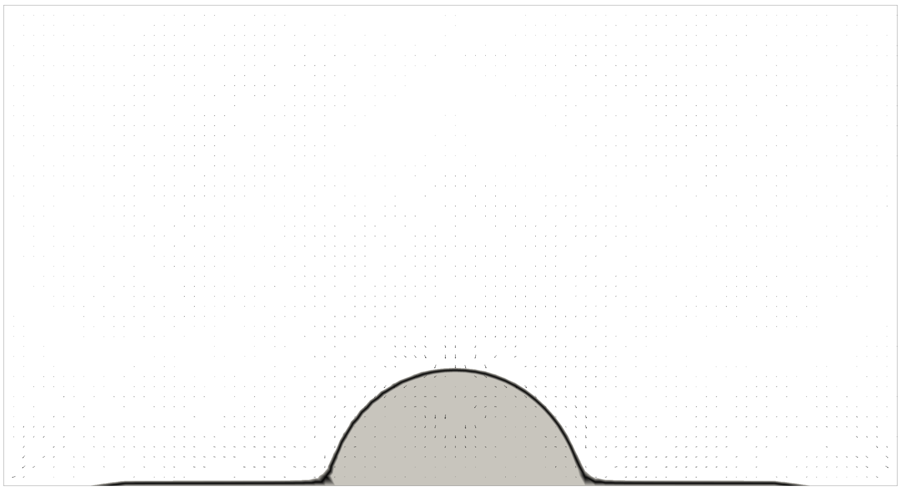


Figure 76: Droplet impact behaviour for different value of the wetting parameter χ.

The chosen multiphase model is that of [87], which uses a Carnahan-Starling equation of state [94] to reach high density ratios and introduces an index function ϕ, to track the different phases and to maintain a sharp interface. That model only provides a starting point, as it describes an isothermal flow. The equations for the model are described in section 2.3.4, they were implemented on the CPU and GPU in both two and three dimensions. The gradients ($\vec{\nabla}$) and Laplacian ($\vec{\nabla}^2$) are computed using a second order finite difference scheme (and up to fourth order by including the diagonal elements) using central-difference in the bulk and forward-difference on the boundaries.

A two-dimensional axisymetric model, with additional source terms such that the emergent dynamics is expressed in a cylindrical coordinates [230, 306, 231], was also implemented but it was discontinued due to the complexity of its implementation.

In the experimental apparatus, a droplet is accelerated by a long vertical tube with compressed air [313], however the simulation domain focuses on the area of impact between the droplet and the (potentially heated) target surface, as described in figure 75. The top boundary condition is an inlet with a constant velocity, a constant pressure is imposed on the the left and right boundaries as to allow the excess air to exit the room, and the bottom boundary is set to bounce-back distribution functions like a wall. The three-dimensional velocity and pressure boundary conditions for this multiphase model are based on the method described in section 3.2.6, but they are challenging because they involve multiple equations. A wetting parameter χ controls the density at the wall, by interpolating between the


(a) No inlet velocity.

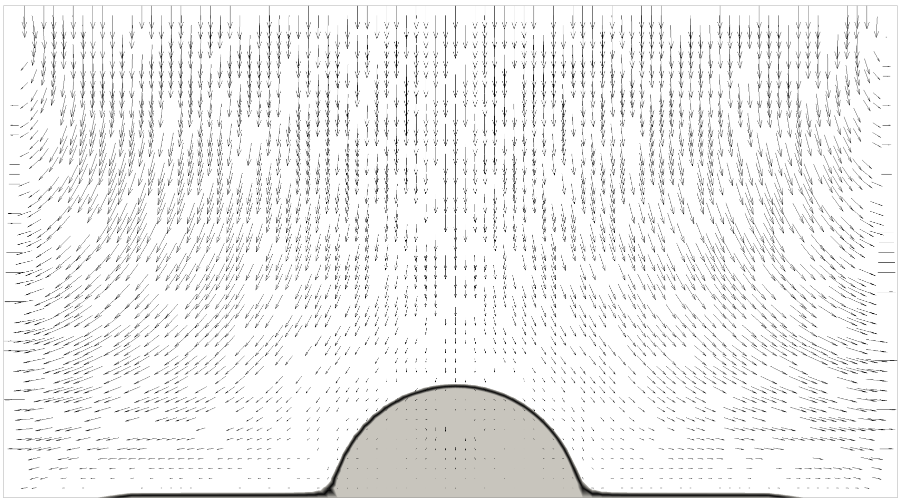

(b) With inlet velocity.

Figure 77: Density and velocity fields of a single droplet with and without an impingement jet.

density of the two phases, and hence controls the contact angle of a droplet at rest on the surface. The dynamic values of the contact angle with gravity and as the droplet spreads and recedes on the surface then naturally emerges from the LBM simulation.

Figure 76 shows different droplet behaviours for four different values of the wetting parameter χ. For positive values, the surface is hydrophilic, i.e. the spreading radius of the droplet is large and the final contact angle is small. For negative values, the surface is hydrophobic and the droplet usually bounces off the surface once (sometimes twice) before settling with a large contact angle.

Figure 77 shows the velocity field for a stationary droplet, with and without the impingement jet, both images use the same velocity scale. For a stationary droplet, it is expected that the velocity field is uniform and nil but in the LBM small spurious velocity currents appear new the interface (figure 77a) due to the large gradients. However, these velocities are small compared to the inlet velocity (figure 77b). The flow created by the inlet achieves qualitatively the expected behaviour, with a high uniform flow coming from the top boundary, exiting symmetrically on each slide and decreasing in magnitude near the bottom surface. This model will allow to study the effect of the impinging jet on the spreading and receding dynamic of the droplet, and ultimately on its evaporation and the cooling of the surface. More details can be found in [313].

8.2.2 *Water in Diesel Filtration*

This study is motivated by the filtration of water droplets in diesel fuel engines, it is part of a collaborative work with another fellow Ph.D. student from the same department and more details on the validation of the model and analysis of the results as well as more references can be found in his thesis [314].

Topically, a diesel tank contains impurities, such as water droplets, that can seriously degrade the engine performance and they need to be extracted by a *filter*, before the fuel reaches the engine [315]. A filter is made out of multiple layers of a non-woven fabric on the principle that its fibres will mechanically force the micro-droplets to *coalesce* into larger ones which can then be more easily separated and collected.

The goal is to simulate the behaviour of the droplets as they move through the fibres of the filter and coalesce into bigger droplets. Once calibrated, such a simulation could be used to optimise the filter design and get a better control on the size of the droplets at the exit. The mixture of water and oil is represented by a multiphase multicomponent model as presented in [88, 85]. In this model, the water and oil are simulated by two different set of distribution functions that are coupled through the inter-particle force; this is essential as the two fluids components have different physical properties. The equations of the model were presented in section 2.3.4.

Although the Shan-Chen model [85] is limited to small density ratio, it is still a choice for this system as the density ratio between the water and the diesel is around 1.2, based on the the European diesel fuel specification EN 590 [316]. The model was implemented in two dimensions, which allows for the exploration of large simulation domains, but a three-dimensional implementation is planned as well.

In the simulation showcased here, the flow is put in motion by an inflow boundary condition on the left that imposes a constant velocity and density at the inlet and by an outflow boundary condition on the right which assumes that the macroscopic quantities at the outlet have a zero gradient. The domain initially contains only oil, but several

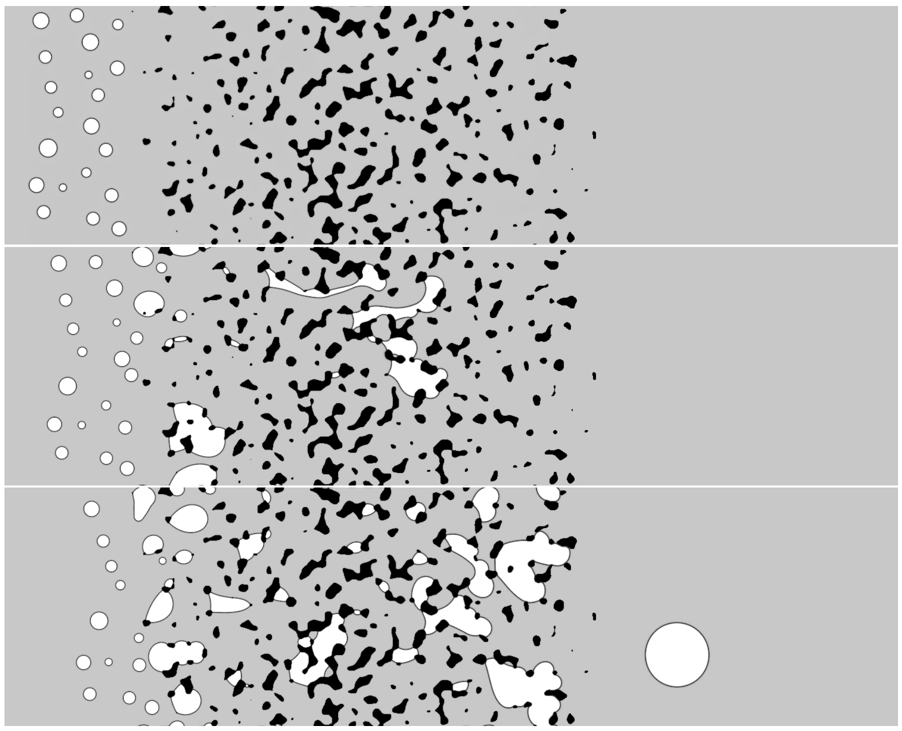


Figure 78: Snapshots of density profiles showing water droplet coalescence within the filter. The water phase is represented in white, the diesel phase in grey and the solid fibres in black.
The full video is available at https://youtu.be/AgLqufg-hys.

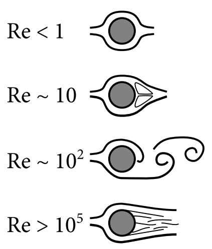
water droplets with small radii are regularly added near the inlet to approximate a water-diesel mixture on the left of the filter.

The filter fibres can take various geometries composed of rectangles and circles, or constructed using a Perlin noise [317], as in figure 78. The Perlin noise algorithm is commonly used in computer graphics to create a smooth homogeneous isotropic noise. Here, a threshold is applied on that noise to define the solid nodes (in black on the figure), by varying this threshold the porosity can easily be adjusted. Moreover, it can be varied in space so that the porosity is lower in the centre, i.e. there are larger fibres in the centre. A simple bounce-back boundary condition is applied on those solid nodes and their wettability is fixed to specify a contact angle at rest ($90°$ for the figure).

The figure 78 shows the evolution of the system, as more and more small water droplets are introduced, they are forced to coalesce on the filter, creating large clusters of water which are then driven by the flow to form large water droplets at the exit of the filter on the right hand side. These preliminary results are promising, as they qualitatively agree with the filter design, but a more detailed analysis for a larger range of configurations will be available in [314].

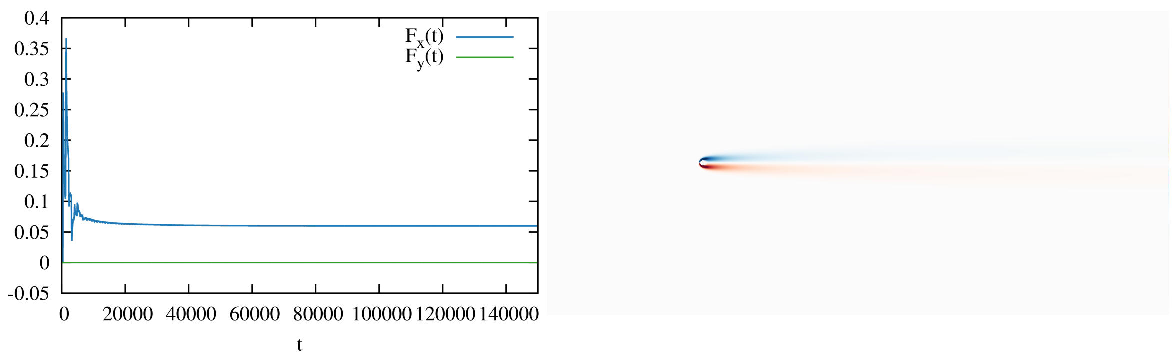
In the problems of fluid-structure interaction (see next section), it is important to accurately measure the force exerted by the fluid on an object. Fortunately, this force can be evaluated very efficiently in the LBM by summing the contributions of the distribution functions coming in and out of the object interface, as described in section 2.3.6. This section briefly presents some of the results on the forces (drag and lift) exerted by a constant moving flow on a stationary two-dimensional sphere, i.e. a cylinder, in the steady and transient regimes and compares the evolution of the drag coefficient with the Reynolds number against the literature.

Keeping the cylinder stationary allows to avoid the difficulties of updating the location of the boundary nodes and, by Galilean invariance, it should be equivalent to a cylinder moving at a constant speed in a fluid at rest [212]. Hence, the cylinder is implemented using the bounce-back scheme (see section 3.2.3, half-way or full-way did not yield a measurable difference on the force) while the inlet and out boundary conditions respectively on the left and right boundaries are achieve with a simple force equilibrium scheme (see section 3.2.2). The radius of the cylinder is set to $r = 10$ nodes, as a minimum to accurately resolve its shape using a staircase approximation on the square lattice. Because the top and bottom boundaries are set as periodic, the system geometry is equivalent to an infinite array of cylinders, but the domain size is increased until the forces on the cylinder do not change, i.e. until the effect of the boundaries on the resulting forces is negligible. It was found that a significantly larger domain is required, the results in this section use a 2048×1001 lattice size. The location of the cylinder within the domain does not affect the drag force significantly, only the time required to reach the final state, hence the cylinder is placed at a quarter of the width and half of the height.

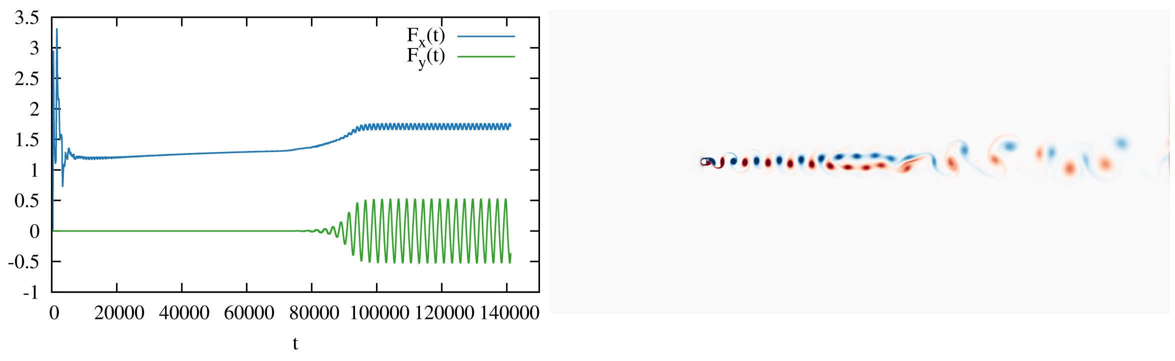


Qualitative description of the fluid flow over a cylinder for a large range of Reynolds number.

As can be seen on figure 79, the behaviour of the flow around the cylinder is strongly dependent on the Reynolds number. For small values, the flow reaches a steady state solution, for intermediary values it displays an oscillatory behaviour and for large values it becomes turbulent. As a result, the drag force (i.e. the x-component of the force exerted by the fluid on the cylinder) converges towards a single value for small Reynolds number but oscillates for larger ones, and it is then required to average the force over time to estimate the drag coefficient. Also, the shredding of the wakes behind the cylinder creates a oscillatory lift force (i.e. the y-component of the force). These two behaviour can observed on figure 79 for two different Reynolds number (Re) of 36 and 360. The Reynolds number is adjusted by keeping the velocity to $u_0 = 0.1$ lattice unit and computing the correct relation time as $\tau = 1/2 + 6u_0 r/\mathrm{Re}$. Very small



(a) $\mathrm{Re} = 36$



(b) $\mathrm{Re} = 360$

Figure 79: Drag force $F_x(t)$ and lift force $F_y(t)$ on a 2D cylinder over time.

Reynolds number (less than 1) would require very large relaxation time and result in inaccurate simulations, hence the inlet velocity is reduced to $u_0 = 0.01$ for this range.

The drag coefficient C_D is a dimensionless quantity that qualifies the amount of resistance induced by the flow on the object, it is typically influenced by the shape of the object and by which direction it is facing the wind. It is a useful quantity for aerodynamics, for instance the shape of an aerofoil is designed to maximise the lift (upward force) and minimise the drag. The drag coefficient is typically defined as

$$C_D = \frac{2F_D}{\rho \vec{u}^2 A}, \tag{163}$$

where F_D is the drag force, i.e. the force in the direction of the flow velocity, ρ is the density of the fluid, $\vec{u}$ is the velocity of the object relative to the fluid and A is the reference area (usually the projected area facing the wind).

The measurement, both experimentally and numerically, of the drag coefficient for a three-dimensional sphere or a two-dimensional cylinder has been the subject of many investigations [318, 319, 320,

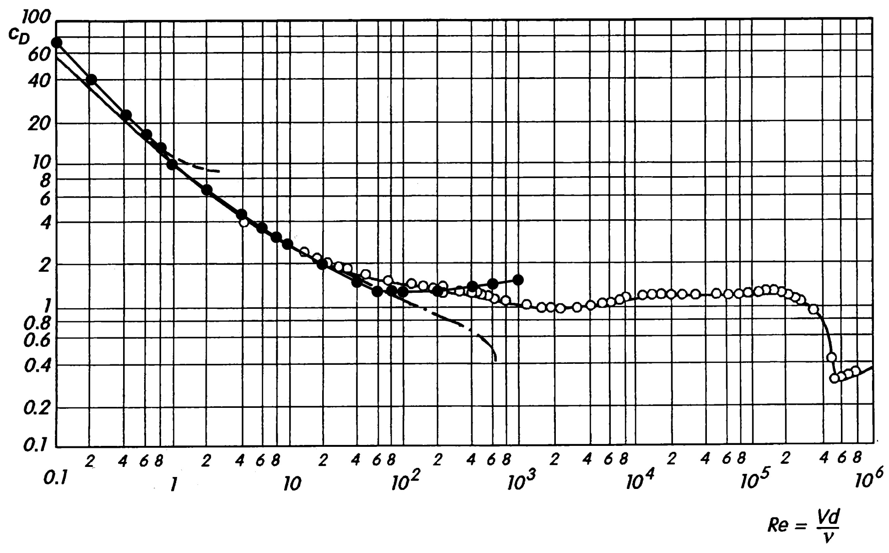


Figure 80: Drag coefficient vs. Reynolds number for a circular cylinder:

●	LBM,
○	measurements by [319] as reproduced in [322],
— — —	asymptotic formula for Re → 0,
— · —·	numerical results by [320] and [321] for steady flows.

321]. For the purpose of this work, a 2D cylinder is considered and C_D can be directly expressed in lattice units

$$C_D = \frac{F_x}{u_0^2 r}. \tag{164}$$

As the force $\vec{F}$ is proportional to the number of boundary nodes in the equation (79) and the number of boundary nodes increases linearly with the radius r of the cylinder, the drag coefficient should be independent of r.

The drag coefficient is measured for a large range of Reynolds number (from 0.1 to 1000) on top of the curve published in [322] and reproduced in figure 80. It can be seen that the results agree very well in the range from 1 to 100, but the LBM tends to overpredict the drag for both small and large values of Reynolds number. This could be due the extremely large (or extremely small) values of relaxation times required in those range. The study of larger Reynolds numbers with LBM would require to either consider larger radius and domain size or an improved collision operator with a turbulence model. It would be interesting to see the drag coefficient curve obtained by keeping the relaxation time fixed and varying only the fluid velocity.

First Picture of Narrows Bridge Falling
Seattle Post-Intelligencer
Going Down

The infamous Tokoma Narrows Bridge collapsed in 1940 after violent torsions caused by winds [323].

In many engineering studies, the dynamic behaviour is the result of one or several movable objects interacting with the surrounding fluid flow, examples include automobile aerodynamics[ref], wind turbine[ref], mixing[ref], etc. In some cases, the objects can be deformable and the interactions of the fluid with the structure can result in oscillatory deformations that can become catastrophic if not kept in check.

This sections looks at some fluid-solid interactions, i.e. the object is a non deformable solid, using the LBM as discussed in section 2.3.6. It is also possible to consider the case of deformable structures with the LBM, as in [324], but this requires an additional solver for the object deformations under the forces exerted by the fluid. Hence, this work only considers one-way coupling (from the solid to the fluid).

Nevertheless, it is also possible to consider two-way coupling on a solid object by computing the force exerted on the boundary of the object (as in the previous section) and considering the force in the momentum equation for that object [119]. And another research group based in the *Institute of Particle Science and Engineering*, at the *University of Leeds*, is developing such a two-coupling solver using the LBM for the fluid component and the *Discrete Element Method* (DEM) for the solid component. They are interested in the study of the settling of objects under gravity for packing and fluidised beds applications [325, 326]. The work presented in this section (and in the previous one) was motivated by some collaborative work with that group and utilises the immersed boundary method described in section 3.2.8.

With this method, the object is represented by an array of voxels, each voxel has an associated cell coverage ratio $B_n(x, y, z)$, describing the amount of solid inside that voxel (a value between 0 and 1 allows to smooth the boundary to avoid staircase edges) and velocity $\vec{u}(x, y, z)$, describing the velocity of the object at that voxel. They are updated at each time-step by a dedicated GPU kernel. This allows for a simple description of a large variety of object shapes and movements that tallies the voxel representation used in DEM. Using the GPU to accelerate the LBM computations allowed to increase the performance of the application by a factor of 10 to 20.

In order to illustrate some examples of applications, the the stirring of the flow a two- or three-dimensional blade is considered, and several snapshots of the instantaneous vorticity field are reported in figure 81 and 82. Both systems can be simulated in real time. For the 2D simulation, the cascaded LBM (see section 2.4.2) is used to allow very small relaxation time, and the simple BGK model is used in 3D. While in 2D, the vorticity is a scalar field and can be represented directly with a colour coded rotational direction, on the other hand in 3D the vorticity is a vector field, hence only its magnitude is

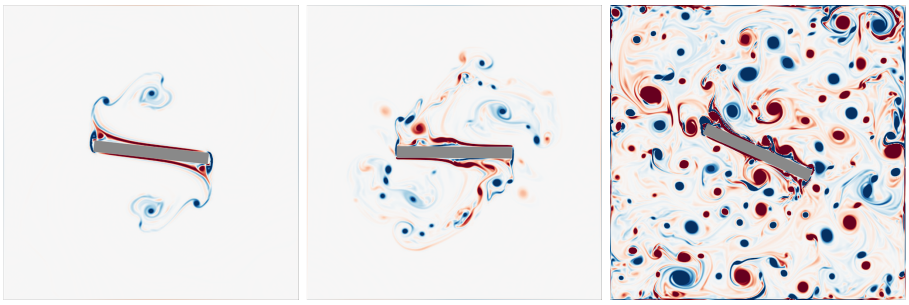

(a) Close to the initial state. (b) Intermediate state. (c) After a long time.

Figure 81: Snapshots of the vorticity field created by a two dimensional rotating blade using the immersed boundary method. Positive vorticity is represented in blue and negative vorticity in red.
The full video is available at https://youtu.be/bSsT4QrQqAA.

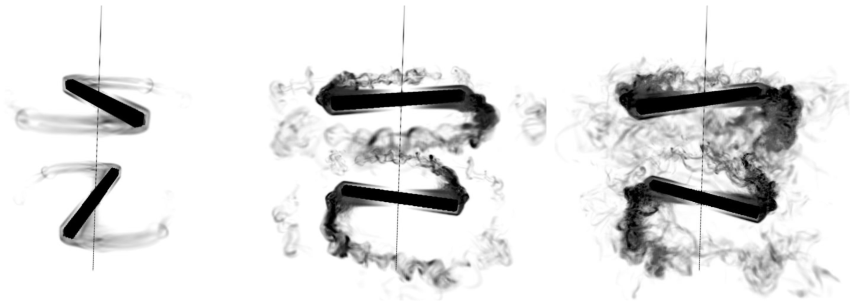

(a) Close to the initial state. (b) Intermediate state. (c) After a long time.

Figure 82: Snapshots of the magnitude of the vorticity field created by two counter rotating three dimensional blade. The axis of rotation is represented by a vertical line.
The full video is available at https://youtu.be/oobdA2XncnE.

represented using real-time volume rendering. Finally, the blades are coloured with a dark grey for both simulations.

It was found that the IBM model removes all the pressure waves typically seen as the object rotate without a proper interpolation of the boundary, as long as the rotational velocity does not change suddenly. In order to avoid pressure waves near the beginning of the simulation, the rotation of the blades is slowly increased over time, until its final velocity is reached. The resulting flows seem to be qualitatively accurate, but a detailed analysis would be required, more results on LBM-DEM can be found in [325].

It is well known that the LBM can become unstable for high Reynolds number flows and many techniques aim to increase its stability. The MRT method is one of such techniques. It introduces additional relaxation times that can be adjusted to suppress non-hydrodynamic modes that do not appear directly in the continuum equations, but may contribute to the instabilities. However, finding a good choice for these free relaxation times can be a difficult task which is why the MRT is often discarded in favour of simpler methods such as TRT and BGK. On the other hand, the cascaded LBM (i.e., central moment MRT) promises to increase the stability domain of the MRT and to effectively remove the need to search for an optimum.

The massive computational power offered by the GPU allows to explore a large range of relaxation times and to accurately determine the stability domain of both MRT and cascaded LBM. Moreover, the fast LBM simulations on the GPU can be combined with some optimisation algorithms to quickly estimate the set of optimal relaxation rates for a specific problem.

While it is expected that the techniques and results presented have the potential to be applied to a wide range of problems, this section focuses on the benchmark problem of a perturbed shear layer in a doubly-periodic domain, as originally studied in [327]. This setting is often used to test the stability of fluid solvers and that of the MRT LBM in particular [15]. It has multiple advantages: it is simple to set up, does not require a special treatment of the boundaries (periodic), and it develops thin layers that are hard to solve on a coarse mesh.

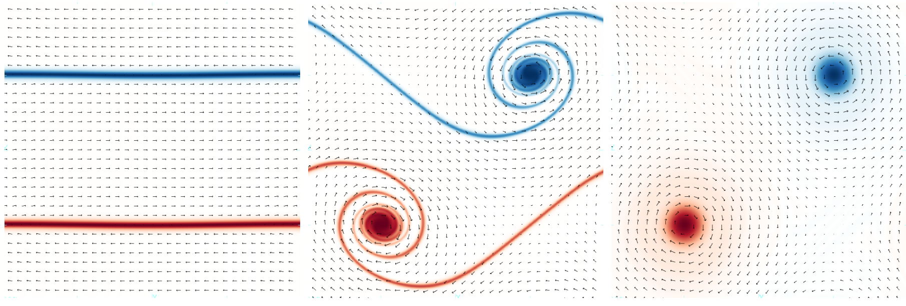

(a) $t = 0$ (b) $t = 1$ (c) $t = 10$

Figure 83: Snapshot of the velocity field (black arrows) and vorticity field (red for positive, blue for negative) over time.

The initial velocity field is given by

$$u_x = \begin{cases} U_0 \cdot \tanh\left(\kappa\left(y - 1/4\right)\right) & y \leqslant 1/2 \\ U_0 \cdot \tanh\left(\kappa\left(3/4 - y\right)\right) & y > 1/2 \end{cases} \tag{165}$$
$$u_y = \delta \cdot U_0 \cdot \sin\left(2\pi\left(x + 1/4\right)\right) + u_s$$

where U_0 is the initial bulk velocity of the flow, κ controls the width of the shear layers, δ is the magnitude of the initial perturbations and u_s is an optional shift velocity, to uniformly move the flow along the y-direction and study the Galilean invariance of the results. All the simulations were performed with $U_0 = 0.04$ (i.e. Mach number $Ma \simeq 0.07$) and parameters $\kappa = 80$ and $\delta = 0.05$ as in [15], and the main relaxation rate (τ associated with the shear viscosity) is adjusted so that the Reynolds number is $Re = U_0 \cdot N/\nu = 30\,000$ with $\tau = 1/2 + 3 \cdot \nu$. The resulting flow and its evolution is illustrated on the figure 83 by taking snapshots at three different times showing the vorticity magnitude and the direction of the velocity field.

It is well known that the BGK has difficulties simulating this problem at small resolutions, as the shear layer becomes too thin for the grid size [328, 15], at a resolution of 256^2 there is a formation of spurious currents and at a resolution of 128^2 the simulation becomes unstable very quickly. The resulting vorticity fields are displayed on figure 84.

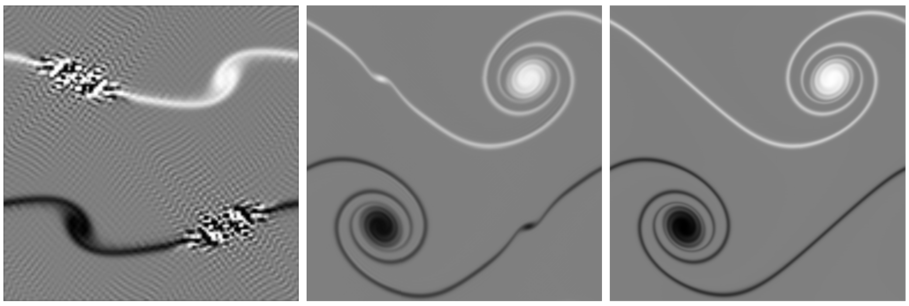

(a) $N = 128$, unstable (b) $N = 256$, spurious eddies (c) $N = 512$, all fine

Figure 84: Snapshot of the vorticity field (black for positive, white for negative) at time $t = 1$ for different resolutions with the BGK collision operator.

The goal of this study is to find the values of the free relaxation times that gives the best stability and accuracy for the coarse resolution of 128^2 where the BGK is unstable. Three models are investigated in two dimensions (i.e. for a D2Q9 node) : (1) the TRT model (see section 2.1.5) with only one free parameter, (2) the MRT model (see section 2.1.5) and (3) the cascaded model (see section 2.4.2) with three free relaxation times, w_b, w_3 and w_4 respectively associated with the bulk viscosity, the energy square and the energy flux (following the model of [143]). The MRT model is built from the cascaded model but without the transformation to central moment space, hence it provides the same three free relaxation times.

The TRT model has the advantage that it has only one free parameter, as one of the relaxation time is fixed by the Reynolds number $\lambda_e = 1/\tau = 1/\left(3\frac{U_0 \cdot N}{Re} + \frac{1}{2}\right)$. The other relaxation time λ_o is free to

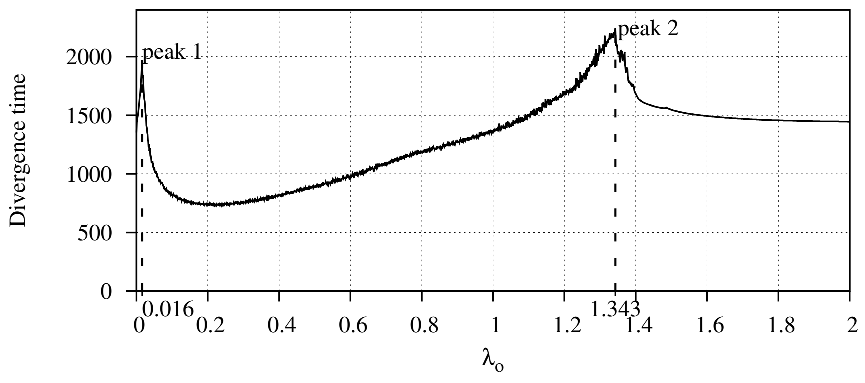


Figure 85: Variation of the divergence time with the free relaxation rate λ_o for the TRT model of [53].

vary in the interval $[0;2]$. To investigate the effect of this parameter on the stability of the simulation, the program is run several times with identical starting conditions and λ_o is varied by increments of 0.001. The program is exited as soon as instabilities appear, and the time at witch these instabilities start (called *divergence time*) is recorded. Instabilities are detected by monitoring the density over the entire domain, and the simulation is considered to be unstable if the density becomes negative, i.e., $\rho \leqslant 0$. It is found that although varying λ_o allows to vary the divergence time and to change the origin of the instabilities, it is never sufficient to make the simulation stable. The figure 85 shows that the divergence varies between 1000 and 2000 time-steps and presents two peaks for $\lambda_o = 0.016$ and $\lambda_0 = 1.343$. Hence, it can be concluded that the TRT model is not capable of simulating the double shear layer problem for on a coarse grid.

The MRT and cascaded LBM offer two additional relaxation times compared to TRT, hence the parameter space for these models is three dimensional. And these relaxation times can be tweaked to obtain a stable simulation. One possible to way to explore this 3D space is to fix one of the relaxation time to 1 (i.e. the middle value) and to vary the other two relaxation times to build a slice. The accuracy of the slice depends on the number of simulation along each direction. The figure 86 is constructed using 100^2 regularly spaced simulations, each simulation is allowed to evolve until instabilities appear or until a total time $t = 100\,000$ is reached (the simulation is then assumed to be stable). The divergence-time is recorded and each simulation is represented as a coloured point based on specified scale (see figure 86). The same technique is applied to the cascaded LBM, and the results are displayed on figure 87.

Clearly, the choice of relaxation times is of prime importance on the stability of the results. The main difference between MRT and cascaded seems to be in the stability range of w_3: in the MRT model most of the domain $w_3 < 1$ is highly unstable while in the cascaded

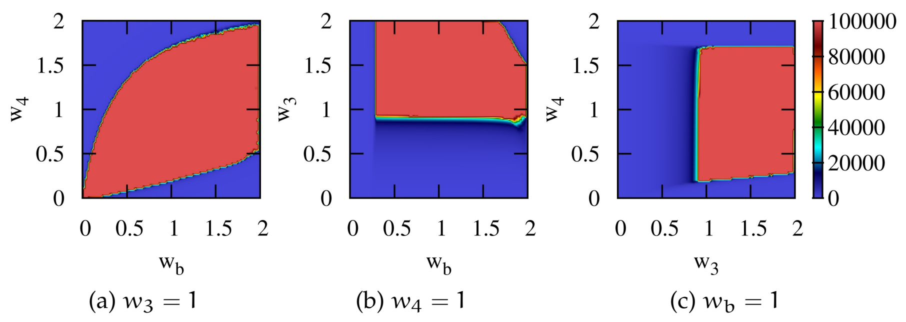


Figure 86: Divergence time for the MRT model on three slices at constant w_3, w_4 or w_b.

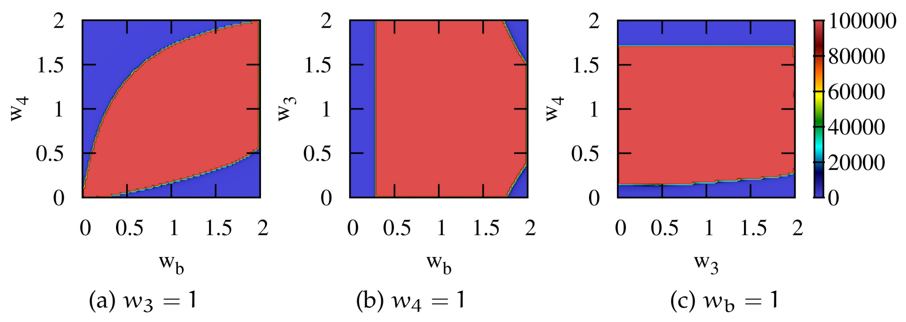


Figure 87: Divergence time for the cascaded model on three slices at constant w_3, w_4 or w_b.

method the whole range of w_3 is available and the stability is mostly controlled by w_b and w_4. The interface between the stable and unstable domains is particularly sharp which seems to indicate that the envelope of the stability domain might be defined by an analytical equation involving the relaxation times. In particular the slice $w_3 = 1$ shows that the stability area is contained within two *circular arcs*. This behaviour is made more evident when taking a slice at $w_3 = 2$, as shown in figure 88, constructed with 1000^2 simulations, where all of the simulations outside of this domain diverge in less than 6 000 time steps.

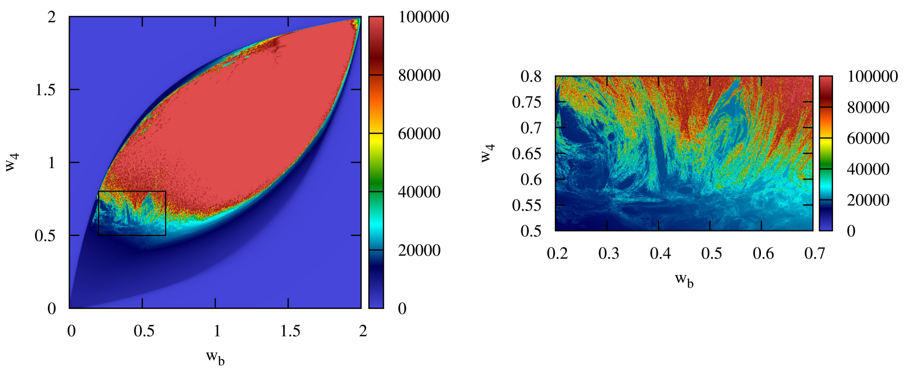


Figure 89: Divergence time on the slice $w_3 = 2$, for the MRT model, up to 100 000 time steps.

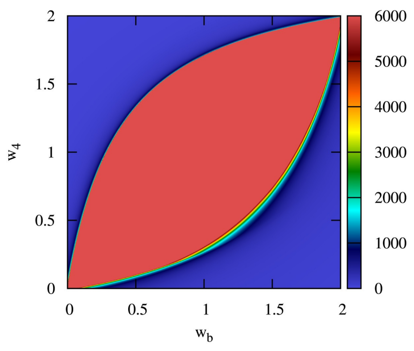
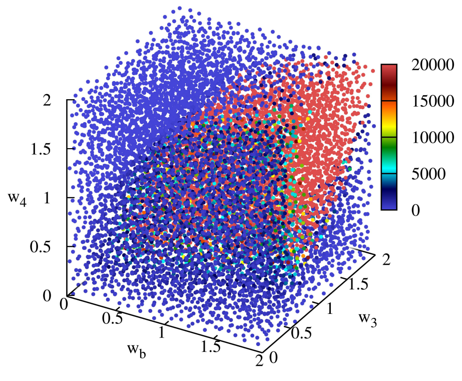


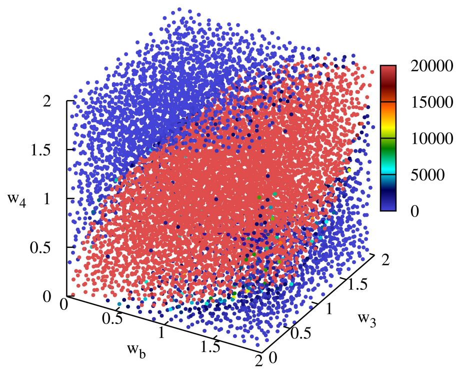
Figure 88: Divergence time on the slice $w_3 = 2$, for the MRT model up to 6 000 time steps.

However, by allowing the simulations to continue up to 100 000 time steps, it appears that some of the simulations within the two circular arcs do become unstable after a long time and the stability domain shrinks. But their distribution is highly chaotic, as can be seen on the figure 89, including a zoom on the chaotic interface area. Hence, it is unlikely that this chaotic interface could be defined by an analytical formula, which contradicts the previous prediction.

But visualising the divergence time on 2D slices only allow to partially explore the 3D space of the parameters w_b, w_3, w_4. Although it is possible to that space using N^3 sample simulations, where N is the number of samples along each axis, the computation time becomes large and it is advantageous to instead spread the samples evenly in the 3D space by sampling on a *Latin hypercube* [329]. With this sampling technique, the number of simulations required to achieve a specified spatial accuracy is significantly reduced. A response surface can also be built from these sample points and be used in an optimisation algorithm to find the location(s) of the most stable set(s) of parameters [330]. Figure 90 shows the results of ten thousands sample simulations (computed in a few hours) that cover the whole 3D domain with the same accuracy as 1000^3, i.e. one billion simulations.



(a) MRT



(b) MRT

Figure 90: Scatter plot of the divergence time for the MRT and cascaded models based on 10000 simulations sampled on a Latin hypercube.

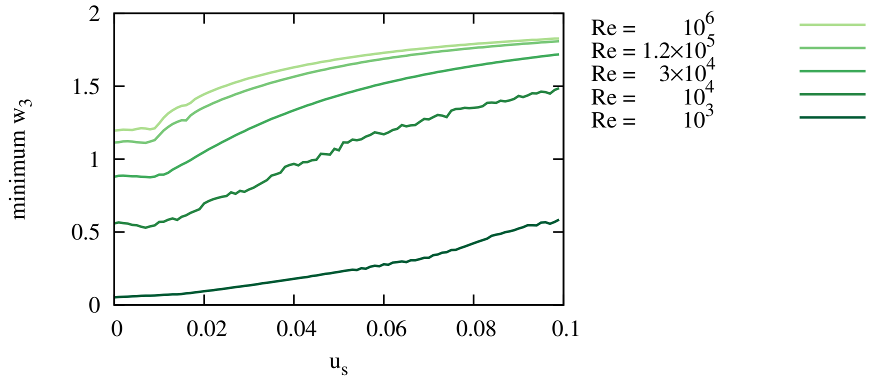


Figure 91: Minimum required value of the relaxation rate w_3 for a stable simulation with the MRT LBM against the shift velocity (u_s), for several values of Reynolds number (Re).

It is clear from this figure that the stability domain of the cascaded is larger than that of the MRT, especially along w_3.

As shown by the figures 86b, 86c and 90, the stability domain is bounded my a minimum value of the relaxation rate w_3, in other terms there is a minimum value of w_3 that is required for the simulation to be stable. In this case (i.e no shift velocity and $\mathrm{Re} = 30\,000$), this critical value is around $w_3 = 0.88$; but as the Reynolds number is increased or as a shift velocity is introduced, the stability domain of the MRT decreases in volume and the minimum w_3 required for a stable simulation increases, as shown on figure 91. According to this graph, the stability is unaltered for small shift velocities (i.e. $u_s < 0.01$) but the stability range of w_3 is significantly reduced as the shift velocity is increased further. This phenomenon is not present with the cascaded LBM, which shows that the method has an enhanced Galilean invariance over the MRT.

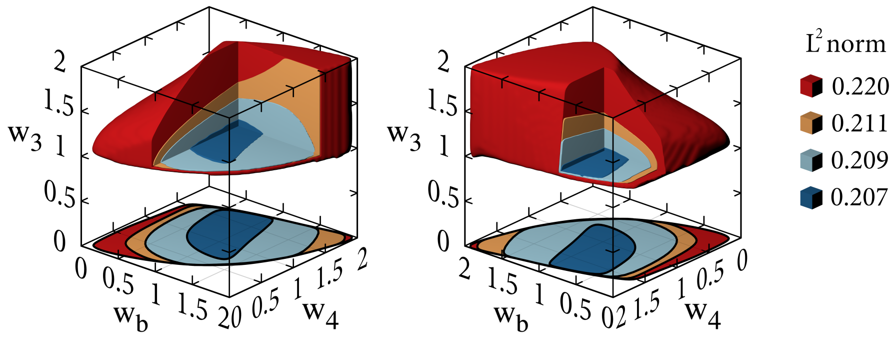


Figure 92: Isosurfaces of the L^2 norm within the stable simulation domain of w_b, w_3 and w_4.

The previous studies focused on the stability of the methods. As a complementary study, the accuracy of the results obtained with the MRT method at time $t = 1$ (see figure 83) are compared against that of the BGK, downscaled from a resolution of 512^2 (i.e. for a stable simulation). To measure the accuracy, the L^2 norm of the velocity between the MRT results and the BGK benchmark is computed, figure 92 shows several isosurfaces of constant L^2. The figure presents several shells of increasing accuracy, indicating the possibility of an *optimum* set of relaxation rates, however the difference in term of the L^2 norm between these shells is minimal. This requires further investigations.

8.6 SUMMARY

As showcased throughout the multiple sections in this chapter, the LBM program developed during this thesis can be applied to a large variety of applications. The GPU is useful in practical engineering applications, by accelerating the computations of the results and allowing a high level of interactivity; but also in more theoretical research, by allowing to research quickly a vast space of parameters.

Most of the results presented throughout this chapter should be seen as *proof of concepts* that will lead to further studies and publications from the author and collaborators.

9 CONCLUSIONS

In this thesis, a novel simulation tool for the study of indoor air flows has been developed that implements the LBM for the novel computing architecture of the GPU. The detailed optimisation of the code, combined with the natural parallelism of the LBM and the massive computational throughput of the GPU allows for the simulations to be computed in real-time, with an unprecedented level of interactivity. The resulting program has then been applied to the challenging problem of thermal management in hospitals and data centre, achieving similar levels of accuracy compared to some of the commonly used CFD software in the field.

In this chapter, conclusions are summarised and suggestions are made for future extensions to the present work.

9.1 SUMMARY OF RESULTS

Despite its unfamiliar formalism, the LBM is gaining popularity as an alternative numerical method for CFD. Originally starting as a simple cellular automaton, it has grown to become a solid numerical method to solve the Navier-Stokes equations efficiently and has been successfully applied to a large variety of applications including thermally driven flows [30, 31, 32, 33] and turbulent flows [24, 25, 26, 27, 28], hence the motivation to apply the LBM to indoor air flows.

With the constant evolution of computing architectures, the recent years have seen the GPU take a more important place in the high performance computing (HPC) scene. The GPU is a relatively new kind of processing unit that relies on massively parallel operations. As the LBM algorithm is intrinsically node based, it is sensible to implement the method on GPU, with the aim of significantly accelerating the computations. Hence, this became the main focus of this thesis.

The CUDA for C programming language was chosen in place of the alternatives because it offers a richer, more mature programming environment and allows for better performances. Though, it does limit the execution of the program to NVIDIA's GPU only. The source code was subject to extensive optimisation studies and as a result, the program can achieve more than a billion node updates per second (MLups) which equals or exceeds most of the performances reported in the literature and represents a speed increase of a factor of 100 when compared to an equivalent CPU implementation. A relationship between the MLups, the lattice resolution and the other problem parameters was established that shows that the achieved performance is sufficient for the real-time simulation of most indoor air flows.

The performance study identified the memory bandwidth as the main bottleneck for the computations and some techniques and tricks were devised to maximise it. A natural consequence is that a gaming GPU (i.e. designed for video games) can achieve higher performances at a lower price compared to a professional GPU (i.e. Tesla brand GPU), mostly because of its increased memory bandwidth.

However, due to the restrictions of the CUDA language in term of the reusability of highly optimised code, it became apparent that a higher level language was required. Hence, an additional tool was created using the LUA scripting language that is able to generate an optimised CUDA source code for a large variety of models and node type. The same tool can also generate specific boundary conditions. This improves code usability without sacrificing any of the performance.

On top of that, a real-time visualisation tool that uses the OpenGL library was integrated with the program to allow for the inspection of the flow structures as the simulation is evolving in order to save disk space and avoid long writing times. The tool also enables computational steering, the user can interact with the simulation as it is evolving to modify boundary conditions, fluid properties..., etc.

The program was then validated for a large variety of academic benchmark flows, including the lid-driven cavity and the naturally-driven double glazing problem. Most of the simulated results provide almost a perfect match when compared against the analytical profiles (used when available) or against published benchmark results from simulations (used when no analytical solution is available). During the process, it was shown that when approached carefully, single precision floating point operations achieve a similar accuracy to double precision while being twice as fast.

The validation was later extended to the case of indoor air flows. These flows provide more of a challenge because their are both transient and turbulent, but the simulation results from the LBM compared favourably against equivalent simulations obtained with various commercial CFD software. Especially in the case of the data centre model, where the velocity and thermal profiles are accurately predicted. While simulating the flows in real-time or faster.

Finally, the application of the program to a few other applications was showcased, including different multiphase flows and fluid structure interaction problems. While these applications are not directly linked to indoor air flows, they were part of different collaborations that took place during the thesis and they demonstrate the applicability of the developed software to a large variety of fluid flow problems. However, the presented results were brief and little detailed, they should lead to a large amount of future work.

In the last section, the stability of the TRT, MRT and cascaded collision operator is discussed. The massive computational power of the GPU allowed to explore the stability domain in the multi-dimensional space of the free relaxation rates with an unprecedented level of details. Hence the GPU is not only useful to accelerate the computations for engineering systems (and enables simulations in real-time) but it is also useful in optimisation processes (by allowing for a faster evaluation of the parameter space).

9.2 FUTURE WORK

Although the results presented here are numerous and promising, in some ways they only represent a beginning and the simulation tool developed during the thesis is still far off a finished CFD software. Possible directions for future research are therefore now suggested. They fall into two categories : (1) improvements to the source code and the program, and (2) possible future research areas.

The source code developed during the thesis does allow for a large variety of problems to be simulated, but its usability is limited by the need to implement boundary conditions and problem description directly in a program. This has been largely simplified through the use of LUA, but the program would benefit from a graphical user interface (GUI) for geometry construction and parameter modification at runtime as well as a commodity tool for converting 3D geometry from some standard file type into voxel data.

Adding new LBM models to the code generator that are not similar to an existing one requires to know both the intricacy of LUA and CUDA. It would be advantageous to allow new model definition by text templates that will be converted by LUA into a CUDA source code rather than constructing the source code directly from LUA (similar to how Sailfish works [255]). Moreover, the code generator could be extended to generate OpenCL code to allow the execution on other GPU models as well as CPUs.

The real-time OpenGL visualisation offers an efficient way to study the flow structures as they evolve, but its execution is limited to a local workstation. As it might become necessary to run the simulation on a distant GPU (for instance on a dedicated server), the visualisation would require significant changes. During some preliminary work, the visualisation was successfully piped to a simple web interface through an http server running on the same machine as the simulation. This could further extended to allow for a fully interactive real-time remote visualisation.

As the program is applied to more complex models, the use of multiple GPUs might become necessary to maintain the real-time capability. The achieved performance on two GPUs using peer to peer memory access is promising but the scalability of the code beyond two GPUs is unknown (and would require MPI communications for use on a large server).

It is also likely that complex models with fine geometry details will require some kind of grid refinement technique. The impact of such techniques on the performance on GPU is unknown. Moreover, it might be beneficial to use interpolation techniques (such as interpolated bounce-back for instance) for geometries that are misaligned with the grid.

The Smagorinsky turbulence model, used in some of the simulations throughout this thesis is in theory only applicable to homogeneous isotropic turbulence. Further work is required to implement custom near-wall turbulent models or even dynamic turbulence models to avoid the issue on the choice of the Smagorinsky constant.

A recurring concern with simulations at high Reynolds number is the apparition of a spurious chequerboard pattern in the velocity and temperature fields due to numerical instabilities. This pattern is linked to the second order spatial convergence and could be tackled with a flux limiter scheme, but both the MRT and cascaded have been shown to significantly decrease this effect. It would be interesting to implement these two models in 3D to see how it affects the simulations.

This work only had scope to accelerate the calculation of the LBM using an optimised implementation on GPU, but the creation of an effective smart hospital (or data centre) will require to overcome several additional challenges.

1. Ability for accurate real-time and faster than real-time simulations.

2. Accurate, non invasive, real-time sensors to capture the flow structure, both in term of flow rates and temperatures.

3. A way to interpret point data from the sensors into volumetric boundary conditions for the simulation.

4. A smart system that interprets the simulation results and update the system through the use of some actuators.

While the first point was the main focus of the thesis and the LBM on GPU does seem capable to achieve accurate real-time simulation of indoor air flows, each of the other points will require special focus that will need expertise from different fields. This is essentially a multidisciplinary problem.

A

NODE DESCRIPTIONS

The implement of the lattice Boltzmann method relies heavily on the supporting lattice structure which can be described by looking at the number and directions of the microscopic velocities of a single node. The naming convention for a generic lattice structure is DdQq where d is the dimension of the lattice (either 2 or 3) and q is the number of microscopic velocities directions.

The following presents the most common lattice structures, i.e. D2Q9 and D3Q19, with the labelling of the directions as used throughout this document and the corresponding weighing factors needed for the calculation of the equilibrium distribution functions in equation 28.

The labelling of the directions is somehow arbitrary, but follows some general rules:

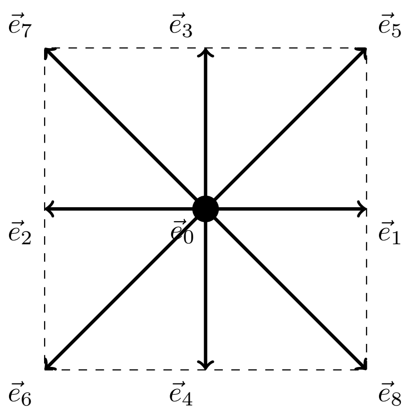
- the centre zero velocity is always called $\vec{e}_0 = \vec{0}$,
- main axes have lower index values than diagonals,
- the number of directions (including zero) must be an odd number for isotropy,
- two opposite directions must have sequential indices,

$$\vec{e}_{2i} = -\vec{e}_{2i-1}, \quad i \in \left\{1, 2, \dots, \frac{q-1}{2}\right\} \tag{166}$$

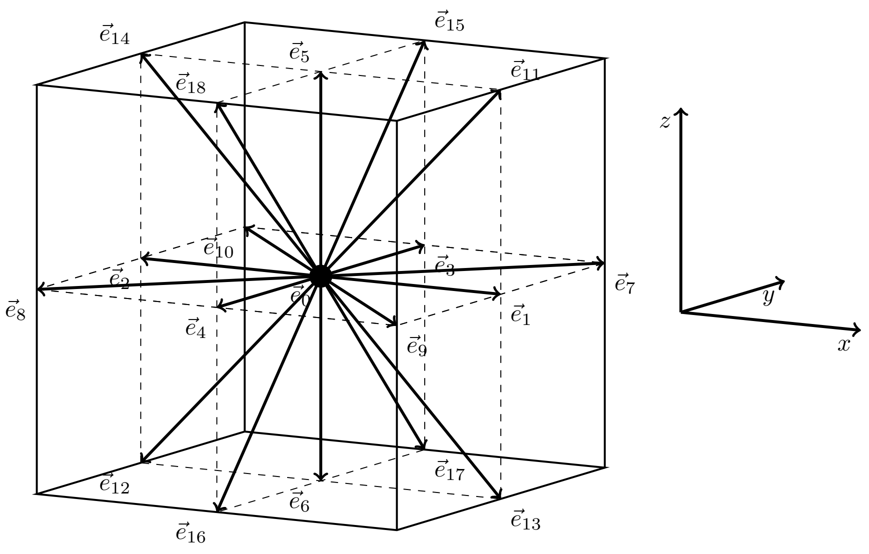
  so that they can be grouped easily in multiple situations like bounce-back (3.2.3), TRT 2.1.5, stress-tensor (2.5.1), etc.

A.1 D2Q9

$$\begin{aligned}
\vec{e}_0 &= (\ \ 0,\ \ 0) & w_0 &= 4/9\\
\vec{e}_1 &= (\ \ 1,\ \ 0) & w_1 &= 1/9\\
\vec{e}_2 &= (-1,\ \ 0) & w_2 &= 1/9\\
\vec{e}_3 &= (\ \ 0,\ \ 1) & w_3 &= 1/9\\
\vec{e}_4 &= (\ \ 0,-1) & w_4 &= 1/9\\
\vec{e}_5 &= (\ \ 1,\ \ 1) & w_5 &= 1/36\\
\vec{e}_6 &= (-1,-1) & w_6 &= 1/36\\
\vec{e}_7 &= (-1,\ \ 1) & w_7 &= 1/36\\
\vec{e}_8 &= (\ \ 1,-1) & w_8 &= 1/36
\end{aligned} \tag{167}$$



A.2 D3Q19



$$
\begin{array}{rlr}
\vec{e}_0 = (& 0, \; 0, \; 0) & w_0 = 1/3 \\
\vec{e}_1 = (& 1, \; 0, \; 0) & w_1 = 1/18 \\
\vec{e}_2 = (& -1, \; 0, \; 0) & w_2 = 1/18 \\
\vec{e}_3 = (& 0, \; 1, \; 0) & w_3 = 1/18 \\
\vec{e}_4 = (& 0, -1, \; 0) & w_4 = 1/18 \\
\vec{e}_5 = (& 0, \; 0, \; 1) & w_5 = 1/18 \\
\vec{e}_6 = (& 0, -0, -1) & w_6 = 1/18 \\
\vec{e}_7 = (& 1, \; 1, \; 0) & w_7 = 1/36 \\
\vec{e}_8 = (& -1, -1, \; 0) & w_8 = 1/36 \\
\vec{e}_9 = (& 1, -1, \; 0) & w_9 = 1/36 \\
\vec{e}_{10} = (& -1, \; 1, \; 0) & w_{10} = 1/36 \\
\vec{e}_{11} = (& 1, \; 0, \; 1) & w_{11} = 1/36 \\
\vec{e}_{12} = (& -1, \; 0, -1) & w_{12} = 1/36 \\
\vec{e}_{13} = (& 1, \; 0, -1) & w_{13} = 1/36 \\
\vec{e}_{14} = (& -1, \; 0, \; 1) & w_{14} = 1/36 \\
\vec{e}_{15} = (& 0, \; 1, \; 1) & w_{15} = 1/36 \\
\vec{e}_{16} = (& 0, -1, -1) & w_{16} = 1/36 \\
\vec{e}_{17} = (& 0, \; 1, -1) & w_{17} = 1/36 \\
\vec{e}_{18} = (& 0, -1, \; 1) & w_{18} = 1/36
\end{array}
\tag{168}
$$

B UNIT CONVERSION

A recurring issue with the LBM, when applied to engineering applications is the need to convert the physical parameters into the parameters of the LBM simulation, which are intrinsically *adimentional* or *dimensionless*, that is, the variables described by the LBM do not have dimensions. Although the rendering of a physical system into an adimensional one is a well know process in CFD, the similar process of converting units for LBM is often a source of error and confusion, especially when starting with the method. This section tries to give a clear overview of the unit conversion process, accompanied with some examples of unit conversion for concrete problems in order to give the reader a better understanding of the principles.

Rendering a system dimensionless, *by bringing out its distinctive scales, is a familiar concept in physics. That is how the dimensionless quantities that describe the behaviour of a system can be brought into the spotlight and how the Reynolds number is constructed from the Navier-Stokes equation. It also justifies the concept of* dynamic similitude, *i.e., replacing a system by an equivalent scaled model.*

In order to convert from the physical units, which will denoted with an underscore $_{\mathrm{phys}}$, into the lattice Boltzmann units, denoted with an underscore $_{\mathrm{lbm}}$, that are used in the simulations, the coefficients of conversions for each physical dimension need to be determined.

For example,

$$L_{\mathrm{phys}} = C_L L_{\mathrm{lbm}} \qquad \text{and} \qquad T_{\mathrm{phys}} = C_T T_{\mathrm{lbm}} \tag{169}$$

are linking the physical length scale L_{phys} to the LBM length scale L_{lbm} and the physical time scale T_{phys} to the LBM time scale T_{lbm}. As the LB units are dimensionless, the dimensions on the right side of the above equations are carried out by the conversions factors C_L (has dimension of a length, e.g. in meters) and C_T (has the dimension of a time, e.g. in seconds). There only exist 7 base units, as defined by the *international system of units* (SI), of which only 4 are relevant to CFD (length, time, mass and temperature). Hence it is very likely for a system to use more units than the base units, thus some of the physical quantities can expressed in term of the other quantities. As an example, it can be shown that the conversion factor for the velocity can be written as

$$C_V = C_L/C_T, \tag{170}$$

which means that the physical velocity can be converted using

$$V_{\mathrm{phys}} = \frac{C_L}{C_T} V_{\mathrm{lbm}}. \tag{171}$$

The best strategy when approaching unit conversion is to make a list of all the quantities that need to be converted and to work out the conversion factors for all of them. While the SI system advocates using length and time as the base units, these are not always the wisest choice for a particular problem. For instance, a characteristic length and speed for a system might be readily available but not a characteristic time. In which case, it is better to express the time scale in term of the length and velocity scales.

The unit conversion method can also be used in combination with the *dynamic similitude* principle. Sometimes it might be convenient to artificially increase the Mach number (Ma) to increase the time step. Indeed it can be shown that

$$\Delta t_{phys} \sim \frac{\Delta x_{phys}}{V_{phys}} Ma\,. \tag{172}$$

EXAMPLE 1 LID DRIVEN CAVITY

The lid driven cavity, as described in section 6.2, is a classical benchmark in CFD. In this problem, the dimension of the cavity and the fluid properties are often omitted to focus instead on the Reynolds number. In most problems, it is advisable to first formulate the problem in term of the dimensionless quantities that characterise the flow, i.e. Reynolds and Rayleigh numbers.

Three quantities need to be defined (in lattice units) for the simulation : the resolution or number of nodes along the length of the cavity (N), the velocity of the top lid (U_{lbm}), the relaxation time (τ). But there are only two fundamental units in this problem : length and time, hence one of the quantities has to be defined in term of the others.

One possible choice is to fix the resolution N and the top lid velocity U_{lbm}, then the relaxation time is computed as

$$\tau_{lbm} = \frac{1}{2} + 3\nu_{lbm} = \frac{1}{2} + 3\frac{U_{lbm}\,(N-1)}{Re}\,, \tag{173}$$

where the viscosity (in lattice unit) can be estimated from the Reynolds number

$$Re = \frac{U_{lbm}\,(N-1)}{\nu_{lbm}}\,. \tag{174}$$

Another possibility is to fix the resolution N and the relaxation time ν_{lbm}, in which case the lid velocity can be obtained as

$$U_0 = \frac{Re\nu_{lbm}}{N-1} = \frac{Re\left(\tau - \frac{1}{2}\right)}{3\,(N-1)}\,. \tag{175}$$

Yet another possibility is to compute the resolution N from the relaxation time and velocity, but this would only give an approximated Reynolds number as N must be an integer. In any case, it is important to check that all the parameters are within an acceptable range.

While the above is sufficient to simulate a lid driven cavity at a given Reynolds number, it does not take the physical time into account. The conversion formula for the time-step Δt gives

$$\Delta t_{\mathrm{phys}} = C_T \Delta t_{\mathrm{lbm}} = C_T = \frac{C_L}{C_V} = \frac{L_{\mathrm{phys}}}{(N-1)} \frac{U_{\mathrm{lbm}}}{U_{\mathrm{phys}}}, \tag{176}$$

as the time-step in lattice unit is 1. Hence, the physical cavity length and lid velocity need to be known to estimate the physical time-step.

C FAST FLUID DYNAMICS

C.1 HISTORY

In 1999, Jos Stam introduced in his article *stable fluid* [331], a new method to simulate fluid in real-time. Interestingly, the article was published in the conference proceedings of SIGGRAPH99 (short for **S**pecial **I**nterest **G**roup on **GRAPH**ics and Interactive Techniques), an annual conference in computer graphics. The method was thus intended as a way of quickly computing visually appealing fluid animations, rather than as an accurate solver of the Navier-Stokes equations. The algorithm is unconditionally stable, which is another indication that it is destined to be used by artists rather than engineers.

The implementations of Jos Stam's algorithm can vary, as does the names used to describe the method. One can find reference to the *stable-fluid method*, the *fast-fluid method*, the *semi-Lagrangian method*, etc... for more clarity, it will refer to within this chapter as the Jos Stam*'s method*.

However, any implementation rely on a Helmholtz-Hodge decomposition [151], that decouples the velocity field into its divergence free part and its irrotational part, combined with a quick (and cheap) way of computing the advection. Most of the errors of the method are introduced in the resolution of the advective term in the Navier-Stokes equations. Other advection schemes, more accurate than the one described by Jos Stam, do exist but often reduce the stability of the method [332].

Implementations of Jos Stam algorithm on the GPU can be found as early as 2003, using shaders [333], and a CUDA implementation can be found as part of the CUDA SDK [332]. Since then, the algorithm has been used proficiently in the film industry[1], and some attempts have been made to apply it to engineering applications [288, 287, 290].

C.2 THEORY

C.2.1 *The Navier-Stokes equation*

The Navier-Stokes equations, describing the evolution of an incompressible fluid, can be written as:

1 Jos Stam was awarded the Academy Award for Technical Achievement during the 2008 Oscar ceremony for the design and implementation of the Maya Fluid Effects system.

$$\frac{\partial \vec{u}}{\partial t} + \left(\vec{u} \cdot \vec{\nabla}\right) \vec{u} = -\frac{1}{\rho}\vec{\nabla} p + \nu \vec{\nabla}^2 \vec{u} + \vec{F} \tag{177}$$

$$\vec{\nabla} \cdot \vec{u} = 0 \tag{178}$$

where $\vec{u}\,(\vec{x}, t) = (u_x, u_y, u_z)$ is the velocity field, $p\,(\vec{x}, t)$ is the pressure field, ρ is the density, ν is the kinematic viscosity, $\vec{F}$ is an external force applied to the fluid, and $\vec{\nabla} = (\partial_x, \partial_y, \partial_z)$ is the nabla operator.

C.2.2 *Helmholtz-Hodge decomposition theorem*

The Helmholtz-Hodge decomposition theorem states that a vector field $\vec{u}\,(\vec{x}, t)$ defined on a domain D can be decomposed (in a unique way) in the form:

$$\vec{u} = \vec{u}_1 + \vec{u}_2 \tag{179}$$

where $\vec{u}_1$ is divergence-free ($\vec{\nabla} \cdot \vec{u} = 0$), and $\vec{u}_2$ is irrotational ($\vec{\nabla} \times \vec{u}_2 = 0$) and can be rewritten as $\vec{u}_2 = \vec{\nabla}\phi$ (because $\vec{\nabla} \times \left(\vec{\nabla}\phi\right) = 0$).

This theorem states that any vector field can be decomposed into the sum of two other vector fields, a divergence-free vector field, and a irrotational field.

C.2.3 *Chorin's projection algorithm*

The algorithm introduced by [151] uses the Helmholtz-Hodge decomposition to solve the equation (177) in two steps:

1. The first step computes an approximation of the vector field at the next time-step, written $\tilde{u}$.

2. The second step corrects $\tilde{u}$ so that the resulting velocity field satisfies the incompressibility condition of equation (178).

First Step

If the pressure p is zero (supposedly), the equation (177) becomes, after an explicit integration in time:

$$\frac{\tilde{u} - \vec{u}^{(n)}}{\Delta t} + \left(\vec{u}^{(n)} \cdot \vec{\nabla}\right) \vec{u}^{(n)} = \nu \vec{\nabla}^2 \vec{u}^{(n)} + \vec{F} \tag{180}$$

where $\vec{u}^{(n)}$ is the velocity field at the n-th time step.

$$\tilde{u} = \vec{u}^{(n)} + \Delta t \left(-\left(\vec{u}^{(n)} \cdot \vec{\nabla}\right) \vec{u}^{(n)} + \nu \vec{\nabla}^2 \vec{u}^{(n)} + \vec{F}\right) \tag{181}$$

In order to compute the intermediate velocity field $\tilde{u}$ the advection operator $\left(\vec{u}^{(n)} \cdot \vec{\nabla}\right) \vec{u}^{(n)}$ and diffusion operator $\vec{\nabla}^2 \vec{u}^{(n)}$ would have to be discretised (using either finite difference or finite volume for example).

Second Step

After the first step, the intermediate velocity field $\tilde{u}$ does not satisfy equation 178, hence it is not divergence-free. But the Helmholtz-Hodge decomposition theorem tells us that it can be decomposed into a unique divergence free velocity field $\vec{u}^{(n+1)}$ and an irrotational velocity field $\vec{u}' = \vec{\nabla}\phi$.

$$\tilde{u} = \vec{u}^{(n+1)} + \vec{u}' \tag{182}$$

The velocity field $\tilde{u}$ needs to be corrected so that $\vec{u}^{(n+1)} = \tilde{u} - \vec{u}'$ satisfies equation (177). Replacing $\vec{u}^{(n+1)}$ in equation (177) gives:

$$\frac{(\tilde{u} - \vec{u}') - \vec{u}^{(n)}}{\Delta t} + \left(\vec{u}^{(n)} \cdot \vec{\nabla}\right) \vec{u}^{(n)} = -\frac{1}{\rho}\vec{\nabla} p^{(n+1)} + \nu \vec{\nabla}^2 \vec{u}^{(n)} + \vec{F} \tag{183}$$

which simplifies in

$$\vec{u}' = \frac{\Delta t}{\rho} \vec{\nabla} p^{(n+1)} \tag{184}$$

so

$$\vec{u}^{(n+1)} = \tilde{u} - \frac{\Delta t}{\rho} \vec{\nabla} p^{(n+1)} \tag{185}$$

The pressure $p^{(n+1)}$ is still unknown, but can be computed using the incompressibility condition $\vec{\nabla} \cdot \vec{u}^{(n+1)} = 0$ which implies

$$\vec{\nabla}^2 p^{(n+1)} = \frac{\rho}{\Delta t} \vec{\nabla} \cdot \tilde{u} \tag{186}$$

This is the standard Poisson equation and can be solved as a linear system using various numerical methods.

After the computation of the pressure, the intermediate velocity field $\tilde{u}$ can be corrected by subtracting from it the gradient of the pressure. The resulting velocity field $\vec{u}^{(n+1)}$ satisfies both the Navier-Stokes equation (177) and the incompressibility equation (178). This step is sometimes called *projection* because the approximative velocity field $\tilde{u}$ is projected onto a new divergence free velocity field $u^{(n+1)}$.

C.3 Jos Stam's Algorithm

The algorithm proposed in [331] is based on the Chorin's projection algorithm, but it introduces several approximation in order to make the computation faster and unconditionally stable. This section presents the differences between Jos Stam's algorithm and the classical Chorin's projection algorithm, as well as some details on the discrete operators used in the Jos Stam's method (only in two dimensions for simplicity).

C.3.1 *Summary of the method*

1. Computation of $\tilde{u}$ using the relation:

$$\frac{\partial \tilde{u}}{\partial t} = -\left(\vec{u}^{(n)} \cdot \vec{\nabla}\right) \vec{u}^{(n)} + \nu \vec{\nabla}^2 \vec{u}^{(n)} + \vec{F} \tag{187}$$

2. Computation of ϕ:
   Compute $\vec{\nabla} \cdot \tilde{u}$, then solve

$$\vec{\nabla}^2 \phi = \vec{\nabla} \cdot \tilde{u} \tag{188}$$

3. Computation of $\vec{u}^{(n+1)}$ using the relation:

$$\vec{u}^{(n+1)} = \tilde{u} - \vec{\nabla}\phi \tag{189}$$

c.3.2 *Advection*

Instead of solving the advection operator $\left(\vec{u} \cdot \vec{\nabla}\right) \vec{u}$ through a computationally expensive finite difference scheme, the velocity field is updated like a particle system, where each grid cell is represented by a particle. Using an explicit integration in time, the position $\vec{x}(t)$ of a particle advected by the velocity field $\vec{u}(\vec{x}, t)$ is updated with:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{u}(\vec{x}, t)\Delta t \tag{190}$$

Fluid quantities (such as velocity, temperature or density) can be advected in the same way. One issue of this explicit discretisation in time is that it can result in unstable simulation if the magnitude of $\vec{u}\Delta t$ is greater than the size of a grid cell.

The solution is an implicit discretisation in time. Instead of advecting quantities using the local value of the velocity field, they can be traced back in time at the position where the particle was at the previous time step. The implicit time integration of a fluid quantity q is achieved with the following formula :

$$q\left(\vec{x}, t + \Delta t\right) = q\left(\vec{x} - \vec{u}\left(\vec{x}, t\right)\Delta t, t\right) \tag{191}$$

In general, the position computed from $\vec{x} - \vec{u}\left(\vec{x}, t\right)$ does not coincide with a node centre, and the value of q must be interpolated from the nearest neighbours.

c.3.3 *Diffusion*

The viscous equation $\frac{\partial \vec{u}}{\partial t} = \nu \vec{\nabla}^2 \vec{u}$ can be solved using an explicit discretisation

$$\vec{u}\left(\vec{x}, t + \Delta t\right) = \vec{u}(\vec{x}, t) + \nu \Delta t \vec{\nabla}^2 \vec{u}\left(\vec{x}, t\right) \tag{192}$$

where $\vec{\nabla}^2$ is the discrete form of the Laplace operator. Once again, this is not stable for large time-steps, and it is better to use the implicit formulation instead :

$$\left(\boldsymbol{I} - \nu \Delta t \vec{\nabla}^2\right) \vec{u}\left(\vec{x}, t + \Delta t\right) = \vec{u}(\vec{x}, t) \tag{193}$$

where $\boldsymbol{I}$ is the identity matrix. This a linear system of equations, which can be solved numerically in various ways, but Jos Stam proposed to use the Jacoby algorithm, an iterative solution technique that starts with an approximate solution and improves on it at every iteration.

Transforming a vector field equation into a matrix equation

As, discussed previously, the viscous equation can be rewritten using an implicit time-stepping scheme as:

$$\left(\boldsymbol{I} - \nu\Delta t\vec{\nabla}^2\right)\vec{u}\left(\vec{x}, t+\Delta t\right) = \vec{u}(\vec{x}, t) \tag{194}$$

But this is an equation on vector fields, i.e. $\vec{u}(\vec{x}) = (u_x(x, y, z), u_y(x, y, z), u_z(x, y, z))$, and in order to be solved by a computer, it needs to be expressed as a matrix equation. Usually for a two dimensional simulation, $\vec{u}(\vec{x})$ is stored on the computer as a 2D matrix, or more precisely, each component of $\vec{u}(\vec{x})$ is stored as a 2D matrix. But in order to transform the discrete viscous equation into a system of linear equation, each component of $\vec{u}(\vec{x})$ is stored as a one dimensional vector. The following tables show how to transform a 2D matrix of size (N_x, N_y) into a 1D vector of size $N_x \times N_y$:

$$\begin{bmatrix} u_{1,1} & u_{2,1} & \cdots & u_{N_x,1} \\ u_{1,2} & u_{2,2} & \cdots & u_{N_x,2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1,N_y} & u_{2,N_y} & \cdots & u_{N_x,N_y} \end{bmatrix} = \begin{bmatrix} u_{1,1} \\ \vdots \\ u_{N_x,1} \\ u_{1,2} \\ \vdots \\ u_{N_x,N_y} \end{bmatrix} \tag{195}$$

The Jacobi algorithm

The Jacobi algorithm can solve a matrix equation of the form $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{a}$ is the vector of values to solve (in this case either x- or y-component of $\vec{u}$ or the pressure ϕ), $\boldsymbol{b}$ a vector of constants, and $\boldsymbol{A}$ is a matrix (i.e., the Laplace operator).

The Jacoby algorithm starts with an initial guess[2] for the solution $\boldsymbol{x}^{(0)}$, and each step k produces an improved solution $\boldsymbol{x}^{(k)}$. For the Poisson equation, involving solely the Laplace operator, the matrix $\boldsymbol{A}$ is a tri-diagonal matrix, and is mainly filled with zeros. It does not have to be built in memory, and the next iteration can be directly computed from the previous one using the following formula :

$$x_{i,j}^{(k+1)} = \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} + \alpha b_{i,j}}{\beta} \tag{196}$$

2 A good initial guess is often the value of the field at the previous time-step.

where $x_{i,j}$ and $b_{i,j}$ both represent one of the components of the velocity at location (i, j), $\alpha = \left(\Delta x^2\right)/\nu\Delta t$ and $\beta = 4 + \alpha$.

C.3.4 *Force*

The force field $\vec{F}(\vec{x}, t)$ is simply added to the current velocity field, by an explicit Euler integration in time :

$$\vec{u}\left(\vec{x}, t + \Delta t\right) = \vec{u}\left(\vec{x}, t\right) + \Delta t \vec{F}(\vec{x}, t) \tag{197}$$

C.3.5 *Projection*

Once the intermediate velocity $\tilde{u}$ is computed, it needs to be projected onto a divergence-free velocity-space.

First, the Poisson equation $\vec{\nabla}^2\phi = \vec{\nabla} \cdot \tilde{u}$ is solved using the Jacobi algorithm as described above :

$$\phi_{i,j}^{(k+1)} = \frac{\phi_{i-1,j}^{(k)} + \phi_{i+1,j}^{(k)} + \phi_{i,j-1}^{(k)} + \phi_{i,j+1}^{(k)} - \Delta x^2 \left(\vec{\nabla} \cdot \tilde{u}\right)_{i,j}}{4} \tag{198}$$

where ϕ is linked to the real pressure via $\phi = p\Delta t$, and $\vec{\nabla} \cdot \tilde{u}$ is computed through finite difference.

Then, the gradient of ϕ (computed with finite difference) is subtracted from the intermediate velocity $\tilde{u}$.

C.4 IMPLEMENTATION

All of the operations in Jos Stam's algorithm are local to a node, or involve the closest neighbours, and are hence easily amenable to the implementation on GPU. A sample implementation can be found as part of the CUDA SDK [334].

For this thesis work, the algorithm was implemented in three dimensions, including the advection-diffusion of temperature and the vorticity confinement for the turbulence (see [335]), on the GPU using CUDA. The velocity $\boldsymbol{u}$ is stored as three 3D arrays (i.e. as a vector field) and the pressure ϕ is stored as a one 3D array (i.e. as a scalar field). A temporary velocity and pressure fields ($\boldsymbol{u}^{\text{tmp}}$ and ϕ^{tmp}) are also allocated to avoid overwriting cells data.

Each step of the algorithm is implemented as a CUDA kernel, and the program has the following structure :

1. Advection : $\vec{u}_{x,y,z}^{\text{tmp}} = \vec{u}_{x_d,y_d,z_d}$,
   where $\vec{x}_d = (x_d, y_d, z_d) = \vec{x} - \vec{u}_{x,y,z}\Delta t$. As $\vec{x}_d$ does not, in general, coincide with the grid, $\vec{u}_{x_d,y_d,z_d}$ is obtained as a tri-linear interpolation of the 6 neighbour values.

2. Diffusion :

$$\begin{aligned}\vec{u}_{x,y,z} = \Big(&\vec{u}^{tmp}_{x,y,z} + \nu\Delta t\big(\vec{u}^{tmp}_{x-1,y,z} + \vec{u}^{tmp}_{x+1,y,z}\\ &+ \vec{u}^{tmp}_{x,y-1,z} + \vec{u}^{tmp}_{x,y+1,z}\\ &+ \vec{u}^{tmp}_{x,y,z-1} + \vec{u}^{tmp}_{x,y,z+1}\big)\Big)/\left(1+6\nu\Delta t\right)\end{aligned}, \qquad (199)$$

The above operation is reiterated for a specified amount of times.

3. Projection :

   a) Divergence of velocity :

   $$\begin{aligned}\phi^{tmp}_{x,y,z} = &\left(u_{x+1,y,z} - u_{x-1,y,z}\right)/2\\ + &\left(v_{x,y+1,z} - v_{x,y-1,z}\right)/2,\\ + &\left(w_{x,y,z+1} - w_{x,y,z-1}\right)/2\end{aligned} \qquad (200)$$

   where $\vec{u} = (u, v, w)$ in the above.

   b) Diffusion of ϕ, similar to equation (199).

   c) Subtract the gradient of ϕ to the velocity :

   $$\begin{aligned}u_{x,y,z} &= u_{x,y,z} - \left(\phi_{x+1,y,z} - \phi_{x-1,y,z}\right)/2\\ v_{x,y,z} &= v_{x,y,z} - \left(\phi_{x,y+1,z} - \phi_{x,y-1,z}\right)/2,\\ w_{x,y,z} &= w_{x,y,z} - \left(\phi_{x,y,z+1} - \phi_{x,y,z-1}\right)/2\end{aligned} \qquad (201)$$

4. Add force : $\vec{u}_{x,y,z} = \vec{u}_{x,y,z} + \vec{F}_{x,y,z}\Delta t$, followed by another application of step 3 to re-establish a divergence-free field.

C.5 NUMERICAL EXPERIMENT

To test the accuracy of the method, the 3D lid-driven cavity, as described in section 6.2, is simulated using a 128^3 resolution, $\Delta t = 0.001$ and 100 iterations of the Jacobi algorithm used in the diffusion for a Reynolds number of 1000. It should be noted that for these settings, the simulation is not real-time, because they are meant to yield a good accuracy. Yet, as show on the figure 93, the accuracy of the method when compared to the benchmark data is not satisfactory, even in such a simple case.

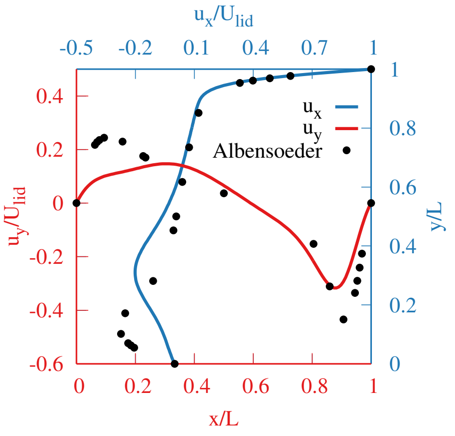


Figure 93: Velocity profiles through the geometric centre lines in the 3D lid-driven cavity using Jos STAM's model.

C.6 SUMMARY

While the fast fluid method can potentially achieve a higher MLups than the LBM, hence could be more computationally efficient, its lack of accuracy, even for simple benchmark problems caused the method to be put to the side in favour of the LBM as the core CFD solver for the real-time indoor air-flow simulation software developed during this thesis.

Nevertheless, the method is promising as it is based directly on the Navier-Stokes equations, hence the limits in which they are approximated and the range of validity of the imposed approximation can in theory be defined clearly. It would be interesting to study the effect of higher order schemes, such as the MacCormark method [332], or the improvements proposed in [287, 288].

D INTRODUCTION TO GPU PROGRAMMING IN CUDA

D.1 ARCHITECTURE

The CUDA programming model is strongly linked to the NVIDIA GPU architecture. This actual details of the architecture changes with every model and generation (see table 13), but the abstraction provided by CUDA stays the same, apart from the addition of new features with each new version, designed to simplify the programming (see section D.2).

Architecture	Release date
Tesla	2006
Fermi	2010
Kepler	2014
Pascal	2016?

Table 13: Evolution of NVIDIA GPU architecture.

D.1.1 *CUDA Thread Organization*

The architecture of Nvidia GPUs is based on highly threaded *Streaming Multiprocessors* (SMs). The Tesla K40, which was used for most of the present work, belongs to the Kepler architecture and features fifteen SM, each SM has

- 192 *Streaming Processors* (SP, i.e. cores) which performs basics operations in single precision,
- 64 *Double Precision Units* (DP Unit) for double precision computations,
- 32 *Load/Store Units* (LD/ST) for loading and storing data,
- 32 *Special Function Units* (SFU) for functions like `cos()` or `exp()`,
- as well as some dedicated memories of different types and speeds (see next section).

The architecture of one SM is represented on the figure 94. In practice, these SM are grouped by tree inside a *Graphics Processing Cluster* (GPC) and the Tesla K40 contains five GPC, in total that is $5(\mathrm{GPC}) \times 3(\mathrm{SM}) \times 192(\mathrm{SP}) = 2880$ cores. The architecture of the whole GPU is reported on figure 95.

The microprocessors manage the creation, organization and execution of the threads. To do so, the SMs use an architecture named *Single Instruction Multiple Threads* (SIMT, specific to Nvidia). A multithreaded program is partitioned into blocks of threads that execute independently from each other, so that a GPU with more cores will automatically execute the program in less time than a GPU with fewer cores. More precisely, once a block is assigned to a SM, it further

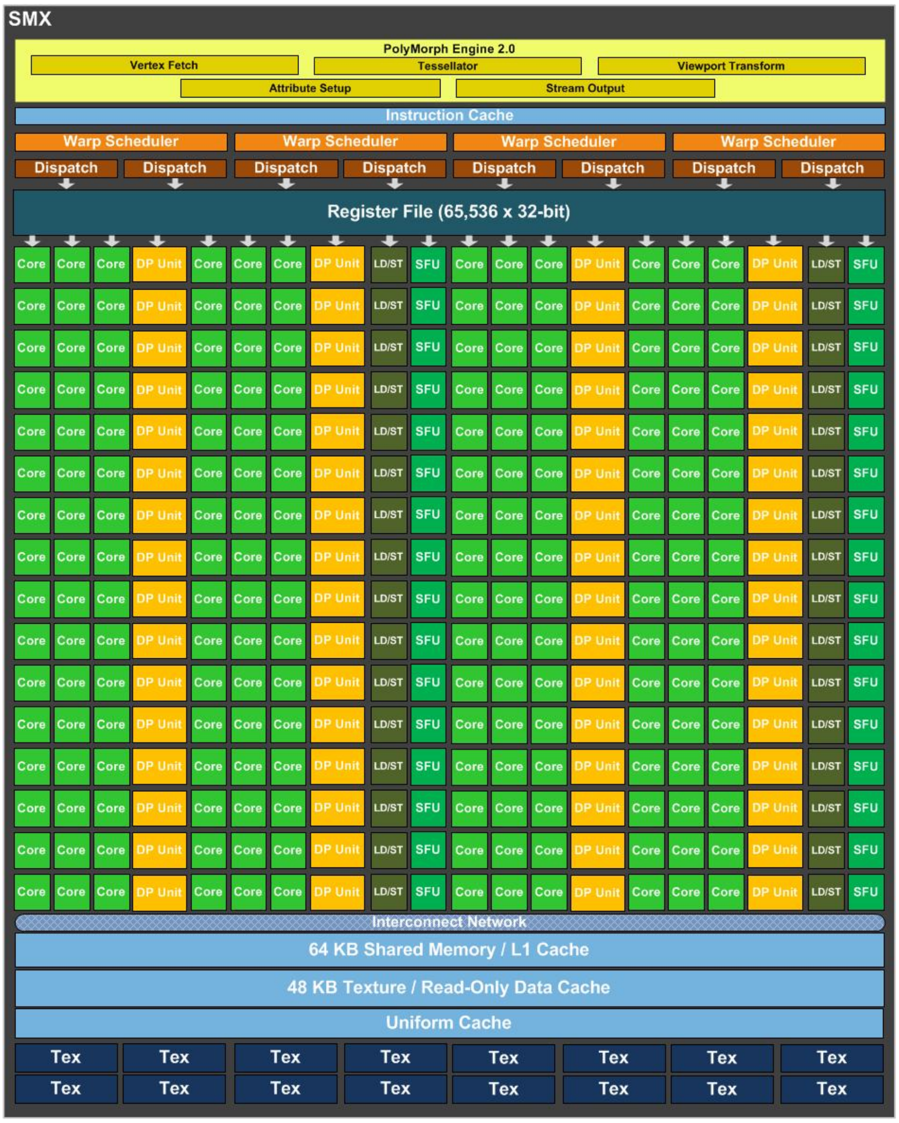


Figure 94: Architecture of one Streaming Multiprocessors for the Tesla K40 (GK110B) NVIDIA GPU. Source: NVIDIA.

divided into groups of 32-threads[1] called *warps*, scheduled by the SIMT unit. At any time, the SM executes only one of its resident warps. This allow the other warps to wait for long latency operations without slowing down the overall execution throughput of the massive number of execution units. Each SM can accommodate up to 64 warps simultaneously, so each SM can execute up to 64×32 or 2048 threads concurrently. The K40 GPU has 15 SM, it means there can be 15×2048 or 30720 threads actives at the same time. This high number of threads allow the control units to hide the latency of memory access.

Indeed, when an instruction executed by threads in a warp needs to wait for the result of a previously initiated long-latency operation

1 The size of warps is implementation specific and might vary in the future.

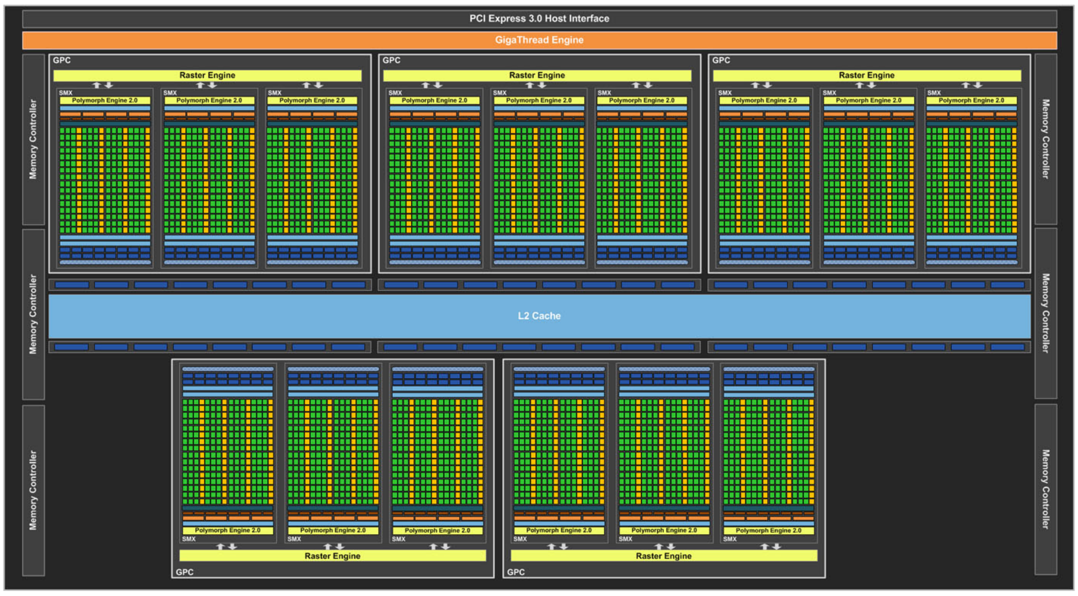


Figure 95: Architecture of the Tesla K40 (GK110B) Nvidia GPU. Source: Nvidia. (see zoom on figure 94)

(such as access to the global memory), the warp is placed in a waiting area. Meanwhile, one of the other resident warps is selected for execution by a priority mechanism.

It is worth noting that CUDA threads are of much lighter weight than typical CPU threads. CUDA programmers can assume that these threads take very few cycles to generate and schedule due to efficient hardware support. This is in contrast with the CPU threads that typically take thousands of clock cycles to generate and schedule.

D.1.2 *Memory model*

The GPU has its own Dynamic Random Access Memory (DRAM, 12GBytes for Tesla K40), which is not directly accessible by the CPU, and inversely CPU RAM is not directly accessible by the GPU. Data are copy to or from the GPU through the PCI express, so data transfers is really slow and it is crucial to minimize these transfers in a program.

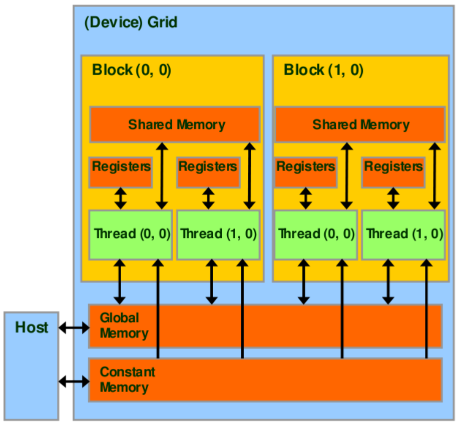


Figure 96: Overview of the CUDA device memory model. Source: NVIDIA.

Figure 96 shows an overview of the CUDA device memory model to understand the allocation, movement, and usage of the various memory types available on the device. At the bottom of the picture are represented the global memory and constant memory these are the only memory types that the host (the CPU) can write to and read from. The global memory correspond to the central memory of the GPU, it can be access by all the threads in reading and writing, it is the largest (12 GB) but it also has the greatest access latency. The constant memory is significantly smaller (64 KB) and allow read-only access by the device but it provides faster access than the global memory.

Above the thread execution boxes in Figure 96 are registers and shared memories. Variables that reside in these memories can be accessed at very high speed, however the small size of these memories (64 KB and 48 KB respectively) limit their usage to small data. Registers are allocated to individual threads; each thread can only access its own registers. A kernel function (see D.2.1) typically uses registers to hold frequently accessed variables that are private to each thread. Shared memories are allocated to thread blocks; all threads in a block can access variables in the shared memory locations allocated to the block, it is hence an efficient means for threads to cooperate.

D.2 PROGRAMING MODEL

Until 2006, graphic cards were very difficult to use because programmers had to use graphic *Application Programming Interface* (API), like OpenGL or direct3D to program these chips. These APIs are meant for graphical programing (like in video games), and are not particularly adequate for scientific computations. That's why only a few scientists could master the skills necessary to use these GPUs at that time. Consequently, it did not become a widespread programming phenomenon. Nonetheless, this technology was sufficiently exciting to inspire some heroic efforts and excellent results.

But everything changed in 2007 with the release of CUDA. The API C for CUDA is an extension of the C standard which enable the use of GPUs in order to draw all their computing power out of them. A CUDA program consists of one or more phases that are executed on either the host (CPU) or a the device (GPU). The phases that exhibit little or no data parallelism are implemented in host code. The phases that exhibit rich amount of data parallelism are implemented in the device code. The program supplies a single source code encompassing both host and device code. The NVIDIA C Compiler (NVCC) separates the two. The host code is straight ANSI C code and is compiled with the host's standard C compilers and runs as an ordinary process. The device code is written using ANSI C extended with keywords for labeling data-parallel functions, called kernels, and their associated data structures. The device code is typically further compiled by the NVCC and executed on a GPU device.

The execution of a typical CUDA program is composed of three steps:

1. upload of data from the CPU to the GPU

2. invocation of the kernel function on the GPU, this generate a large number of threads and each thread executes the some code but on different parts of the data

3. download of the resulting data from the GPU to the CPU.

D.2.1 *Threads and Kernels*

This section discusses the CUDA kernel functions and the organizations of threads generated by the invocation of kernel functions. See figure 16 for a visual representation of threads organisation. In CUDA, a kernel function specifies the code to be executed by all threads of a parallel phase. Since all threads of a parallel phase execute the same code, CUDA programming is an instance of the well-known Single-Program Multiple-Data (SPMD) parallel programming style, a popular programming style for massively parallel computing systems.

A kernel is defined by using a CUDA specific keyword : "`__global__`" in front of the declaration of a function. This keyword indicates that the function is a kernel and that it can be called from host functions to generate a grid of threads. A kernel function must return a `void`.

Listing 14: Kernel declaration in CUDA

```
//kernel declaration
__global__ void kernelFunc(type1 parameter1,
                           type2 parameter2,
                           ...)
{
  //function body
}
```

When a kernel is invoked, or launched, it is executed as grid of parallel threads. Threads in a grid are organized into a two-level hierarchy. At the top level, each grid consists of one or more thread blocks. And each block in a grid must have the same number of threads organized in the same manner. The call to a kernel has to always specify the characteristics of the grid and the blocks:

Listing 15: Kernel call in CUDA

```
//launch the device computation kernel
kernelFunc<<<grid_size,block_size>>>(parameter1,
   parameter2);
```

A grid of threads is a two dimensional array of three dimensional blocks of threads. The parameters grid_size and block_size in the previous call are the dimensions of the grid in number of blocks and the dimension of the blocks in number of threads. These value are generally of type dim3, which is a structure to define 3D dimension.

Listing 16: Grid and block definition in CUDA

```
//setup the execution configuration
dim3 grid_size(nb_block_x, nb_block_y);
dim3 block_size(nb_block_x, nb_block_y,
   nb_block_z);
```

Since all threads in a grid execute the same kernel function, they rely on unique coordinates to distinguish themselves from each other and to identify the appropriate portion of the data to process. The CUDA runtine system define two set of coordinates: `blockId` indicates which block a thread is in and `threadId` indicates the position of the thread in the block. The `blockId` and `threadId` appear as built-in variables that are initialized by the runtime system and can be accessed within the kernel functions. When a thread executes the kernel function, references to the `blockId` and `threadId` variables return the appropriate values that form coordinates of the thread. In a similar way, the variables `gridDim` and `blockDim` can be used to know the dimension of the grid and the blocks respectively.

In the grid level, the executing order of blocks in undetermined and each block is executed completely independently from the others.

In the block level, each block have the same number of threads, threads from a same block can share data through shared memory and coordinate their activities using a barrier synchronization function called `syncthreads()`.

D.2.2 *Memory Management*

In CUDA, host and devices have separate memory spaces. In order to execute a kernel on a device, the programmer needs to allocate memory on the device and transfer pertinent data from the host memory to the allocated device memory. Similarly, after device execution, the programmer needs to transfer result data from the device back to the host and free up the device memory that is no longer needed.

The function `cudaMalloc()` can be called from the host code to allocate some Global Memory. The first parameter for the `cudaMalloc` function is the address of a pointer that will point to the allocated Global Memory. The second parameter gives the requested allocation size in byte.

The data transfers are handle by the function `cudaMemcpy()` which requires four parameters. The first and the second are pointers which respectively point the destination memory address and the source address to be copied from. The third parameter specify the number of bytes to be copied. And the fourth parameters indicates the type of transfer: from host to device, from device to host, from host to host and from device to device.

After the computation, `cudaFree()` is called with pointer to device data to free the memory on the device.

Listing 17: Memory management in CUDA

```
type data[N],res[N];
int size = sizeof(type)*N;
type* devPtr;
cudaMalloc((void**)&devPtr, size);
cudaMemcpy(devPtr,data,size,
    cudaMemcpyHostToDevice);
[...] //Kernel invocation code
cudaMemcpy(res, devPtr,size,
    cudaMemcpyDeviceToHost);
cudaFree(devPtr);
```

Multidimensional arrays can be achieved by indexing a 1D array (i.e. by computing a 1D index based on a 3D location) or through the use of the dedicated functions `cudaMallocPitch()` and `cudaMallocArray()`. These functions can allow for a better alignment of data in memory, hence faster memory access.

All these functions allocate global memory, but during the execution each thread can access to the different levels of memory seen previously. CUDA defines registers, shared memory, and constant memory that can be accessed at higher speed and in a more parallel manner than the global memory.

All automatic variables except for arrays declared in a kernel or a device function are placed into registers. The scopes of these automatic variables are within individual threads. When a kernel function declares an automatic variable, a private copy of that variable is generated for every thread that executes the kernel function. When a thread terminates, all its automatic variables also cease to exist. Note that accessing these variables is extremely fast but one must be careful not to exceed the limited capacity of the register storage in the hardware implementations.

Automatic array variables are not stored in registers. Instead, they are stored into the global memory and incur long access delays and potential access congestion. The scopes of these arrays are, same as automatic scalar variable, within individual threads. Due to the slow nature of automatic array variables, one should avoid using such variables.

If a variable declaration is preceded by keywords "`__shared__`", it declares a shared variable in CUDA. Such declaration must reside within a kernel function or a device function. The scope of a shared variable is within a thread block, that is, all threads in a block see the same version of a shared variable. The lifetime of a shared variable is within the duration of the kernel. Shared variables are an efficient means for threads within a block to collaborate with each other. Accessing to shared memory is extremely fast and highly parallel. CUDA programmers often use shared memory to hold the portion of global memory data that are heavily used in an execution phase of kernel.

If a variable declaration is preceded by keywords "`__constant__`", it declares a constant variable in CUDA. Declaration of constant variables must reside outside any function body. The scope of a constant variable is all grids, meaning that all threads in all grids see the same version of a constant variable. The lifetime of a constant variable is the entire application execution. Constant variable are often used for variables that provide input values to kernel functions. Constant variables are stored in the global memory but are cached for efficient access. However, the total size of constant variables in an application is quite small (65Kb for Tesla 10 cards) so it can be used only for few constants. The function `cudaMemcpyToSymbol()` must used to transfer data from the host memory to the device constant memory.

D.3 EXECUTION MODEL

When a CUDA kernel is launched it generates a grid of threads, threads blocks are numbered and distributed to the Streaming Multiprocessors, and resources are dynamically distributed among the blocks and threads. Then all threads in a blocks run concurrently on a multiprocessor and once a block has finished its tasks, another block is launched. This scalability allows the CUDA architecture to span a wide range of GPUs by simply scaling the number of processors and memory partitions.

The execution resources in a SM include register, thread block slots, and thread slots. These resources are dynamically partitioned and assigned to thread blocks during runtime. This can alter performances, from under-utilization of the GPU to the impossibility to run if required resources are higher than available resources (for example if the number of threads per blocks is bigger than the maximum number authorized).

In CUDA, the *occupancy* is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU, and maximizing this value can help to hide the latency of memory access and increase performances. In the other hand, some optimization, like the add of a local variable (which will increase the number of registers) or the use of shared memory, will increase the number of required resources and may reduce the occupancy, but will still improve performances. It is up to the programmer to find a compromise, by experimenting.

Thereby, the computation configuration must be though in order to maximize the use of multiprocessors. Nvidia provide developers with two tools to ease this task:

- the CUDA Occupancy Calculator allows to compute the multiprocessor occupancy of a GPU by a given CUDA kernel,
- the CUDA Visual Profiler allows to analyze the execution of kernels and the use of SMs.

BIBLIOGRAPHY

[1] Sergio Hoyas and Javier Jiménez. Scaling of the velocity fluctuations in turbulent channels up to $\mathrm{Re}_\tau = 2003$. *Physics of Fluids (1994-present)*, 18(1):011702, 2006. (Cited on page 8.)

[2] C. Bailly and G. Comte-Bellot. *Turbulence*. Experimental Fluid Mechanics. Springer International Publishing, 2015. (Cited on page 8.)

[3] X. He and L.S. Luo. Lattice Boltzmann Model for the Incompressible Navier-Stokes Equation. *Journal of Statistical Physics*, 88(3-4):927–944, 1997. (Cited on pages 15, 24, 25, and 32.)

[4] Robert Geist, James Westall, and Robert Schalkoff. Lattice-Boltzmann Lighting. In *Eurographics SYmposium on Rendering*, 2004. (Cited on page 15.)

[5] Robert Geist and James Westall. Lattice-Boltzmann Lighting Models. *GPU GEMS*, 4, 2011. (Cited on page 15.)

[6] Paul J. Dellar. An exact energy conservation property of the quantum lattice Boltzmann algorithm . *Physics Letters A*, 376(1):6 – 13, 2011. (Cited on pages 15 and 22.)

[7] T. Ohwada, Pietro Asinari, and D. Yabusaki. Artificial Compressibility Method and Lattice Boltzmann Method: Similarities and Differences. *COMPUTERS & MATHEMATICS WITH APPLICATIONS*, 61(12):3461–3474, 2011. (Cited on pages 15, 22, and 48.)

[8] J. Hardy, Y. Pomeau, and O. de Pazzis. Time evolution of a two-dimensional classical lattice system. *Phys. Rev. Lett.*, 31:276–279, Jul 1973. (Cited on page 21.)

[9] Martin Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223(4): 120–123, 1970. (Cited on page 21.)

[10] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier-stokes equation. *Phys. Rev. Lett.*, 56:1505–1508, Apr 1986. (Cited on page 21.)

[11] Uriel Frisch, Dominique D'Humieres, Brosl Hasslacher, Pierre Lallemand, Yves Pomeau, and Jean-Pierre Rivet. Lattice Gas Hydrodynamics in Two and Three Dimensions. *Complex Systems*, 1(4):649–707, 1987. (Cited on page 21.)

[12] Guy R. McNamara and Gianluigi Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Phys. Rev. Lett.*, 61:2332–2335, Nov 1988. (Cited on page 21.)

[13] F. J. Higuera and J. Jiménez. Boltzmann Approach to Lattice Gas Simulations. *EPL (Europhysics Letters)*, 9(7):663, 1989. (Cited on page 21.)

[14] F. J. Higuera, S. Succi, and R. Benzi. Lattice Gas Dynamics with Enhanced Collisions. *EPL (Europhysics Letters)*, 9(4):345, 1989. (Cited on page 21.)

[15] P. J. Dellar. Incompressible limits of lattice Boltzmann equations using multiple relaxation times. *J. Comput. Phys.*, 190:351–370, 2003. (Cited on pages 22, 61, 186, and 187.)

[16] Li-Shi Luo. Theory of the lattice Boltzmann method: Lattice Boltzmann models for nonideal gases. *Phys. Rev. E*, 62:4982–4996, Oct 2000. (Cited on page 22.)

[17] R. R. Nourgaliev, T. N. Dinh, T. G. Theofanous, and D. Joseph. The lattice Boltzmann equation method: Theoretical interpretation, numerics and implications. *Int. Multiphase Flow, J*, pages 117–169, 2003. (Cited on page 22.)

[18] Santosh Ansumali and IliyaV. Karlin. Entropy function approach to the lattice boltzmann method. *Journal of Statistical Physics*, 107(1-2): 291–308, 2002. (Cited on pages 22 and 45.)

[19] Martin Geier, Andreas Greiner, and Jan G. Korvink. Cascaded digital lattice boltzmann automata for high reynolds number flow. *Phys. Rev. E*, 73:066705, Jun 2006. (Cited on pages 22 and 47.)

[20] D. d'Humieres, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.S. Luo. Multiple-Relaxation-Time Lattice Boltzmann Models in Three Dimensions. *Phil. Trans. R. Soc. A*, 360:437–451, 2002. (Cited on pages 22, 25, and 61.)

[21] Pietro Asinari, T. Ohwada, Eliodoro Chiavazzo, and Antonio Fabio Di Rienzo. Link-wise Artificial Compressibility Method. *JOURNAL OF COMPUTATIONAL PHYSICS*, 231:5109–5143, 2012. (Cited on pages 22 and 48.)

[22] Zhaoli Guo and T. S. Zhao. Lattice Boltzmann model for incompressible flows through porous media. *Phys. Rev. E*, 66:036304, Sep 2002. (Cited on pages 22 and 64.)

[23] Andrew K. Gunstensen and Daniel H. Rothman. Lattice-Boltzmann studies of immiscible two-phase flow through porous media. *Journal of Geophysical Research: Solid Earth*, 98(B4):6431–6441, 1993. (Cited on pages 22 and 64.)

[24] S. Hou, J. Sterling, S. Chen, and G.D. Doolen. *A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows*, volume 6 of *Fields Institute Communications*, pages 151–166. AMS, Providence, 1996. (Cited on pages 22, 130, and 193.)

[25] Orestis Malaspinas and Pierre Sagaut. Advanced large-eddy simulation for lattice Boltzmann methods: The approximate deconvolution model. *Physics of Fluids*, 23(10):105103, 2011. (Cited on pages 22, 52, and 193.)

[26] Sauro Succi, Giorgio Amati, and Roberto Benzi. Challenges in lattice Boltzmann computing. *Journal of Statistical Physics*, 81(1-2):5–16, 1995. (Cited on pages 22, 52, and 193.)

[27] Yan Peng, Wei Liao, Li-Shi Luo, and Lian-Ping Wang. Comparison of the lattice Boltzmann and pseudo-spectral methods for decaying turbulence: Low-order statistics . *Computers & Fluids*, 39(4):568 – 591, 2010. (Cited on pages 22 and 193.)

[28] Leila Jahanshaloo, Emad Pouryazdanpanah, and Nor Azwadi Che Sidik. A Review on the Application of the Lattice Boltzmann Method for Turbulent Flow Simulation. *Numerical Heat Transfer, Part A: Applications*, 64(11):938–953, 2013. (Cited on pages 22 and 193.)

[29] Li Chen, Qinjun Kang, Yutong Mu, Ya-Ling He, and Wen-Quan Tao. A critical review of the pseudopotential multiphase lattice Boltzmann model: Methods and applications . *International Journal of Heat and Mass Transfer*, 76:210 – 236, 2014. (Cited on pages 22, 35, and 39.)

[30] Zhaoli Guo, Baochang Shi, and Chuguang Zheng. A coupled lattice BGK model for the Boussinesq equations. *International Journal for Numerical Methods in Fluids*, 39(4):325–342, 2002. (Cited on pages 22, 40, 42, and 193.)

[31] GuyR. McNamara, AlejandroL. Garcia, and BerniJ. Alder. A hydrodynamically correct thermal lattice Boltzmann model. *Journal of Statistical Physics*, 87(5-6):1111–1121, 1997. (Cited on pages 22, 40, and 193.)

[32] Xiaoyi He, Shiyi Chen, and Gary D. Doolen. A Novel Thermal Model for the Lattice Boltzmann Method in Incompressible Limit . *Journal of Computational Physics*, 146(1):282 – 300, 1998. (Cited on pages 22 and 193.)

[33] Pierre Lallemand and Li-Shi Luo. Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions. *Phys. Rev. E*, 68:036706, Sep 2003. (Cited on pages 22 and 193.)

[34] Paul J Dellar. Lattice and discrete Boltzmann equations for fully compressible flow. *Computational Fluid and Solid Mechanics*, pages 632–635, 2005. (Cited on page 22.)

[35] Jens Zudrop, Sabine Roller, and Pietro Asinari. Lattice Boltzmann scheme for electrolytes by an extended Maxwell-Stefan approach. *Phys. Rev. E*, 89:053310, May 2014. (Cited on page 22.)

[36] Zhaoli Guo, T. S. Zhao, and Yong Shi. A lattice Boltzmann algorithm for electro-osmotic flows in microfluidic devices. *The Journal of Chemical Physics*, 122(14):144907, 2005. (Cited on page 22.)

[37] P. J. Dellar. Lattice kinetic schemes for magnetohydrodynamics. *J. Comput. Phys.*, 179:95–126, 2002. (Cited on page 22.)

[38] Einat Aharonov and Daniel H. Rothman. Non-Newtonian flow (through porous media): A lattice-Boltzmann method. *Geophysical Research Letters*, 20(8):679–682, 1993. (Cited on pages 22 and 64.)

[39] Burkhard Dunweg and Anthony J.C. Ladd. Lattice Boltzmann Simulations of Soft Matter Systems. Advances in Polymer Science, pages 1–78. Springer Berlin Heidelberg, 2008. (Cited on page 22.)

[40] Jian Guo Zhou. *Lattice Boltzmann methods for shallow water flows*, volume 4. Springer, 2004. (Cited on page 22.)

[41] S. Succi, G. Bella, and F. Papetti. Lattice Kinetic Theory for Numerical Combustion. *Journal of Scientific Computing*, 12(4):395–408, 1997. (Cited on page 22.)

[42] Pietro Asinari, Subhash C. Mishra, and Romano Borchiellini. A Lattice Boltzmann Formulation for the Analysis of Radiative Heat Transfer Problems in a Participating Medium. *Numerical Heat Transfer, Part B: Fundamentals*, 57(2):126–146, 2010. (Cited on page 22.)

[43] M. Mendoza, B. M. Boghosian, H. J. Herrmann, and S. Succi. Fast Lattice Boltzmann Solver for Relativistic Hydrodynamics. *Phys. Rev. Lett.*, 105:014502, Jun 2010. (Cited on page 22.)

[44] J. Tölke and M. Krafczyk. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *International Journal of Computational Fluid Dynamics*, 22(7):443–456, 2008. (Cited on pages 22, 74, and 89.)

[45] Olga Filippova and Dieter Hanel. Grid refinement for lattice-bgk models. *Journal of Computational Physics*, 147(1):219 – 228, 1998. (Cited on pages 22 and 54.)

[46] N. Delbosc, J.L. Summers, A.I. Khan, N. Kapur, and C.J. Noakes. Optimized implementation of the lattice boltzmann method on a graphics processing unit towards real-time fluid simulation. *Computers & Mathematics with Applications*, 67(2):462 – 475, 2014. (Cited on pages 22 and 155.)

[47] Sauro Succi. Lattice Boltzmann 2038. *EPL (Europhysics Letters)*, 109(5): 50001, 2015. (Cited on page 22.)

[48] Philip Gressman and Robert Strain. Global classical solutions of the Boltzmann equation without angular cut-off. *Journal of the American Mathematical Society*, 24(3):771–847, 2011. (Cited on page 23.)

[49] Prabhu Lal Bhatnagar, Eugene P Gross, and Max Krook. A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. *Physical review*, 94(3):511, 1954. (Cited on page 24.)

[50] J. Wu and C. Shu. A solution-adaptive lattice Boltzmann method for two-dimensional incompressible viscous flows. *Journal of Computational Physics*, 230(6):2246–2269, 2011. (Cited on page 25.)

[51] Dominique d'Humieres. Generalized Lattice-Boltzmann Equations. *Rarefied Gas Dynamics: Theory and Simulations - Progress in Astronautics and Aeronautics*, pages 450–458, 1992. (Cited on page 25.)

[52] Dominique d'Humières and Irina Ginzburg. Viscosity independent numerical errors for Lattice Boltzmann models: from recurrence equations to "magic" collision numbers. *Computers & Mathematics with Applications*, 58(5):823–840, 2009. (Cited on page 26.)

[53] Irina Ginzburg. *Une variation sur les proprietes magiques de modeles de Boltzmann pour l'ecoulement microscopique et macroscopique*. PhD thesis, Universite Pierre at Marie Curie, Paris, France, 2009. (Cited on pages 26 and 188.)

[54] Ilya Karlin, Pietro Asinari, and Sauro Succi. Matrix lattice Boltzmann reloaded. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 369(1944):2202–2210, 2011. (Cited on page 26.)

[55] YH Qian, Dominique d'Humières, and Pierre Lallemand. Lattice BGK models for Navier-Stokes equation. *EPL (Europhysics Letters)*, 17(6):479, 1992. (Cited on page 30.)

[56] Sauro Succi. *The lattice Boltzmann equation: for fluid dynamics and beyond*. Oxford university press, 2001. (Cited on page 31.)

[57] Zhaoli Guo, Baochang Shi, and Nengchao Wang. Lattice BGK Model for Incompressible Navier-Stokes Equation . *Journal of Computational Physics*, 165(1):288–306, 2000. (Cited on page 32.)

[58] He Nan-Zhong, Wang Neng-Chao, Shi Bao-Chang, and Guo Zhao-Li. A unified incompressible lattice BGK model and its application to three-dimensional lid-driven cavity flow. *Chinese Physics*, 13(1):40, 2004. (Cited on page 32.)

[59] Li-Shi Luo, Wei Liao, Xingwang Chen, Yan Peng, and Wei Zhang. Numerics of the lattice Boltzmann method: Effects of collision models on the lattice Boltzmann simulations. *Phys. Rev. E*, 83:056710, May 2011. (Cited on pages 33, 45, 46, 65, 125, 126, and 133.)

[60] Paul J. Dellar. Lattice Boltzmann algorithms without cubic defects in Galilean invariance on standard lattices . *Journal of Computational Physics*, 259:270 – 283, 2014. (Cited on pages 33 and 47.)

[61] Y.H. Qian. Simulating thermohydrodynamics with lattice BGK models. *Journal of Scientific Computing*, 8(3):231–242, 1993. (Cited on page 33.)

[62] Frank J Alexander, Shiyi Chen, and JD Sterling. Lattice boltzmann thermohydrodynamics. *Physical Review E*, 47(4):R2249, 1993. (Cited on pages 33 and 40.)

[63] Feng Chen, Aiguo Xu, Guangcai Zhang, Yingjun Li, and Sauro Succi. Multiple-relaxation-time lattice Boltzmann approach to compressible flows with flexible specific-heat ratio and Prandtl number. *EPL (Europhysics Letters)*, 90(5):54003, 2010. (Cited on page 33.)

[64] Feng Chen, Aiguo Xu, Guangcai Zhang, and Yingjun Li. Multiple-relaxation-time lattice Boltzmann model for compressible fluids. *Physics Letters A*, 375(21):2129–2139, 2011. (Cited on page 33.)

[65] Qing Li, YL He, Yong Wang, and WQ Tao. Coupled double-distribution-function lattice boltzmann method for the compressible navier-stokes equations. *Physical Review E*, 76(5):056705, 2007. (Cited on page 33.)

[66] K Qu, C Shu, and YT Chew. Alternative method to construct equilibrium distribution functions in lattice-Boltzmann method simulation of inviscid compressible flows at high Mach number. *Physical Review E*, 75(3):036706, 2007. (Cited on page 33.)

[67] Chenghai Sun. Lattice-Boltzmann models for high speed flows. *Physical review E*, 58(6):7283, 1998. (Cited on page 33.)

[68] Paul J Dellar. Two routes from the Boltzmann equation to compressible flow of polyatomic gases. *Progress in Computational Fluid Dynamics, an International Journal*, 8(1-4):84–96, 2008. (Cited on page 33.)

[69] Guy R McNamara, Alejandro L Garcia, and Berni J Alder. Stabilization of thermal lattice Boltzmann models. *Journal of Statistical Physics*, 81(1-2):395–408, 1995. (Cited on page 33.)

[70] Takeshi Kataoka and Michihisa Tsutahara. Lattice Boltzmann model for the compressible Navier-Stokes equations with flexible specific-heat ratio. *Physical review E*, 69(3):035701, 2004. (Cited on page 33.)

[71] Kun Qu, Chang Shu, and Yong Tian Chew. Simulation of shock-wave propagation with finite volume lattice Boltzmann method. *International Journal of Modern Physics C*, 18(04):447–454, 2007. (Cited on page 33.)

[72] QU KUN. *Development of lattice Boltzmann method for compressible flows*. PhD thesis, Northwestern Polytechnical University, China, 2009. (Cited on page 33.)

[73] J. R. Castrejón-Pita, K. J. Kubiak, A. A. Castrejón-Pita, M. C. T. Wilson, and I. M. Hutchings. Mixing and internal dynamics of droplets impacting and coalescing on a solid surface. *Phys. Rev. E*, 88:023023, Aug 2013. (Cited on pages 34 and 39.)

[74] G. Falcucci, S. Chibbaro, S. Succi, X. Shan, and H. Chen. Lattice Boltzmann spray-like fluids. *EPL (Europhysics Letters)*, 82(2):24005, 2008. (Cited on pages 34 and 39.)

[75] Jonas Tölke. Lattice Boltzmann methods for digital rock physics. 23rd International Conference on Discrete Simulation of Fluid Dynamics, 2014. (Cited on page 34.)

[76] Maclean O Amabeoku, Tariq M Al-Ghamdi, Yaoming Mu, R Ingrain, and Jonas Tölke. Evaluation and Application of Digital Rock Physics (DRP) for Special Core Analysis in Carbonate Formations. In *IPTC 2013: International Petroleum Technology Conference*, 2013. (Cited on pages 34 and 39.)

[77] Junfeng Zhang. Lattice Boltzmann method for microfluidics: models and applications. *Microfluidics and Nanofluidics*, 10(1):1–28, 2011. (Cited on pages 34 and 39.)

[78] Andrew K. Gunstensen, Daniel H. Rothman, Stéphane Zaleski, and Gianluigi Zanetti. Lattice Boltzmann model of immiscible fluids. *Phys. Rev. A*, 43:4320–4327, Apr 1991. (Cited on page 34.)

[79] DanielH. Rothman and JeffreyM. Keller. Immiscible cellular-automaton fluids. *Journal of Statistical Physics*, 52(3-4):1119–1127, 1988. (Cited on page 34.)

[80] Francis J Alexander, Shiyi Chen, and Daryl W Grunau. Hydrodynamic spinodal decomposition: Growth kinetics and scaling functions. *Physical Review B*, 48(1):634, 1993. (Cited on page 35.)

[81] Jonas Tölke. Lattice Boltzmann simulations of binary fluid flow through porous media. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 360(1792): 535–545, 2002. (Cited on pages 35 and 64.)

[82] Daryl Grunau, Shiyi Chen, and Kenneth Eggert. A lattice Boltzmann model for multiphase fluid flows. *Physics of Fluids A: Fluid Dynamics (1989-1993)*, 5(10):2557–2562, 1993. (Cited on page 35.)

[83] U d'Ortona, D Salin, Marek Cieplak, Renata B Rybka, and Jayanth R Banavar. Two-color nonlinear Boltzmann cellular automata: surface tension and wetting. *Physical Review E*, 51(4):3718, 1995. (Cited on page 35.)

[84] EG Flekkøy. Lattice Bhatnagar-Gross-Krook models for miscible fluids. *Physical Review E*, 47(6):4247, 1993. (Cited on page 35.)

[85] Xiaowen Shan and Hudong Chen. Lattice Boltzmann model for simulating flows with multiple phases and components. *Physical Review E*, 47(3):1815, 1993. (Cited on pages 35, 36, and 179.)

[86] Xiaowen Shan and Gary Doolen. Multicomponent lattice-Boltzmann model with interparticle interaction. *Journal of Statistical Physics*, 81 (1-2):379–393, 1995. (Cited on page 35.)

[87] Xiaoyi He, Shiyi Chen, and Raoyang Zhang. A lattice boltzmann scheme for incompressible multiphase flow and its application in simulation of rayleigh–taylor instability. *Journal of Computational Physics*, 152(2):642–663, 1999. (Cited on pages 35, 37, and 177.)

[88] Haibo Huang, Daniel T. Thorne, Marcel G. Schaap, and Michael C. Sukop. Proposed approximation for contact angles in Shan-and-Chen-type multicomponent multiphase lattice Boltzmann models. *Phys. Rev. E*, 76:066701, Dec 2007. (Cited on pages 35 and 179.)

[89] Xiaowen Shan. Pressure tensor calculation in a class of nonideal gas lattice Boltzmann models. *Phys. Rev. E*, 77:066702, Jun 2008. doi: 10.1103/PhysRevE.77.066702. URL `http://link.aps.org/doi/10.1103/PhysRevE.77.066702`. (Cited on page 35.)

[90] Daniel Lycett-Brown and Kai H. Luo. Improved forcing scheme in pseudopotential lattice Boltzmann methods for multiphase flow at arbitrarily high density ratios. *Phys. Rev. E*, 91:023305, Feb 2015. (Cited on page 36.)

[91] Peng Yuan and Laura Schaefer. Equations of state in a lattice Boltzmann model. *Physics of Fluids (1994-present)*, 18(4):042101, 2006. (Cited on page 36.)

[92] Michael R. Swift, E. Orlandini, W. R. Osborn, and J. M. Yeomans. Lattice Boltzmann simulations of liquid-gas and binary fluid systems. *Phys. Rev. E*, 54:5041–5052, Nov 1996. (Cited on page 36.)

[93] H Kusumaatmaja, A Dupuis, and Julia M Yeomans. Lattice Boltzmann simulations of drop dynamics. *Mathematics and Computers in Simulation*, 72(2):160–164, 2006. (Cited on pages 36 and 39.)

[94] Norman F Carnahan and Kenneth E Starling. Equation of state for nonattracting rigid spheres. *The Journal of Chemical Physics*, 51(2):635–636, 1969. (Cited on pages 38 and 177.)

[95] Irina Ginzburg and Konrad Steiner. Lattice Boltzmann model for free-surface flow and its application to filling process in casting. *Journal of Computational Physics*, 185(1):61–99, 2003. (Cited on page 39.)

[96] Cyril W Hirt and Billy D Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1): 201–225, 1981. (Cited on page 39.)

[97] U Rüde and N Thürey. Free surface lattice-Boltzmann fluid simulations with and without level sets. In *Vision, Modeling, and Visualization 2004: Proceedings, November 16-18, 2004, Standford, USA*, page 199. IOS Press, 2004. (Cited on page 39.)

[98] Mark Sussman, Peter Smereka, and Stanley Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational physics*, 114(1):146–159, 1994. (Cited on page 39.)

[99] Nils Thurey. *Physically based Animation of Free Surface Flows with the Lattice Boltzmann Method* . PhD thesis, 2007. (Cited on page 39.)

[100] Christian F Janßen, Dennis Mierke, Micha Überrück, Silke Gralher, and Thomas Rung. Validation of the GPU-Accelerated CFD Solver ELBE for Free Surface Flow Problems in Civil and Environmental Engineering. *Computation*, 3(3):354–385, 2015. (Cited on page 39.)

[101] Shuling Hou, Xiaowen Shan, Qisu Zou, Gary D. Doolen, and Wendy E. Soll. Evaluation of Two Lattice Boltzmann Models for Multiphase Flows . *Journal of Computational Physics*, 138(2):695 – 713, 1997. (Cited on page 39.)

[102] Xiaowen Shan. Analysis and reduction of the spurious current in a class of multiphase lattice Boltzmann models. *Phys. Rev. E*, 73:047701, Apr 2006. (Cited on page 39.)

[103] Kevin Connington and Taehun Lee. A review of spurious currents in the lattice Boltzmann method for multiphase flows. *Journal of mechanical science and technology*, 26(12):3857–3863, 2012. (Cited on page 39.)

[104] Chris Teixeira, Hudong Chen, and David M. Freed. Multi-speed thermal lattice Boltzmann method stabilization via equilibrium under-relaxation . *Computer Physics Communications*, 129(1-3):207–226, 2000. (Cited on page 40.)

[105] Y Chen, H Ohashi, and M Akiyama. Thermal lattice Bhatnagar-Gross-Krook model without nonlinear deviations in macrodynamic equations. *Physical Review E*, 50(4):2776, 1994. (Cited on page 40.)

[106] Y Chen, H Ohashi, and M Akiyama. Two-parameter thermal lattice BGK model with a controllable Prandtl number. *Journal of scientific computing*, 12(2):169–185, 1997. (Cited on page 40.)

[107] A Bartoloni, C Battista, S Cabasino, PS Paolucci, J Pech, R Sarno, GM Todesco, M Torelli, W Tross, P Vicini, et al. LBE simulations of Rayleigh-Benard convection on the APE100 parallel processor. *International Journal of Modern Physics C*, 4(05):993–1006, 1993. (Cited on page 40.)

[108] Xiaowen Shan. Simulation of rayleigh-bénard convection using a lattice boltzmann method. *Physical Review E*, 55(3):2780, 1997. (Cited on page 40.)

[109] Bruce J Palmer and David R Rector. Lattice Boltzmann algorithm for simulating thermal flow in compressible fluids. *Journal of Computational Physics*, 161(1):1–20, 2000. (Cited on page 40.)

[110] Du Hong-Yan, Chai Zhen-Hua, and Shi Bao-Chang. Lattice Boltzmann study of mixed convection in a cubic cavity. *Communications in Theoretical Physics*, 56(1):144, 2011. (Cited on page 41.)

[111] Zhaoli Guo, Chuguang Zheng, and Baochang Shi. Discrete lattice effects on the forcing term in the lattice Boltzmann method. *Physical Review E*, 65(4):046308, 2002. (Cited on pages 42 and 150.)

[112] Olga Filippova and Dieter Hänel. A novel lattice BGK approach for low Mach number combustion. *Journal of Computational Physics*, 158 (2):139–160, 2000. (Cited on page 42.)

[113] Ahmed Mezrhab, M'hamed Bouzidi, and Pierre Lallemand. Hybrid lattice-Boltzmann finite-difference simulation of convective flows. *Computers & Fluids*, 33(4):623–641, 2004. (Cited on page 42.)

[114] Frederik Verhaeghe, Bart Blanpain, and Patrick Wollants. Lattice Boltzmann method for double-diffusive natural convection. *Physical Review E*, 75(4):046705, 2007. (Cited on page 42.)

[115] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. The TheLMA project: A thermal lattice Boltzmann solver for the GPU . *Computers & Fluids*, 54:118 – 126, 2012. (Cited on page 42.)

[116] Charles S Peskin. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25(3):220 – 252, 1977. (Cited on pages 43 and 70.)

[117] Zhi-Gang Feng and Efstathios E Michaelides. The immersed boundary-lattice boltzmann method for solving fluid-particles interaction problems. *Journal of Computational Physics*, 195(2):602–628, 2004. (Cited on page 43.)

[118] X.D. Niu, C. Shu, Y.T. Chew, and Y. Peng. A momentum exchange-based immersed boundary-lattice boltzmann method for simulating incompressible viscous flows. *Physics Letters A*, 354(3):173 – 182, 2006. (Cited on page 43.)

[119] D. R. J. Owen, C. R. Leonardi, and Y. T. Feng. An efficient framework for fluid-structure interaction using the lattice Boltzmann method and immersed moving boundaries. *International Journal for Numerical Methods in Engineering*, 87(1-5):66–95, 2011. (Cited on pages 43, 44, 70, 71, and 184.)

[120] Daniel A. Reasor, Jonathan R. Clausen, and Cyrus K. Aidun. Coupling the lattice-Boltzmann and spectrin-link methods for the direct numerical simulation of cellular blood flow. *International Journal for Numerical Methods in Fluids*, 68(6):767–781, 2012. (Cited on page 44.)

[121] Xiaoyi He and Gary Doolen. Lattice Boltzmann Method on Curvilinear Coordinates System: Flow around a Circular Cylinder. *Journal of Computational Physics*, 134(2):306 – 315, 1997. (Cited on pages 44 and 54.)

[122] Anthony J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. *Journal of Fluid Mechanics*, 271:285–309, 7 1994. (Cited on page 44.)

[123] Anthony J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 2. Numerical results. *Journal of Fluid Mechanics*, 271:311–339, 7 1994. (Cited on page 44.)

[124] Renwei Mei, Dazhi Yu, Wei Shyy, and Li-Shi Luo. Force evaluation in the lattice Boltzmann method involving curved geometry. *Phys. Rev. E*, 65:041203, Apr 2002. (Cited on pages 44 and 45.)

[125] Pierre Lallemand and Li-Shi Luo. Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. *Phys. Rev. E*, 61:6546–6562, Jun 2000. (Cited on pages 45, 61, and 95.)

[126] Ludwig Boltzmann. Weitere Studien ÃŒber das WÃ€rmegleichgewicht unter GasmolekÃŒlen. In *Kinetische Theorie II*, volume 67 of *WTB Wissenschaftliche TaschenbÃŒcher*, pages 115–225. Vieweg+Teubner Verlag, 1970. (Cited on page 45.)

[127] Hudong Chen and Chris Teixeira. H-theorem and origins of instability in thermal lattice Boltzmann models . *Computer Physics Communications*, 129(1-3):21–31, 2000. (Cited on page 45.)

[128] Santosh Ansumali and Iliya V. Karlin. Stabilization of the lattice Boltzmann method by the H-theorem: A numerical test. *Phys. Rev. E*, 62: 7999–8003, Dec 2000. (Cited on pages 45 and 46.)

[129] Orestis Malaspinas, Michel Deville, and Bastien Chopard. Towards a physical interpretation of the entropic lattice Boltzmann method. *Phys. Rev. E*, 78:066705, Dec 2008. (Cited on page 45.)

[130] G. Vahala, B. Keating, M. Soe, J. Yepez, L. Vahala, and S. Ziegeler. Entropic, LES and boundary conditions in lattice Boltzmann simulations of turbulence. *The European Physical Journal Special Topics*, 171(1): 167–171, 2009. (Cited on page 45.)

[131] Tjalling J Ypma. Historical development of the Newton-Raphson method. *SIAM review*, 37(4):531–551, 1995. (Cited on page 46.)

[132] Francesca Tosi, Stefano Ubertini, S. Succi, and I.V. Karlin. Optimization strategies for the entropic lattice boltzmann method. *Journal of Scientific Computing*, 30(3):369–387, 2007. (Cited on page 46.)

[133] T. Yasuda and N. Satofuka. An improved entropic lattice Boltzmann model for parallel computation. *Computers & Fluids*, 45(1):187–190, 2011. 22nd International Conference on Parallel Computational Fluid Dynamics (ParCFD 2010) ParCFD. (Cited on page 46.)

[134] I. V. Karlin, S. Succi, and S. S. Chikatamarla. Comment on "numerics of the lattice boltzmann method: Effects of collision models on the lattice boltzmann simulations". *Phys. Rev. E*, 84:068701, Dec 2011. (Cited on page 46.)

[135] Li-Shi Luo. Reply to "comment on 'numerics of the lattice boltzmann method: Effects of collision models on the lattice boltzmann simulations"'. *Phys. Rev. E*, 86:048701, Oct 2012. (Cited on page 46.)

[136] S.S. Chikatamarla and I.V. Karlin. Entropic lattice boltzmann method for turbulent flow simulations: Boundary conditions. *Physica A: Statistical Mechanics and its Applications*, 392(9):1925 – 1930, 2013. (Cited on page 46.)

[137] I. V. Karlin, F. Bosch, and S. S. Chikatamarla. Gibbs' principle for the lattice-kinetic theory of fluid dynamics. *Phys. Rev. E*, 90:031302, Sep 2014. (Cited on page 46.)

[138] Shyam S. Chikatamarla Fabian Bosch and Ilya Karlin. Entropic multi-relaxation models for simulation of fluid turbulence. *ESAIM: Proceedings and surveys.*, ?:1–10, 2015. (Cited on page 46.)

[139] A. Mazloomi M, S. S. Chikatamarla, and I. V. Karlin. Entropic Lattice Boltzmann Method for Multiphase Flows. *Phys. Rev. Lett.*, 114:174502, May 2015. (Cited on page 46.)

[140] N. Frapolli, S. S. Chikatamarla, and I. V. Karlin. Multispeed entropic lattice Boltzmann model for thermal flows. *Phys. Rev. E*, 90:043306, Oct 2014. (Cited on page 46.)

[141] M Geier. *Ab initio derivation of the cascaded Lattice Boltzmann Automaton*. PhD thesis, University of Freiburg–IMTEK, 2006. (Cited on page 47.)

[142] XB Nie, Xiaowen Shan, and H Chen. Galilean invariance of lattice Boltzmann models. *EPL (Europhysics Letters)*, 81(3):34005, 2008. (Cited on page 47.)

[143] Daniel Lycett-Brown and Kai H. Luo. Multiphase cascaded lattice boltzmann method. *Computers & Mathematics with Applications*, 67(2): 350 – 362, 2014. (Cited on pages 47 and 187.)

[144] Daniel Lycett-Brown, Kai H. Luo, Ronghou Liu, and Pengmei Lv. Binary droplet collision simulations by a multiphase cascaded lattice boltzmann method. *Physics of Fluids*, 26(2):023303, 2014. (Cited on page 47.)

[145] Martin Geier, Martin Schönherr, Andrea Pasquali, and Manfred Krafczyk. The cumulant lattice boltzmann equation in three dimensions: Theory and validation. *Computers & Mathematics with Applications*, (0):–, 2015. (Cited on pages 47 and 48.)

[146] S. Geller, S. Uphoff, and M. Krafczyk. Turbulent jet computations based on MRT and Cascaded Lattice Boltzmann models. *Computers & Mathematics with Applications*, 65(12):1956 – 1966, 2013. (Cited on pages 48 and 53.)

[147] M. Geier, A. Greiner, and J. G. Korvink. A factorized central moment lattice boltzmann method. *The European Physical Journal Special Topics*, 171(1):55–61, 2009. (Cited on page 48.)

[148] KannanN. Premnath and Sanjoy Banerjee. On the three-dimensional central moment lattice boltzmann method. *Journal of Statistical Physics*, 143(4):747–794, 2011. (Cited on page 48.)

[149] Ronald A Fisher and John Wishart. The derivation of the pattern formulae of two-way partitions from those of simpler patterns. *Proceedings of the London Mathematical Society*, 2(1):195–208, 1932. (Cited on page 48.)

[150] Pietro Asinari. Generalized local equilibrium in the cascaded lattice Boltzmann method. *Phys. Rev. E*, 78:016701, Jul 2008. (Cited on page 48.)

[151] Alexandre Joel Chorin. A Numerical Method for Solving Incompressible Viscous Flow Problems. *Journal of Computational Physics*, 2(1):12 – 26, 1967. (Cited on pages 48, 203, and 204.)

[152] Takaji Inamuro. A lattice kinetic scheme for incompressible viscous flows with heat transfer. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 360(1792):477–484, 2002. (Cited on page 49.)

[153] Christian Obrecht, Pietro Asinari, Frederic Kuznik, and Jean-Jacques Roux. High-performance implementations and large-scale validation of the link-wise artificial compressibility method. *Journal of Computational Physics*, 275(0):143 – 153, 2014. (Cited on page 49.)

[154] Christian Obrecht, Pietro Asinari, Frederic Kuznik, and Jean-Jacques Roux. Thermal link-wise artificial compressibility method: GPU implementation and validation of a double-population model. *Computers & Mathematics with Applications*, (0):–, 2015. (Cited on page 49.)

[155] Jonas Latt and Bastien Chopard. Lattice boltzmann method with regularized pre-collision distribution functions. *Mathematics and Computers in Simulation*, 72(2-6):165–168, 2006. Discrete Simulation of Fluid Dynamics in Complex Systems. (Cited on page 49.)

[156] Jonas Latt. *Hydrodynamic limit of lattice Boltzmann equations*. PhD thesis, University of Geneva, 2007. (Cited on pages 49 and 72.)

[157] Sergei Konstantinovich Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*, 89(3):271–306, 1959. (Cited on page 50.)

[158] RA Brownlee, Alexander N Gorban, and Jeremy Levesley. Stabilization of the lattice Boltzmann method using the Ehrenfests coarse-graining idea. *Physical Review E*, 74(3):037703, 2006. (Cited on page 50.)

[159] RA Brownlee, Alexander N Gorban, and Jeremy Levesley. Stability and stabilization of the lattice Boltzmann method. *Physical Review E*, 75(3):036711, 2007. (Cited on page 50.)

[160] RA Brownlee, Alexander N Gorban, and Jeremy Levesley. Nonequilibrium entropy limiters in lattice Boltzmann methods. *Physica A: Statistical Mechanics and its Applications*, 387(2):385–406, 2008. (Cited on page 50.)

[161] Panos Tamamidis and Dennis N Assanis. Evaluation of various high-order-accuracy schemes with and without flux limiters. *International Journal for Numerical Methods in Fluids*, 16(10):931–948, 1993. (Cited on page 50.)

[162] Robert A Brownlee, Jeremy Levesley, David Packwood, and Alexander N Gorban. Add-ons for lattice Boltzmann methods: regularization, filtering and limiters. *Progress in Computational Physics*, 3:31–52, 2013. (Cited on page 50.)

[163] Denis Ricot, Simon Marié, Pierre Sagaut, and Christophe Bailly. Lattice Boltzmann method with selective viscosity filter. *Journal of Computational Physics*, 228(12):4478–4490, 2009. (Cited on page 50.)

[164] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, 1963. (Cited on page 51.)

[165] S. Hou, J. Sterling, S. Chen, and G.D. Doolen. *A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows*, volume 6 of *Fields Institute Communications*, pages 151–166. AMS, Providence, 1996. (Cited on pages 51 and 53.)

[166] Huidan Yu, Sharath S. Girimaji, and Li-Shi Luo. DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method. *Journal of Computational Physics*, 209(2):599 – 616, 2005. (Cited on pages 51 and 53.)

[167] A.M.O. Smith and Tuncer Cebeci. Numerical solution of the turbulent-boundary-layer equations. Technical report, DTIC Document, 1967. (Cited on page 52.)

[168] Barrett Stone Baldwin and Harvard Lomax. *Thin layer approximation and algebraic model for separated turbulent flows*, volume 257. American Institute of Aeronautics and Astronautics, 1978. (Cited on page 52.)

[169] Dennis A Johnson and LS King. A mathematically simple turbulence closure model for attached and separated turbulent boundary layers. *AIAA journal*, 23(11):1684–1692, 1985. (Cited on page 52.)

[170] Barrett Stone Baldwin and Timothy J Barth. *A one-equation turbulence transport model for high Reynolds number wall-bounded flows*. National Aeronautics and Space Administration, Ames Research Center, 1990. (Cited on page 52.)

[171] Philipe R Spalart and Stephen R Allmaras. A one-equation turbulence model for aerodynamic flows. *American Institute of Aeronautics and Astronautics*, 1992. (Cited on page 52.)

[172] WP Jones and BEi Launder. The prediction of laminarization with a two-equation model of turbulence. *International journal of heat and mass transfer*, 15(2):301–314, 1972. (Cited on page 52.)

[173] VSASTBCG Yakhot, SA Orszag, S Thangam, TB Gatski, and CG Speziale. Development of turbulence models for shear flows by a double expansion technique. *Physics of Fluids A: Fluid Dynamics (1989-1993)*, 4(7):1510–1520, 1992. (Cited on page 52.)

[174] David C Wilcox et al. *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA, 1998. (Cited on page 52.)

[175] Florian R Menter. Zonal two equation k-turbulence models for aerodynamic flows. *AIAA paper*, 2906:1993, 1993. (Cited on page 52.)

[176] Christopher M. Teixeira. Incorporating turbulence models into the lattice-Boltzmann method. *Int. J. Mod. Phys. C*, 9:1159–1175, 1998. (Cited on pages 52 and 72.)

[177] C. Shu, Y. Peng, C. F. Zhou, and Y. T. Chew. Application of Taylor series expansion and Least-squares-based lattice Boltzmann method to simulate turbulent flows. *Journal of Turbulence*, 7:N38, 2006. (Cited on page 52.)

[178] Xiao-Peng Chen. Applications of Lattice Boltzmann Method to Turbulent Flow Around Two-Dimensional Airfoil. *Engineering Applications of Computational Fluid Mechanics*, 6(4):572–580, 2012. (Cited on pages 52 and 53.)

[179] Kai Li, Chengwen Zhong, Congshan Zhuo, and Jun Cao. Non-body-fitted Cartesian-mesh simulation of highly turbulent flows using multi-relaxation-time lattice Boltzmann method . *Computers & Mathematics with Applications*, 63(10):1481 – 1496, 2012. (Cited on page 52.)

[180] Kannan N. Premnath, Martin J. Pattison, and Sanjoy Banerjee. Dynamic subgrid scale modeling of turbulent flows using lattice-boltzmann method. *Physica A: Statistical Mechanics and its Applications*, 388(13):2640 – 2658, 2009. (Cited on pages 52 and 53.)

[181] Kannan N. Premnath, Martin J. Pattison, and Sanjoy Banerjee. Generalized lattice boltzmann equation with forcing term for computation of wall-bounded turbulent flows. *Phys. Rev. E*, 79:026703, Feb 2009. (Cited on pages 52 and 53.)

[182] Pierre Sagaut. Toward advanced subgrid models for lattice-boltzmann-based large-eddy simulation: Theoretical formulations. *Computers & Mathematics with Applications*, 59(7):2194 – 2199, 2010. Mesoscopic Methods in Engineering and Science International Conferences on Mesoscopic Methods in Engineering and Science. (Cited on page 52.)

[183] Orestis Malaspinas and Pierre Sagaut. Consistent subgrid scale modelling for lattice boltzmann methods. *Journal of Fluid Mechanics*, 700: 514–542, 6 2012. (Cited on page 52.)

[184] S. Stolz and N. A. Adams. An approximate deconvolution procedure for large-eddy simulation. *Physics of Fluids*, 11(7):1699–1701, 1999. (Cited on page 52.)

[185] Huidan Yu, Li-Shi Luo, and Sharath S. Girimaji. LES of turbulent square jet flow using an MRT lattice Boltzmann model. *Computers & Fluids*, 35(8-9):957–965, 2006. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science. (Cited on page 53.)

[186] Shin K. Kang and Yassin A. Hassan. The effect of lattice models within the lattice boltzmann method in the simulation of wall-bounded turbulent flows. *Journal of Computational Physics*, 232(1):100 – 117, 2013. (Cited on page 53.)

[187] B Crouse, M Krafczyk, S KÃŒhner, E Rank, and C van Treeck. Indoor air flow analysis based on lattice boltzmann methods. *Energy and Buildings*, 34(9):941–949, 2002. A View of Energy and Bilding Performance Simulation at the start of the third millennium. (Cited on page 53.)

[188] M. Weickert, G. Teike, O. Schmidt, and M. Sommerfeld. Investigation of the LES WALE turbulence model within the lattice Boltzmann framework. *Computers & Mathematics with Applications*, 59(7):2200 – 2214, 2010. Mesoscopic Methods in Engineering and Science International Conferences on Mesoscopic Methods in Engineering and Science. (Cited on pages 53 and 72.)

[189] Sheng Chen. A large-eddy-based lattice Boltzmann model for turbulent flow simulation. *Applied Mathematics and Computation*, 215(2): 591–598, 2009. (Cited on pages 53 and 130.)

[190] L.S. Luo, M. Krafczyk, and J. Tölke. Large-eddy simulations with a multiple-relaxation-time LBE model. *International Journal of Modern Physics B*, 17:33–39, 2003. (Cited on page 53.)

[191] Xiaopei Liu, Wai-Man Pang, Jing Qin, and Chi-Wing Fu. Turbulence simulation by adaptive multi-relaxation lattice boltzmann modeling. *Visualization and Computer Graphics, IEEE Transactions on*, 20(2):289–302, Feb 2014. (Cited on page 53.)

[192] JMVA Koelman. A simple lattice boltzmann scheme for navier-stokes fluid flow. *EPL (Europhysics Letters)*, 15(6):603, 1991. (Cited on page 54.)

[193] Jian Guo Zhou. Mrt rectangular lattice boltzmann method. *International Journal of Modern Physics C*, 23(05):1250040, 2012. (Cited on page 54.)

[194] Yuan Zong, Cheng Peng, Zhaoli Guo, and Lian-Ping Wang. Designing correct fluid hydrodynamics on a rectangular grid using MRT lattice Boltzmann approach . *Computers & Mathematics with Applications*, pages –, 2015. (Cited on page 54.)

[195] Alexandre Dupuis and Bastien Chopard. Theory and applications of an alternative lattice Boltzmann grid refinement algorithm. *Physical Review E*, 67(6):066707, 2003. (Cited on page 54.)

[196] Dazhi Yu, Renwei Mei, and Wei Shyy. A multi-block lattice Boltzmann method for viscous fluid flows. *International journal for numerical methods in fluids*, 39(2):99–120, 2002. (Cited on page 54.)

[197] Daniel Lagrava, Orestis Malaspinas, Jonas Latt, and Bastien Chopard. Advances in multi-domain lattice Boltzmann grid refinement. *Journal of Computational Physics*, 231(14):4808–4822, 2012. (Cited on page 54.)

[198] Yu Chen, Qinjun Kang, Qingdong Cai, and Dongxiao Zhang. Lattice Boltzmann method on quadtree grids. *Physical Review E*, 83(2):026707, 2011. (Cited on page 54.)

[199] Bernd Crouse, Ernst Rank, Manfred Krafczyk, and Jonas Tölke. A LB-based approach for adaptive flow simulations. *International Journal of Modern Physics B*, 17(01n02):109–112, 2003. (Cited on page 54.)

[200] Jonas Tölke, Sören Freudiger, and Manfred Krafczyk. An adaptive scheme using hierarchical grids for lattice boltzmann multi-phase flow simulations. *Computers & fluids*, 35(8):820–830, 2006. (Cited on page 54.)

[201] Xiaoyi He, Li-Shi Luo, and Micah Dembo. Some progress in lattice Boltzmann method. Part I. Nonuniform mesh grids. *Journal of Computational Physics*, 129(2):357–363, 1996. (Cited on page 54.)

[202] Masoud Mirzaei and Amin Poozesh. Simulation of fluid flow in a body-fitted grid system using the lattice Boltzmann method. *Physical Review E*, 87(6):063312, 2013. (Cited on page 54.)

[203] Francesca Nannelli and Sauro Succi. The lattice boltzmann equation on irregular lattices. *Journal of Statistical Physics*, 68(3-4):401–407, 1992. (Cited on page 54.)

[204] Taehun Lee and Ching-Long Lin. A characteristic Galerkin method for discrete Boltzmann equation. *Journal of Computational Physics*, 171 (1):336–356, 2001. (Cited on page 54.)

[205] M'hamed Bouzidi, Dominique d'Humières, Pierre Lallemand, and Li-Shi Luo. Lattice Boltzmann Equation on a Two-dimensional Rectangular Grid. *J. Comput. Phys.*, 172(2):704–717, September 2001. (Cited on page 54.)

[206] Catalin Teodosiu, Raluca Hohota, Gilles Rusaouën, and Monika Woloszyn. Numerical prediction of indoor air humidity and its effect on indoor environment. *Building and Environment*, 38(5):655–664, 2003. (Cited on page 55.)

[207] Richard C Chu, Robert E Simons, Michael J Ellsworth, Roger R Schmidt, and Vincent Cozzolino. Review of cooling technologies for computer products. *Device and Materials Reliability, IEEE Transactions on*, 4(4):568–585, 2004. (Cited on page 55.)

[208] J-I Choi and Jack R Edwards. Large eddy simulation and zonal modeling of human-induced contaminant transport. *Indoor air*, 18(3):233–249, 2008. (Cited on page 55.)

[209] Jonas Latt, Bastien Chopard, Orestis Malaspinas, Michel Deville, and Andreas Michler. Straight velocity boundaries in the lattice Boltzmann method. *Physical Review E*, 77(5):056703, 2008. (Cited on pages 57, 62, and 72.)

[210] Markus Wittmann, Thomas Zeiser, Georg Hager, and Gerhard Wellein. Comparison of different propagation steps for lattice Boltzmann methods . *Computers & Mathematics with Applications*, 65(6):924 – 935, 2013. Mesoscopic Methods in Engineering and Science. (Cited on page 59.)

[211] P. Bailey, J. Myre, S.D.C. Walsh, D.J. Lilja, and M.O. Saar. Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors. In *International Conference on Parallel Processing, ICPP 2009.*, pages 550–557, Sept 2009. (Cited on pages 59 and 89.)

[212] Pierre Lallemand and Li-Shi Luo. Lattice Boltzmann method for moving boundaries. *Journal of Computational Physics*, 184(2):406 – 421, 2003. (Cited on pages 62, 68, 69, 122, and 181.)

[213] Kim Dehee, Kim Hyung Min, S Myung Jhon, J Stephen Vinay III, and Buchanan John. A characteristic non-reflecting boundary treatment in lattice Boltzmann method. *Chinese Physics Letters*, 25(6):1964, 2008. (Cited on pages 62 and 72.)

[214] Joris CG Verschaeve. Analysis of the lattice Boltzmann Bhatnagar-Gross-Krook no-slip boundary condition: Ways to improve accuracy and stability. *Physical Review E*, 80(3):036703, 2009. (Cited on page 62.)

[215] Xiaoyi He, Qisu Zou, Li-Shi Luo, and Micah Dembo. Analytic solutions of simple flows and analysis of nonslip boundary conditions for the lattice Boltzmann BGK model. *Journal of Statistical Physics*, 87(1-2): 115–136, 1997. (Cited on pages 62, 65, and 118.)

[216] Martha A Gallivan, David R Noble, John G Georgiadis, and Richard O Buckius. An evaluation of the bounce-back boundary condition for lattice Boltzmann simulations. *International Journal for Numerical Methods in Fluids*, 25(3):249–263, 1997. (Cited on page 62.)

[217] Chih-Fung Ho, Cheng Chang, Kuen-Hau Lin, and Chao-An Lin. Consistent boundary conditions for 2d and 3d lattice boltzmann simulations. *Computer Modeling in Engineering and Sciences (CMES)*, 44(2):137, 2009. (Cited on pages 62 and 67.)

[218] PA Skordos. Initial and boundary conditions for the lattice Boltzmann method. *Physical Review E*, 48(6):4823, 1993. (Cited on page 62.)

[219] Takaji Inamuro, Masato Yoshino, and Fumimaru Ogino. A non-slip boundary condition for lattice Boltzmann simulations. *Physics of Fluids*, 7(12):2928–2930, 1995. (Cited on page 62.)

[220] David R Noble, Shiyi Chen, John G Georgiadis, and Richard O Buckius. A consistent hydrodynamic boundary condition for the lattice Boltzmann method. *Physics of Fluids (1994-present)*, 7(1):203–209, 1995. (Cited on page 62.)

[221] M'hamed Bouzidi, Mouaouia Firdaouss, and Pierre Lallemand. Momentum transfer of a Boltzmann-lattice fluid with boundaries. *Physics of Fluids*, 13(11):3452–3459, 2001. (Cited on pages 62, 68, and 69.)

[222] Zhaoli Guo, Chuguang Zheng, and Baochang Shi. An extrapolation method for boundary conditions in lattice Boltzmann method. *Physics of Fluids*, 14(6):2007–2010, 2002. (Cited on pages 62 and 72.)

[223] Michael Junk and Zhaoxia Yang. One-point boundary condition for the lattice Boltzmann method. *Phys. Rev. E*, 72:066701, Dec 2005. (Cited on page 62.)

[224] Shiyi Chen, Daniel MartÃnez, and Renwei Mei. On boundary conditions in lattice Boltzmann methods. *Physics of Fluids*, 8(9):2527–2536, 1996. (Cited on page 62.)

[225] Qisu Zou and Xiaoyi He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids*, 9(6):1591, 1997. (Cited on pages 62, 66, and 118.)

[226] Rupert W. Nash, Hywel B. Carver, Miguel O. Bernabeu, James Hetherington, Derek Groen, Timm Krüger, and Peter V. Coveney. Choice of boundary condition for lattice-boltzmann simulation of moderate-reynolds-number flow in complex domains. *Phys. Rev. E*, 89:023303, Feb 2014. (Cited on page 62.)

[227] Dominique d'Humieres and Piere Lallemand. Numerical simulations of hydrodynamics with lattice gas automata in two dimensions. *Complex Systems*, 1(4):599–632, 1987. (Cited on page 64.)

[228] Irina Ginzburg. Consistent lattice Boltzmann schemes for the Brinkman model of porous flow and infinite Chapman-Enskog expansion. *Phys. Rev. E*, 77:066704, Jun 2008. (Cited on page 64.)

[229] Jian Guo Zhou. Axisymmetric lattice boltzmann method revised. *Phys. Rev. E*, 84:036704, Sep 2011. (Cited on page 66.)

[230] Shiladitya Mukherjee and John Abraham. Lattice Boltzmann simulations of two-phase flow with high density ratio in axially symmetric geometry. *Phys. Rev. E*, 75:026701, Feb 2007. (Cited on pages 66 and 177.)

[231] Kannan N. Premnath and John Abraham. Lattice Boltzmann model for axisymmetric multiphase flows. *Phys. Rev. E*, 71:056706, May 2005. (Cited on pages 66 and 177.)

[232] Nils Thürey, Klaus Iglberger, and Ulrich Rüde. Free Surface Flows with Moving and Deforming Objects for LBM. In *Proceedings of Vision, Modeling and Visualization*, volume 2006, pages 193–200, 2006. (Cited on page 66.)

[233] Martin Hecht and Jens Harting. Implementation of on-site velocity boundary conditions for D3Q19 lattice Boltzmann simulations. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(01):P01018, 2010. (Cited on pages 66 and 67.)

[234] Christian Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. Direct Numerical Simulation of the Flow Past an Inclined Flat Plate. International Conference for Mesoscopic Methods in Engineering and Science, ICMMES, July 2012. (Cited on page 69.)

[235] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. Efficient GPU implementation of the linearly interpolated bounce-back boundary condition. *Computers & Mathematics with Applications*, 65(6):936 – 944, 2013. Mesoscopic Methods in Engineering and Science. (Cited on page 70.)

[236] Wei Li, Xiaoming Wei, and Arie Kaufman. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19 (7-8):444–456, 2003. (Cited on page 74.)

[237] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. GPU Cluster for High Performance Computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, SC '04, pages 47–, Washington, DC, USA, 2004. IEEE Computer Society. (Cited on page 74.)

[238] Jonas Tölke. Implementation of a lattice boltzmann kernel using the compute unified device architecture developed by nvidia. *Computing and Visualization in Science*, 13(1):29–39, 2010. (Cited on page 74.)

[239] Adam Lichtl Stephen Jones. GPUs to Mars: Full-Scale Simulation of SpaceX's Mars Rocket Engine. GPU Technology Conference, 2015. (Cited on page 74.)

[240] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *CoRR*, abs/1410.0759, 2014. (Cited on page 74.)

[241] M. Creel and M. Zubair. High Performance Implementation of an Econometrics and Financial Application on GPUs. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1147–1153, Nov 2012. (Cited on page 74.)

[242] Thomas Kroes, Frits H. Post, and Charl P. Botha. Exposure render: An interactive photo-realistic volume rendering framework. *PLoS ONE*, 7 (7):e38586, 07 2012. (Cited on page 74.)

[243] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. A new approach to the lattice boltzmann method for graphics processing units. *Computers & Mathematics with Applications*,

61(12):3628–3638, 2011. Mesoscopic Methods for Engineering and Science - Proceedings of ICMMES-09 Mesoscopic Methods for Engineering and Science. (Cited on pages 86 and 89.)

[244] A Lattice-Boltzmann solver for 3D fluid simulation on GPU. *Simulation Modelling Practice and Theory*, 25(0):163–171, 2012. (Cited on page 89.)

[245] J. Habich, T. Zeiser, G. Hager, and G. Wellein. Performance analysis and optimization strategies for a D3Q19 lattice Boltzmann kernel on NVIDIA GPUs using CUDA. *Advances in Engineering Software*, 42(5): 266 – 272, 2011. PARENG 2009. (Cited on page 89.)

[246] Yu Ye, Kenli Li, Yan Wang, and Tan Deng. Parallel computation of entropic lattice boltzmann method on hybrid cpu-gpu accelerated system. *Computers & Fluids*, 110(0):114–121, 2015. ParCFD 2013. (Cited on page 89.)

[247] Jiri Kraus. Introduction to CUDA-aware MPI and NVIDIA GPUDirect. GPU Technology Conference, 2013. (Cited on page 92.)

[248] J. Kraus, M. Pivanti, S.F. Schifano, R. Tripiccione, and M. Zanella. Benchmarking gpus with a parallel lattice-boltzmann code. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*, pages 160–167, Oct 2013. (Cited on page 92.)

[249] Christian Obrecht, Frederic Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. The TheLMA project: Multi-GPU implementation of the lattice Boltzmann method. *International Journal of High Performance Computing Applications*, 25(3):295–303, 2011. (Cited on page 92.)

[250] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. Multi-gpu implementation of the lattice boltzmann method. *Computers & Mathematics with Applications*, 65(2):252 – 261, 2013. Special Issue on Mesoscopic Methods in Engineering and Science (ICMMES-2010, Edmonton, Canada). (Cited on page 92.)

[251] Arie Kaufman, Zhe Fan, and Kaloian Petkov. Implementing the lattice Boltzmann model on commodity graphics hardware. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(06):P06016, 2009. (Cited on page 92.)

[252] Alan Gray, Alistair Hart, Oliver Henrich, and Kevin Stratford. Scaling soft matter physics to thousands of graphics processing units in parallel. *International Journal of High Performance Computing Applications*, 2015. (Cited on page 92.)

[253] Alan Gray, Alistair Hart, Alan Richardson, and Kevin Stratford. *Lattice Boltzmann for Large-Scale GPU Systems*, pages 167–174. Advances in Parallel Computing. IOS PRESS, 2012. (Cited on pages 92 and 93.)

[254] Kenneth Moreland, Brian Wylie, and Constantine Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics*, PVG '01, pages 85–92. IEEE Press, 2001. (Cited on page 93.)

[255] M. Januszewski and M. Kostur. Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method. *Computer Physics Communications*, 185(9):2350 – 2368, 2014. (Cited on pages 93 and 195.)

[256] Simon McIntosh-Smith, Michael Boulton, Dan Curran, and James Price. On the Performance Portability of Structured Grid Codes on Many-Core Computer Architectures. In JulianMartin Kunkel, Thomas Ludwig, and HansWerner Meuer, editors, *Supercomputing: 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014, Proceedings*, volume 8488 of *Lecture Notes in Computer Science*, pages 53–75. Springer International Publishing, 2014. (Cited on page 100.)

[257] Jiri Kraus. Optimizing a LBM Code For Compute Clusters With Kepler GPUs. GPU Technology Conference, 2013. (Cited on page 106.)

[258] J. Habich, C. Feichtinger, H. Köstler, G. Hager, and G. Wellein. Performance engineering for the lattice boltzmann method on gpgpus: Architectural requirements and performance results. *Computers & Fluids*, 80(0):276–282, 2013. Selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011. (Cited on page 112.)

[259] Nicholas Wilt. *The CUDA Handbook : A Comprehensive Guide to GPU Programming* . Addison-Wesley Professional, 2012. (Cited on page 112.)

[260] Michael Junk, Axel Klar, and Li-Shi Luo. Asymptotic analysis of the lattice Boltzmann equation. *J. Comput. Phys.*, 210:676–704, 2005. (Cited on pages 118 and 125.)

[261] C. K. Aidun, N. G. Triantafillopoulos, and J. D. Benson. Global stability of a lid-driven cavity with throughflow: Flow visualization studies. *Physics of Fluids A: Fluid Dynamics*, 3(9):2081–2091, 1991. (Cited on page 121.)

[262] J. R. Koseff and R. L. Street. The Lid-Driven Cavity Flow: A Synthesis of Qualitative and Quantitative Observations. 106(4):390, 1984. (Cited on page 121.)

[263] U Ghia, K.N Ghia, and C.T Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387–411, 1982. (Cited on pages 121, 122, 123, 125, and 129.)

[264] W a R C Ku and Richard S Hirsh. A Pseudospectral Method for Solution of the Three-Dimensional Incompressible Navier-Stokes Equations. *Journal of Computational Physics*, 70:439–462, 1987. (Cited on pages 121 and 133.)

[265] S. Albensoeder and H. C. Kuhlmann. Accurate three-dimensional lid-driven cavity flow. *Journal of Computational Physics*, 206(2):536–558, 2005. (Cited on pages 121 and 133.)

[266] Y. Ning and K. N. Premnath. Numerical Study of the Properties of the Central Moment Lattice Boltzmann Method. *ArXiv e-prints*, 2012. (Cited on pages 122, 125, 126, and 127.)

[267] A. Montessori, G. Falcucci, P. Prestininzi, M. La Rocca, and S. Succi. Regularized lattice Bhatnagar-Gross-Krook model for two- and three-dimensional cavity flow simulations. *Phys. Rev. E*, 89:053317, May 2014. (Cited on pages 126 and 127.)

[268] E. Aslan, I. Taymaz, and A. C. Benim. Investigation of the Lattice Boltzmann SRT and MRT Stability for Lid Driven Cavity Flow. *International Journal of Materials, Mechanics and Manufacturing*, 2(4):317–324, 2014. (Cited on page 126.)

[269] Ercan Erturk. Discussions on driven cavity flow. *International Journal for Numerical Methods in Fluids*, 60(3):275–294, 2009. (Cited on page 129.)

[270] G. De Vahl Davis. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids*, 3(3):249–264, 1983. (Cited on pages 145 and 148.)

[271] David A. Mayne, Asif S. Usmani, and Martin Crapper. h-adaptive finite element solution of high Rayleigh number thermally driven cavity problem. *International Journal of Numerical Methods for Heat & Fluid Flow*, 10(6):598–615, 2000. (Cited on page 145.)

[272] M. Hortmann, M. Peric, and G. Scheuerer. Finite volume multigrid prediction of laminar natural convection: Benchmark solutions. *International Journal for Numerical Methods in Fluids*, 11(2):189–207, 1990. (Cited on page 145.)

[273] P Le Quere and T.Alziary De Roquefortt. Computation of natural convection in two-dimensional cavities with Chebyshev polynomials. *Journal of Computational Physics*, 57(2):210–228, 1985. (Cited on page 145.)

[274] P. Le Quere. Accurate solutions to the square thermally driven cavity at high Rayleigh number. *Computers & Fluids*, 20(1):29–41, 1991. (Cited on page 145.)

[275] Bert Blocken and Jan Carmeliet. A review of wind-driven rain research in building science. *Journal of Wind Engineering and Industrial Aerodynamics*, 92(13):1079 – 1130, 2004. (Cited on page 153.)

[276] Bert Blocken, Ted Stathopoulos, Jan Carmeliet, and Jan L.M. Hensen. Application of computational fluid dynamics in building performance simulation for the outdoor environment: an overview. *Journal of Building Performance Simulation*, 4(2):157–184, 2011. (Cited on page 153.)

[277] B. Blocken, G. Dezso, J. van Beeck, and J. Carmeliet. Comparison of calculation models for wind-driven rain deposition on building facades. *Atmospheric Environment*, 44(14):1714 – 1725, 2010. (Cited on page 153.)

[278] H. Boyer, J.P. Chabriat, B. Grondin-Perez, C. Tourrand, and J. Brau. Thermal building simulation and computer generation of nodal models. *Building and Environment*, 31(3):207 – 214, 1996. (Cited on page 153.)

[279] Ahmed Cherif Megri and Fariborz Haghighat. Zonal Modeling for Simulating Indoor Environment of Buildings: Review, Recent Developments, and Applications. *HVAC&R Research*, 13(6):887–905, 2007. (Cited on page 153.)

[280] Liangzhu Wang and Qingyan Chen. Validation of a coupled multizone-cfd program for building airflow and contaminant transport simulations. *HVAC&R Research*, 13(2):267–281, 2007. (Cited on page 153.)

[281] Zhiqiang Zhai. Application of Computational Fluid Dynamics in Building Design: Aspects and Trends. *Indoor and Built Environment*, 15(4):305–313, 2006. (Cited on page 153.)

[282] Eliton Fontana, Adriano da Silva, and Viviana Cocco Mariani. Natural convection in a partially open square cavity with internal heat source: An analysis of the opening mass flow. *International Journal of Heat and Mass Transfer*, 54(7-8):1369–1386, 2011. (Cited on page 153.)

[283] Omar S. Asfour and Mohamed B. Gadi. A comparison between CFD and Network models for predicting wind-driven ventilation in buildings. *Building and Environment*, 42(12):4079 – 4085, 2007. Indoor Air 2005 Conference. (Cited on page 153.)

[284] Bin Zhao and Ping Guan. Modeling particle dispersion in personalized ventilated room. *Building and Environment*, 42(3):1099 – 1109, 2007. (Cited on page 153.)

[285] T.X. Qin, Y.C. Guo, C.K. Chan, and W.Y. Lin. Numerical simulation of the spread of smoke in an atrium under fire scenario. *Building and Environment*, 44(1):56 – 65, 2009. (Cited on page 153.)

[286] C. J. Noakes, P. A. Sleigh, A. R. Escombe, and C. B. Beggs. Use of cfd analysis in modifying a tb ward in lima, peru. *Indoor and Built Environment*, 15(1):41–47, 2006. (Cited on page 153.)

[287] Mingang Jin, Wangda Zuo, and Qingyan Chen. Improvements of Fast Fluid Dynamics for Simulating Air Flow in Buildings. *Numerical Heat Transfer, Part B: Fundamentals*, 62(6):419–438, 2012. (Cited on pages 153, 154, 203, and 210.)

[288] Wangda Zuo, Jianjun Hu, and Qingyan Chen. Improvements on FFD modeling by using different numerical schemes. *Numerical Heat Transfer, Part B Fundamentals*, 58:1–16, 2010. (Cited on pages 153, 154, 203, and 210.)

[289] C. Beghein, Y. Jiang, and Q. Y. Chen. Using large eddy simulation to study particle motions in a room. *Indoor Air*, 15(4):281–290, 2005. (Cited on page 154.)

[290] W. Zuo and Q. Chen. Real-time or faster-than-real-time simulation of airflow in buildings. *Indoor Air*, 19(1):33–44, 2009. (Cited on pages 154 and 203.)

[291] Marcel Loomans. *The measurement and simulation of indoor air flow*. University of Eindhoven, 1998. (Cited on page 154.)

[292] M.-F. King, C.J. Noakes, P.A. Sleigh, and M.A. Camargo-Valero. Bioaerosol deposition in single and two-bed hospital rooms: A numerical and experimental study. *Building and Environment*, 59(0):436 – 447, 2013. (Cited on pages 154 and 157.)

[293] Jonathan Koomey. Growth in Data center electricity use 2005 to 2010. Oakland, CA: Analytics Press, August 2011. (Cited on page 163.)

[294] Liam Newcombe, Mark Acton, John Booth, Sophia Flucker, Paul Latham, Steve Strutt, and Robert Tozer. 2012 Best Practices for the EU Code of Conduct on Data Centres. European Commision, Joint Research Centre, 2012. (Cited on page 163.)

[295] Energy Efficiency in Data Centers: Recommendations for Government-Industry Coordination. U.S. Department of Energy and U.S. Environmental Protection Agency, 2008. (Cited on page 163.)

[296] Yogendra Joshi and Pramod Kumar. Introduction to Data Center Energy Flow and Thermal Management. In Yogendra Joshi and Pramod Kumar, editors, *Energy Efficient Thermal Management of Data Centers*, pages 1–38. Springer US, 2012. (Cited on page 163.)

[297] Emad Samadiani, Jeffrey Rambo, and Yogendra Joshi. Numerical modeling of perforated tile flow distribution in a raised-floor data center. *Journal of Electronic Packaging*, 132(2):021002, 2010. (Cited on page 163.)

[298] Jayantha Siriwardana, Saman K Halgamuge, Thomas Scherer, and Wolfgang Schott. Minimizing the thermal impact of computing equipment upgrades in data centers. *Energy and Buildings*, 50:81–92, 2012. (Cited on page 163.)

[299] Kyosung Choo, Renan Manozzo Galante, and Michael M. Ohadi. Energy consumption analysis of a medium-size primary data center in an academic campus. *Energy and Buildings*, 76(0):414 – 421, 2014. (Cited on page 163.)

[300] Dustin W. Demetriou and H. Ezzat Khalifa. Optimization of Enclosed Aisle Data Centers Using Bypass Recirculation. *Journal of Electronic Packaging*, 134:020904, June 2012. (Cited on page 163.)

[301] Vaibhav K Arghode, Vikneshan Sundaralingam, Yogendra Joshi, and Wally Phelps. Thermal characteristics of open and contained data center cold aisle. *Journal of Heat Transfer*, 135(6):061901, 2013. (Cited on page 163.)

[302] S.K. Shrivastava, B.G. Sammakia, R. Schmidt, M. Iyengar, and J.W. VanGilder. Experimental-Numerical Comparison for a High-Density Data Center: Hot Spot Heat Fluxes in Excess of 500 W/FT2. In *Thermal and Thermomechanical Phenomena in Electronics Systems, 2006. ITHERM '06. The Tenth Intersociety Conference on*, pages 402–411, May 2006. (Cited on page 163.)

[303] Vaibhav K Arghode, Pramod Kumar, Yogendra Joshi, Thomas Weiss, and Gary Meyer. Rack level modeling of air flow through perforated tile in a data center. *Journal of Electronic Packaging*, 135(3):030902, 2013. (Cited on page 164.)

[304] Vaibhav K Arghode and Yash Joshi. Modeling strategies for air flow through perforated tiles in a data center. *Components, Packaging and Manufacturing Technology, IEEE Transactions on*, 3(5):800–810, 2013. (Cited on page 164.)

[305] Kenneth Moreland. Diverging Color Maps for Scientific Visualization. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, MiguelL. EncarnaÃ§Ã£o, ClÃ¡udioT. Silva, and Daniel Coming, editors, *Advances in Visual Computing*, volume 5876 of *Lecture Notes in Computer Science*, pages 92–103. Springer Berlin Heidelberg, 2009. (Cited on page 172.)

[306] Shiladitya Mukherjee and John Abraham. Investigations of drop impact on dry walls with a lattice-boltzmann model. *Journal of colloid and interface science*, 312(2):341–354, 2007. (Cited on pages 176 and 177.)

[307] JR Castrejón-Pita, ES Betton, KJ Kubiak, MCT Wilson, and IM Hutchings. The dynamics of the impact and coalescence of droplets on a solid surface. *Biomicrofluidics*, 5(1):014112, 2011. (Cited on page 176.)

[308] R Rioboo, M Marengo, and C Tropea. Time evolution of liquid drop impact onto solid, dry surfaces. *Experiments in Fluids*, 33(1):112–124, 2002. (Cited on page 176.)

[309] S Chandra and CT Avedisian. On the collision of a droplet with a solid surface. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 432, pages 13–41. The Royal Society, 1991. (Cited on page 176.)

[310] Ilker S Bayer and Constantine M Megaridis. Contact angle dynamics in droplets impacting on flat surfaces with different wetting characteristics. *Journal of Fluid Mechanics*, 558:415–449, 2006. (Cited on page 176.)

[311] ZB Yan, KC Toh, F Duan, TN Wong, KF Choo, PK Chan, and YS Chua. Experimental study of impingement spray cooling for high power devices. *Applied Thermal Engineering*, 30(10):1225–1230, 2010. (Cited on page 176.)

[312] Andrés J Díaz and Alfonso Ortega. Investigation of a gas-propelled liquid droplet impinging onto a heated surface. *International Journal of Heat and Mass Transfer*, 67:1181–1190, 2013. (Cited on page 176.)

[313] Mahsa Ebrahim. *Identification of the Impact Regimes of a Liquid Droplet Propelled by a Gas Stream Impinging onto a Heated Surface* . PhD thesis, University of Villanova, forthcoming. (Cited on pages 176, 177, and 179.)

[314] Kirijen Vengadasalam. *Water in diesel fuel filtration using lattice Boltzmann method* . PhD thesis, University of Leeds, forthcoming. (Cited on pages 179 and 180.)

[315] Kenneth S Sutherland and George Chase. *Filters and filtration handbook*. Elsevier, 2011. (Cited on page 179.)

[316] EN 590. Automotive fuels - diesel - requirements and test methods, 1999. British Standards Institution. (Cited on page 179.)

[317] Kenneth Perlin. Course in advanced image synthesis. In *ACM SIGGRAPH Conference*, volume 18, 1984. (Cited on page 180.)

[318] Faith A Morrison. Data correlation for drag coefficient for sphere. *Michigan Technology University, Houghton, MI*, 2010. (Cited on page 182.)

[319] Carl Wieselsberger. New data on the laws of fluid resistance. 1922. (Cited on pages 182 and 183.)

[320] AE Hamielec and JD Raal. Numerical studies of viscous flow around circular cylinders. *Physics of Fluids (1958-1988)*, 12(1):11–17, 1969. (Cited on pages 182 and 183.)

[321] Bengt Fornberg. Steady viscous flow past a circular cylinder up to reynolds number 600. *Journal of Computational Physics*, 61(2):297–320, 1985. (Cited on page 183.)

[322] Herrmann Schlichting, Klaus Gersten, and Klaus Gersten. *Boundary-layer theory*. Springer Science & Business Media, 2000. (Cited on page 183.)

[323] K. Yusuf Billah and Robert H. Scanlan. Resonance, tacoma narrows bridge failure, and undergraduate physics textbooks. *American Journal of Physics*, 59(2):118–124, 1991. (Cited on page 184.)

[324] Julien Favier, Alistair Revell, and Alfredo Pinelli. A Lattice Boltzmann-Immersed Boundary method to simulate the fluid interaction with moving and slender flexible objects. *Journal of Computational Physics*, 261:145–161, 2014. (Cited on page 184.)

[325] Rodrigo Guadarrama-Lara. *Modelling fluid-structure interaction problems with coupled DEM-LBM* . PhD thesis, University of Leeds, forthcoming. (Cited on pages 184 and 185.)

[326] Rodrigo Guadarrama-Lara, Xiaodong Jia, and Michael Fairweather. A meso-scale model for fluid-microstructure interactions. *Procedia Engineering*, 102:1356–1365, 2015. (Cited on page 184.)

[327] Michael L Minion and David L Brown. Performance of under-resolved two-dimensional incompressible flow simulations, ii. *Journal of Computational Physics*, 138(2):734–765, 1997. (Cited on page 186.)

[328] Paul J. Dellar. Bulk and shear viscosities in lattice Boltzmann equations. *Phys. Rev. E*, 64:031203, Aug 2001. (Cited on page 187.)

[329] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1): 55–61, 2000. (Cited on page 190.)

[330] G Gary Wang. Adaptive response surface method using inherited latin hypercube design points. *Journal of Mechanical Design*, 125(2): 210–220, 2003. (Cited on page 190.)

[331] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999. (Cited on pages 203 and 205.)

[332] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable MacCormack method. *Journal of Scientific Computing*, 35(2-3):350–371, 2008. (Cited on pages 203 and 210.)

[333] Mark J Harris, William V Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 92–101. Eurographics Association, 2003. (Cited on page 203.)

[334] Nolan Goodnight. CUDA/OpenGL fluid simulation. *NVIDIA Corporation*, 2007. (Cited on page 208.)

[335] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 2001. (Cited on page 208.)